Міністерство освіти і науки України
Донбаська державна машинобудівна академія

КОМП'ЮТЕРНІ ТЕХНОЛОГІЇ І ПРОГРАМУВАННЯ

Методичні вказівки для самостійної роботи
для студентів спеціальності
123

Краматорськ 2019

Міністерство освіти і науки України
Донбаська державна машинобудівна академія

КОМП'ЮТЕРНІ ТЕХНОЛОГІЇ І ПРОГРАМУВАННЯ

Методичні вказівки для самостійної роботи
для студентів спеціальності
123

Затверджено
на засіданні кафедри
"Автоматизація виробничих процесів"
Протокол № від

Краматорськ 2019

УДК 681.5:681.3

Методичні вказівки для самостійної роботи для студентів спеціальності 123/ Сост.: Е.В.Пищулина - Краматорськ: ДГМА, 2019. – 86 с.

Укладачі: Е.В.Пищулина, ст.викл.

Відп. за випуск Г.П.Клименко

ЗМІСТ

1 ВВЕДЕННЯ В ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ 6
1.1 Структурний підхід в програмуванні
1.2. Концепції об'єктно-орієнтованого програмування 10
1.3. Етапи об'єктно-орієнтованого програмування 14
2 КЛАСИ І ІНКАПСУЛЯЦІЯ 16
2.1 Опис класу 17
2.2 Створення і використання об'єктів 18
2.3. Конструктори і деструкції 20
3 СПАДКОЄМСТВО 23
3.1 Управління доступом похідних класів 26
3.2. Поодиноке спадкоємство 32
4 ПОЛІМОРФІЗМ 35
4.1 Перевантаження функцій 36
4.2 Вибір екземпляра функції 36
4.3 Перевантаження стандартних операцій 38
4.4 Віртуальні функції 50
5 ОСНОВИ ОРГАНІЗАЦІЇ ВВЕДЕННЯ-ВИВОДУ 54
5.1. Класифікація засобів введення-виводу 54
5.2 Принципи роботи з потоками і файлами 55
5.3. Форматоване уведення-виведення базових типів 62
5.4 Маніпулятори 67
5.5 Прапори стану потоку 69
5.6 Зв'язування потоків 70
6 ДОДАТКОВІ МОЖЛИВОСТІ ВВЕДЕННЯ-ВИВОДУ 71
6.1 Форматоване уведення-виведення призначених для користувача типів 72
6.2 Файлове уведення-виведення 74
6.3 Неформатоване уведення-виведення 78
6.4 Обмін з рядком в пам'яті 82
6.5 Використання бібліотеки *stdio* 84
Использование библиотеки *stdio* 84
7 ШАБЛОНИ 95
7.1 ФУНКЦІЇ 96
7.2. Класи 99
7.3. Стандартна бібліотека шаблонів 100
Література 105

1 ВВЕДЕННЯ В ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ

Нині об'єктно-орієнтоване програмування (ТОП) є домінуючим стилем при створенні великих програм. Розглянемо основні етапи еволюції структурного підходу в програмуванні, які допомагають краще зрозуміти взаємозв'язок структурного підходу, модульного програмування і ТОП.

1.1 Структурний підхід в програмуванні

Перші програми для цифрових обчислювальних машин рідко перевищували об'єм 1 кбайт. З тієї пори істотно змінилася архітектура і технічні характеристики програмованих обчислювальних засобів (ВС), надзвичайно знизилася вартість зберігання, пересилки і обробки 1 байта інформації. Об'єми використовуваних програм і програмних систем вимірюються не лише десятками кілобайт, але і сотнями мегабайтів. В той же час питома вартість створення програм (нормована об'ємом програми) до останнього часу мінялася мало. Більше того, з ростом об'єму програми питома вартість її створення могла нелінійно зростати. Було виявлено, що час створення складних програм пропорційний квадрату або навіть кубу об'єму програм. Тому не дивно, що одним з основних чинників, що визначають розвиток технології програмування, є зниження вартості проектування і створення програмних продуктів (ПП), або боротьба із складністю програмування.

Іншими найважливішими чинниками, що впливають на еволюцію методів проектування і створення ПП, є:

- зміна архітектури обчислювальних засобів (ВС) в інтересах підвищення продуктивності, надійності і комунікативності;
- спрощення взаємодії користувачів зі ВС і інтелектуалізація ВС.

Дія двох останніх чинників, як правило, зв'язана з ростом складності програмного забезпечення ВС. ***Складність*** представляє невід'ємну властивість програмування і програм, яке *проявляється* в часі і вартості
програм, в об'ємі або довжині тексту програми, характеристиках її логічної структури, що задається операторами передачі управління (галуження, цикли, виклики підпрограм та ін.).

Можна виділити 5 наступних джерел складності програмування :

1) вирішувана задача;
2) мова програмування;
3) середовище виконання програми;
4) технологічний процес колективної розробки і створення ПП;
5) прагнення до універсальності і ефективності алгоритмів і типів даних.

Від властивості складності не можна позбавитися, але можна змінювати характеристики його прояву шляхом управління або організації.

У програмуванні широко використовується фундаментальний принцип управління складними системами, який відомий людині з глибокої старовини — *devide et impera* (розділяй і володарюй, лат.) і широко їм застосовується при розробці і проектуванні будь-яких складних технічних систем. Згідно першої частини цього принципу, при проектуванні складної програмної системи проводиться ***алгоритмічна декомпозиція*** вирішуваної задачі.

Метою декомпозиції є представлення системи, що розробляється, у вигляді взаємодіючих невеликих підсистем (модулів або блоків), кожну з яких можна відлагодити (випробувати) незалежно від інших. В цьому випадку при розробці системи, розділеної на «великі» елементи або підсистеми, необхідно тримати в думці інформацію про набагато менше число деталей, чим у відсутність такого розділення. Разом з терміном *декомпозиція,* також використовується термін ***структуризація*** проблеми, завдання або програми. Ідеї розділення програм на відносно самостійні великі частини, що реалізовують певні процедури і функції і утворюють певну ієрархію взаємозв'язків, знайшли відображення в ***структурному підході*** *до* розробки і створення програмних засобів. У програмуванні структурний підхід з'явився з виникненням перших підпрограм, процедур і функцій, написаних в так званому ***процедурно-орієнтованому стилі.*** Цей стиль спирається на просте правило: визначити змінні і константи, які знадобиться зберігати в пам'яті комп'ютера, і описати або використати алгоритм їх обробки.Наряду с термином *декомпозиция,* также используется термин ***структуризация*** проблемы, задачи или программы. Теоретичне оформлення структурний підхід отримав у кінці 60-х — початку 70-х років в роботах Э. Дейкстры, А.П. Єршова, Э. Йодана, Н. Вирта, Э. Брукса і інших теоретиків і практиків програмування. Слід зазначити поява структурного програмування, в якому знайшла певне відображення ідея впорядковування структури програми. ***Структурне програмування*** орієнтує на складання програм, структура яких близька до «дерева» операторів або блоків. Використання структури типу «дерево» в якості своєрідного еталону пояснюється тим, що це проста для аналізу і реалізації структура. Подальший розвиток структурного підходу привів до ***модульного програмування.*** Воно передбачає декомпозицію прикладного завдання у вигляді ієрархії взаємодіючих модулів або програм. Модуль, що містить дані і процедури їх обробки, зручний для автономної розробки і створення. Спеціалізація модулів по видах обробки і наявність в них цих певних типів — це властивості, які відбивають генетичний зв'язок модульного програмування і ТОП. Найважливішими інструментами виробників ПП, в яких знаходять відображення практично усі аспекти еволюції, є мови програмування. Мова програмування спочатку орієнтована на комп'ютер і містить набір типів даних, операторів, операцій, функцій і інших операційних

одиниць мови, які досить просто можуть бути переведені в інструкції (команди) по управлінню апаратним і програмним забезпеченням комп'ютера. При цьому бажано максимізувати ефективність трансляції пропозицій мови в машинні коди в сенсі мінімізації необхідної пам'яті, часу виконання програми і вартості створення транслятора. В той же час мова програмування орієнтована на програміста і надає засоби для моделювання об'єктів, їх властивостей і поведінки при рішенні прикладних завдань в деякій предметній області у вигляді програм. Розвиток мов у напрямі підвищення ефективності складання застосовних програм привів до розділення мов по наступних рівнях.

1. Низький рівень (машинно-орієнтовані мови — мови асемблера).

2. Високий рівень (процедурно-орієнтовані мови: FORTRAN, ALGOL, PL/I, Pascal).Высокий уровень (процедурно-ориентированные языки: FORTRAN, ALGOL, PL/I, Pascal).

3. Рівень вирішуваної задачі (проблемно-орієнтовані мови — GPS, SQL).Уровень решаемой задачи (проблемно-ориентированные языки — GPS, SQL).

Введення ***типів даних*** позначило ще один напрям розвитку технології програмування. Типізація даних призначена як для полегшення складання програм, так і для автоматизації виявлення помилок використання даних у вигляді операндів і фактичних параметрів при виклику функцій. Використання ***структурних типів даних*** дозволяє, по-перше, спростити роботу алгоритмиста при зіставленні структур даних прикладного завдання і даних, що обробляються процедурами і функціями програмних модулів, і, по-друге, скоротити об'єм рутинної роботи програміста при кодуванні алгоритму обробки.

Результатом узагальнення поняття «Тип даних» є класи об'єктів (C++) або об'єктні типи (Pascal), які можуть містити в якості елементів не лише дані певного типу, але і методи їх обробки (функції і процедури). Таким чином, у міру розвитку технології програмування і в програмах, і в типах даних все адекватніше відбивалася структура вирішуваної прикладної задачі і здійснювалася відповідна інтеграція даних і програм в модулях. Одночасно з цим мови програмування поповнилися засобами, необхідними для опису подібних структур. Розвиток ідей абстрагування і модульності привів до появи в програмуванні об'єктного підходу. Людина мислить образами або об'єктами, він знає їх властивості і маніпулює ними, узгодившись з певними подіями. Ще древнім грекам належить думка про те, що світ можна розглядати у вигляді об'єктів і подій. Рене Декарт відмічав, що люди зазвичай мають об'єктно-орієнтований погляд на світ. Так, подумавши про телефонний апарат, людина може представити не лише його форму і колір, але і можливість подзвонити, характер звучання дзвінка і ряд

інших властивостей (залежно від його спонукань, технічних знань, фантазії).Человек мыслит образами или объектами, он знает их свойства и манипулирует ими, сообразуясь с определенными событиями. Еще древним грекам принадлежит мысль о том, что мир можно рассматривать в виде объектов и событий. Рене Декарт отмечал, что люди обычно имеют объектно-ориентированный взгляд на мир. Так, подумав о телефонном аппарате, человек может представить не только его форму и цвет, но и возможность позвонить, характер звучания звонка и ряд других свойств (в зависимости от его побуждений, технических знаний, фантазии).

Мова програмування дозволяє описати властивості модельованих об'єктів і порядок маніпуляції з об'єктами або порядок їх взаємодії, узгодившись з умовами вирішуваної задачі. Перші мови програмування орієнтувалися, з одного боку, на математичні об'єкти, а з іншої — на певну модель обчислювача (ЕОМ з архітектурою фон Неймана). Тому вони містили такі конструкції як змінна, константа, процедура, функція, формальні і фактичні параметри. Програмісти представляли свої програми у вигляді взаємодіючих підпрограм (блоків, процедур, функцій) і модулів. Характер програмування був процедурно-орієнтованим, оскільки первинна увага приділялася послідовностям дій з даними (процедурам). Відповідно такі мови програмування, як FORTRAN, ALGOL, COBOL, PL — 1, З називають ***процедурно-орієнтованими.*** условиями решаемой задачи. Дані і підпрограми об'єднувалися в модулі відповідно до логіки проектувальників, що створюють складні програмні системи для певних сфер їх застосування. Логіка інтеграції в модулі визначалася рядом чинників, серед яких слід зазначити властивості предметної області : дані і підпрограми їх обробки, що відповідають певному класу об'єктів предметної області, об'єднувалися в модуль. Наприклад, модуль обробки рядків містив підпрограми виконання основних операцій з рядками: об'єднання, порівняння, копіювання, знаходження входження підрядка в рядок, обчислення довжини рядка.Данные и подпрограммы объединялись в модули в соответствии с логикой проектировщиков, создающих сложные программные системы для определенных областей их применения. Логика интеграции в модули определялась рядом факторов, среди которых следует отметить свойства предметной области: данные и подпрограммы их обработки, соответствующие определенному классу объектов предметной области, объединялись в модуль. Например, модуль обработки строк содержал подпрограммы выполнения основных операций со строками: объединения, сравнения, копирования, нахождения вхождения подстроки в строку, вычисления длины строки.

Розвитком ідеї модульного програмування є зіставлення об'єктам предметної області (модельованим об'єктам) програмних конструкцій, що називаються ***об'єктами,*** *об'єктними типами* або *класами* (моделюючими об'єктами). Моделюючі об'єкти містять дані і підпрограми, які описують

властивості модельованих об'єктів. Так, дані можуть відбивати признаковые або кількісні властивості (маса, довжина, потужність, ціна, наявність, видимість і т. п.), а підпрограми відбивають поведінкові або операційні властивості (змінити масу, вичислити потужність, встановити ціну, перевірити наявність, зробити видимим і т. п.). Таким чином, при об'єктному підході інтеграція даних і процедур їх обробки визначається структурою предметної області, т.е набором модельованих об'єктів, їх взаємозв'язком або взаємодією у рамках вирішуваної задачі.Развитием идеи модульного программирования является сопоставление объектам предметной области (моделируемым объектам) программных конструкций, называемых ***объектами,*** *объектными типами* или *классами* (моделирующими объектами)..

Модельований об'єкт завжди представляється людині чимось єдиним, цілісним, хоча може складатися з частин або інших об'єктів. Наприклад, телефонний апарат складається з трубки, дроту, корпусу з номеронабирачем, а трубка містить мікрофон, динамік і дроти і т. д. Цілісне представлення об'єкту у вигляді взаємозв'язаної сукупності його властивостей або компонентів є базовим принципом об'єктного підходу. Об'єктний підхід почав розвиватися в програмуванні з другої половини 60-х років (мова Simula - 67) і знайшов своє відображення в мовах Smalltalk, CLOS, Ada і у ряді інших. Ці мови називаються об'єктними. Ієрархічна класифікація об'єктів і спадкоємство властивостей є відправними ідеями об'єктно-орієнтованого підходу, що з'явився на початку 80-х років. Одній з причин порівняно повільного становлення об'єктно-орієнтованого стилю програмування являється його істотна відмінність від пануючого процедурно-орієнтованого стилю.

1.2. Концепції об'єктно-орієнтованого програмування

ТОП є третім великим етапом (після структурного і модульного програмування) в процесі розвитку структурного підходу. Створювані в середині 70-х років великі програмні системи продемонстрували, що у рамках процедурно-орієнтованого стилю використання структурного підходу не дає бажаного ефекту. У міру збільшення числа компонентів в створюваних програмних системах число помилок, пов'язаних з неправильним використанням процедур і некоректним обліком взаємозв'язків між компонентами, стало нелінійно рости. Терміни введення в експлуатацію цих систем постійно зривалися. Зменшити число подібних помилок і спростити їх виявлення могла дозволити алгоритмічна декомпозиція, що орієнтується на «природні» елементи (компоненти або об'єкти) простору вирішуваної задачі. В цьому випадку на етапі кодування і відладки спрощувалося зіставлення конструкцій програмістів з модельованими об'єктами. - Таку

декомпозицію називатимемо об'єктно-орієнтованим аналізом простору вирішуваної задачі або предметної області. Для опису результатів об'єктно-орієнтованого аналізу і подальшого програмного синтезу потрібні адекватні мовні засоби, які побудовані на певних принципах. Розглянемо їх.

сновним поняттям ТОП є *об'єкт* або ***клас*** **в** С++, який можна розглядати з двох позицій. По-перше, з позиції предметної області : *клас* відповідає певному характерному (типовому) об'єкту цієї області. По-друге, з позиції технології програмування, що реалізовує цю відповідність : «клас» в ТОП — це певна програмна структура, яка має три найважливіші властивості :Основным понятием ООП является *объект* или ***класс*** **в** С++, который можно рассматривать с двух позиций. Во-первых, с позиции предметной области: *класс* соответствует определенному характерному (типичному) объекту этой области. Во-вторых, с позиции технологии программирования, реализующей это соответствие: «класс» в ООП — это определенная программная структура, которая обладает тремя важнейшими свойствами:

- інкапсуляції;
- спадкоємство;
- поліморфізму.

Ці властивості використовуються програмістом, а забезпечуються об'єктно-орієнтованою мовою програмування (транслятором, що реалізовує це мова). Вони дозволяють адекватно відбивати структуру предметної області.

Деякі автори називають ці властивості об'єктів принципами ТОП. Але, мабуть, це занадто вузьке розуміння суті ТОП, оскільки осторонь залишилися зовсім не деталі:

- об'єктно-орієнтований аналіз предметної області і парадигма ТОП;
- створення і знищення об'єктів
- принципи організації взаємодії об'єктів.

Об'єкти і класи

Концепція об'єктів призначена для моделювання (відображення) понять предметної області у вигляді програмних одиниць, що об'єднують в собі атрибути і поведінку (стан і функціонування) відповідних об'єктів (сутностей) предметної області.

Клас об'єктів є програмною структурою, в якій дані і функції утворюють єдине ціле і відбивають властивості і поведінку цього цілого у рамках модельованої предметної області. На відміну від модуля, в якому на склад даних і функцій накладається менше смислових обмежень, в об'єкті є присутніми тільки ті дані і функції, Kqropbie потрібні для опису властивостей і поведінки об'єкту певного типу.

Об'єкти виділяються в процесі аналізу предметної області з використанням ідей абстрагування від несуттєвого і класифікації споріднених об'єктів. Результатом об'єктно-орієнтованого аналізу є ***класи об'єктів,*** які є присутніми або в перспективі можуть бути присутніми в просторі вирішуваної задачі і

утворюють ***ієрархії класів,*** *спадкоємство властивостей, що* представляється у вигляді *дерев..*

Наприклад, на основі аналізу авіаційної техніки можна виділити клас об'єктів *Літак.* При цьому ми абстрагувалися від таких властивостей як: форма крил, довжина флюзеляжа, використовувані матеріали конструкцій, прихильність крил і т. п. До числа основних властивостей класу об'єктів *Літак* можна віднести: швидкість польоту, розмах крил, тип двигуна, вантажопідйомність, висота польоту, функціональне призначення і т. д

Для прикладу приведемо також ієрархію класів *Авіамоделі,* яку можна представити в наступному виді :Для примера приведем также иерархию классов *Авиамодели,* которую можно представить в следующем виде:

Авіамодель:

- ♦ кордові моделі:
  - перегони;
  - швидкісні;
  - пілотажні;
  - повітряного бою;
  - копії;
- ♦ моделі вільного польоту :
  - рухові:
    - — таймерні;
    - — керовані по радіо;
    - — копії;
  - резиномоторные;
  - планерні.

Клас об'єктів характеризується унікальним набором властивостей і йому привласнюється унікальне ім'я, як і будь-якому типу даних. Як змінні програми використовуються ***об'єкти*** певного класу. Створювані об'єкти, навіть одного класу, можуть відрізнятися значеннями (мірою прояву) властивостей і, звичайно, повинні відрізнятися іменами.Класс объектов характеризуется уникальным набором свойств и ему присваивается уникальное имя, как и любому типу данных. В качестве переменных программы используются ***объекты*** определенного класса. Создаваемые объекты, даже одного класса, могут отличаться значениями (степенью проявления) свойств и, конечно, должны отличаться именами.

Інкапсуляція властивостей об'єктів

Інкапсуляція (дослівно — «зміст в оболонці») є об'єднанням і локалізацією у рамках об'єкту, як єдиного цілого, даних і функцій, оброблювальних ці дані. В сукупності вони відбивають властивості об'єкту.***Инкапсуляция*** (дословно — «содержание в оболочке») представляет собой объединение и локализацию в рамках объекта, как единого целого, данных и функций, обрабатывающих эти данные. В совокупности они отражают свойства объекта.

У С++ дані класу і об'єкту називаються *елементами даних* або *полями,* а функції — *методами* або *елементами-функціями*В С++ данные класса и объекта называются *элементами данных* или *полями,* а функции — *методами* или *элементами-функциями,*

Доступ до полів і методів об'єкту здійснюється через ім'я об'єкту і відповідні імена полів і методів за допомогою операцій вибору «». і «->». Це дозволяє в максимальному ступені ізолювати зміст об'єкту від зовнішнього оточення, тобто обмежити і наочно контролювати доступ до елементів об'єкту. В результаті заміна або модифікація полів і методів, інкапсульованих в об'єкт, як правило, не спричиняє за собою погано контрольованих наслідків для програми в цілому. При необхідності вказівки імені об'єкту в тілі опису цього об'єкту в С++ використовується зарезервоване слово ***this,*** яке у рамках об'єкту є спеціальним синонімом імені об'єкту — покажчиком на об'єкт.

Відмітимо, що іменування класів, елементів даних і методів має велике значення в ТОП. Назви повинні або співпадати з назвами, що використовуються в предметній області, або ясно відбивати сенс або призначення (функціональність) іменованого класу, поля або методу. При цьому не слід боятися довгих імен — витрати на написання сторицею окупляться при відладці і супроводі прот.е.е.ного продукту. Текст подібної програми стає зрозумілим без особливих коментарів. Додатковим коштом т.е.е. е.ения доступу до даних і методів являється опис елементів класів за допомогою специфікаторів ***private, protected і public,*** які визначають три відповідні рівні доступу до компонентів класу : власного, захищеного і загальнодоступного.

Для розширення доступу до елементів даних, що мають атрибути *private* або *protected,* в класі можна реалізувати з атрибутом *public* спеціальні методи доступу до власних і захищених елементів даних.Для расширения доступа к элементам данных, имеющим атрибуты *private* или *protected,* в классе можно реализовать с атрибутом *public* специальные методы доступа к собственным и защищенным элементам данных.

Гнучке розмежування доступу дозволяє зменшити небажані (безконтрольні) спотворення властивостей об'єкту або несанкціоноване використання властивостей класів.

Хорошим стилем в ТОП вважається організація доступу до елементів даних за допомогою функцій або методів без використання оператора привласнення. Звичайно, це положення не має бути догмою, але випадки відходу від нього мають бути добре обдумані.

Спадкоємство властивостей

Спадкоємство є властивість класів породжувати своїх нащадків і наслідувати властивості (елементи даних і методи) своїх батьків. Клас-нащадок автоматично наслідує від батька усі *елементи даних* і *методи,* а також може містити нові елементи даних і методи і навіть замінювати (перекривати, перевизначати) методи батька або. модифікувати (доповнювати) їх. Спадкоємство в ТОП дозволяє адекватно відбивати *споріднені стосунки* об'єктів предметної

області. Якщо клас В має усі властивості класу А і ще має додаткові властивості, то клас А називається ***базовим*** (батьківським), а клас В називається ***спадкоємцем*** класу А. У С++ можливе *поодиноке* (з одним батьком) і *множинне* (з декількома батьками) спадкоємство.

Споріднені стосунки, або ***стосунки включення властивостей класів,*** можуть відбиватися не лише за допомогою спадкоємства, але і шляхом інкапсуляції в класі в якості елементів даних інших класів. Наприклад, стек може бути побудований, як спадкоємець одного класу — «елемент стека». Можлива побудова стека, як класу, що містить в якості елементу даних об'єкт класу «

Властивість спадкоємства спрощує модифікацію властивостей класів, забезпечує ТОП виняткову гнучкість і скорочує витрати на написання нових класів на основі старих (батьків). Програміст зазвичай визначає базовий клас, що має найбільш загальні властивості, а потім створює послідовність нащадків, які мають свої специфічні властивості. В результаті виходить ієрархія спадкоємства властивостей класів.

Базові класи, або корені подібних дерев, мають такі абстрактні властивості, що часто безпосередньо в програмах не використовуються, а потрібні «усього лише» для породження необхідних класів. Правильний вибір кореня забезпечує зручність «вирощування» дерева, тобто простоту розвитку бібліотеки класів Застосування правила ТОП «***наслідуй і змінюй властивості об'єктів»*** добре узгоджується з поетапним підходом до розробки і створення великих програмних систем..

Приклади споріднених класів : Координати на екрані -> Кольорова точка ~> Пряма -« Прямокутник. Тут напрям стрілки вказує порядок спадкоємства властивостей класів.

Поліморфізм поведінкових властивостей об'єктів

Поліморфізм часто визначають як властивість об'єктів-родичів по-різному здійснювати однотипні (і навіть однаково пойменовані) дії, тобто однотипні дії у безлічі споріднених класів мають безліч різних форм. Наприклад, метод «намалювати на екрані» повинен по-різному реалізовуватися для споріднених класів «точка», «пряма», що «ламається», «прямокутник». У ТОП дії або поведінка об'єкту визначається набором методів. Змінюючи алгоритм методу в нащадках класу, програміст надає спадкоємцям специфічні поведінкові властивості. Для зміни методу необхідно перекрити його в нащадку, тобто оголосити в нащадку однойменний метод і реалізувати в нім потрібні дії, що відбивають специфіку нащадка.

Властивість поліморфізму реалізується не лише в механізмі заміщення (перекриття) однойменних методів при спадкоємстві властивостей, але і в механізмі віртуалізації методів, або пізньому зв'язуванні методів. Якщо заміщення методу реалізується на етапі компіляції ***(раннє зв'язування*** об'єкту з методом), то заміщення оголошеного в описі класу віртуальним (virtual) відбувається на етапі вьшолнения ***(пізнє зв'язування).*** Але нічого не дається

дарма: віртуалізація вимагає додаткових ресурсів пам'яті і часу. Віртуальні методи виконуються повільніше, із-за необхідності додаткових дій із зв'язування.

Створення і знищення об'єктів

Опис класу в ТОП є деяка програмна структура, яку використовують при створенні об'єктів, що відрізняються іменами і окремими властивостями. Створення і видалення об'єктів здійснюється за допомогою спеціальних методів, які називаються *конструкторами (constructor)* і *деструкціями (destructor)* відповідно.

Конструктор класу створює і ініціалізував об'єкти, а також може виконувати підготовку механізму пізнього зв'язування, необхідного для використання віртуальних функцій. ***Деструкція*** класу виконує дії, що завершують роботу з об'єктом, наприклад: звільняє динамічну пам'ять, відновлює екран, закриває файлові змінні.

Об'єкти можна розміщувати і в динамічній пам'яті. Для цього потрібні покажчики на ці об'єкти.

Взаємодія об'єктів і повідомлення

Відношення спорідненості або включення властивостей — це тільки один з багатьох типів стосунків між класами. Ще одним найважливішим типом стосунків між класами і між об'єктами являються стосунки взаємодії або відношення «клієнт-сервер». Стосунки спорідненості також мають на увазі певне відношення взаємодії, обмежене властивістю інкапсуляції.

Завдяки інкапсуляції об'єкти так добре ізолюються один від одного, що необхідно спеціально піклується про механізм їх взаємодії в програмі. Це стосується, в першу чергу, незалежних об'єктів, а не тих, які занурені в інші об'єкти у вигляді елементів цих цих об'єктів. Але і в останньому випадку взаємодія може бути обмежено синтаксисом мови і угодами про видимість (доступності) елементів даних і методів споріднених об'єктів і інкапсульованих об'єктів. Тому для його розширення вимагається спеціально піклуватися про це.

Часто зв'язок встановлюють за допомогою ***покажчиків,*** що робить програму схожою на структуру даних в динамічній пам'яті. Прикладами такого способу є програми, написані за допомогою бібліотек Turbo Vision і Object Windows. Основним змістом програми є опис класів і сукупності взаємодіючих об'єктів. У бібліотеках класів для доступу об'єкту до самого собі (усередині себе) використовується зарезервоване слово ***this,*** що є синонімом імені об'єкту, в контексті якого воно використовується.

Другим способом обміну інформацією між об'єктами є передача через ***глобальну змінну*** (буфер повідомлень), якій один об'єкт привласнює значення, а інший — прочитує. Цей спосіб простий в реалізації, але вимагає ретельного контролю обміну повідомленнями з боку програміста, оскільки структура взаємодії більше прихована від програміста« чим при першому способі.

У ТОП термін «послати об'єкту повідомленням часто трактують, як виклик методу об'єкту, якому посилається повідомлення, визначуване цим викликом.

1.3. Етапи об'єктно-орієнтованого програмування

Програма, що вирішує деяку задачу, містить в собі опис частини світу, що відноситься до цього завдання, або певної предметної області. Опис дійсності у формі взаємодіючих об'єктів, мабуть, природніше, ніж у формі ієрархії підпрограм, і тому полегшує програмне моделювання в деякій предметній області.

В процесі програмування в об'єктно-орієнтованому стилі можна виділити наступні етапи:

1. Визначення основних понять предметної області і класів, що відповідають їм, що мають певні властивості (можливі стани і дії). Обгрунтування варіантів створення об'єктів.

2. Визначення або формулювання принципів взаємодії класів і взаємодії об'єктів у рамках програмної системи.

3. Встановлення ієрархії взаємозв'язку властивостей споріднених класів.

4. Реалізація ієрархії класів за допомогою механізмів інкапсуляції, спадкоємства і поліморфізму.

5. Для кожного класу реалізація повного набору методів для управління властивостями об'єктів.

В результаті буде створено об'єктно-орієнтоване середовище, або бібліотеку класів, що дозволяє вирішувати завдання моделювання в певній предметній області.

Перші три етапи і є об'єктно-орієнтованим аналізом предметної області.

На закінчення розділу відмітимо, що ТОП має наступні достоїнства:

• використання природніших понять і імен з повсякденного життя або предметної області;

• простота введення нових понять на основі старих;

• відображення у бібліотеці класів найбільш загальних властивостей і стосунків між об'єктами модельованої предметної області;

• природність відображення простору вирішуваної задачі в простір об'єктів програми;

• простота внесення змін до класів, об'єктів і програми в цілому;

• поліморфізм класів спрощує складання і розуміння програм;

• локалізація властивостей і поведінки на основі інкапсуляції і розмежування доступу спрощують розуміння структури програми і її відладку;

• передача параметрів в конструкторах об'єктів підвищує гнучкість створюваних програмних систем.

14. Назвіть достоїнства застосування ТОП.
15. Якими чинниками визначається доцільність застосування технології ТОП?
16. За допомогою яких методів виконується створення і видалення об'єктів?
17. Назвіть способи встановлення зв'язку і обміну інформацією між об'єктами.

18. У чому суть спадкоємства?
Які різновиди спадкоємства є в мовах програмування?
19. Запропонуйте варіант ієрархії класів в області транспортних засобів.
20. Запропонуйте варіант ієрархії класів в області обчислювальної техніки.
21. Запропонуйте варіант ієрархії класів у тваринному світі.

2 КЛАСИ І ІНКАПСУЛЯЦІЯ

У цьому розділі розглядається опис і використання класів в C++ і інкапсуляція як одна з найважливіших властивостей класів. ***Інкапсуляція*** є об'єднанням цих і оброблювальних їх функцій в одному класі як типі об'єктів. Метою інкапсуляції є автономність модулів, що дозволяє локалізувати наслідки зміни структур даних конкретного модуля. Для інших модулів зміни будуть непомітні. Інкапсуляція при описі класу забезпечується завданням специфікаторів доступу для компонентів класу.В данном разделе рассматривается описание и использование классов в C++ и инкапсуляция как одно из важнейших свойств классов..

2.1 Опис класу

Клас є абстрактним типом (визначуваний програмістом), який створюється на основі існуючих типів. Окремий клас вклюает в себе дані, що називаються *елементами даних,* і функції, що називаються *методами.* Елементи даних і методи є рівноправними компонентами класу.

Опис класу має наступний формат:

class | struct | union ім'я_класу {список компонентів};class | struct | union имя_класса {список компонентов};

У цьому описі:

- одне з ключових слів *class, struct* або *union* вказує на початок опису класу, визначає використовуваний за умовчанням статус доступу до компонент класу, а також впливає на можливості спадкоємства властивостей цього класу;одно из ключевых слов *class, struct* или *union* указывает на начало описания класса, определяет используемый по умолчанию статус доступа к компонентам класса, а также влияет на возможности наследования свойств этого класса;
- *ім'я_класу* — ідентифікатор;*имя_класса* — идентификатор;
- *список компонентів* — перелік оголошень елементів даних і описів методів класу.*список компонентов* — перечень объявлений элементов данных и описаний методов класса.

Відповідно до синтаксису мови C++ кожен компонент класу має статус доступу. Таких статусу три: загальнодоступний, власний і захищений. Як ***специфікатори доступу*** використовуються ключові слова ***public*** *(*загальнодоступний*),* ***private*** (власний), ***protected*** (захищений), за якими йде двокрапка. Дія специфікатора на компоненти класу починається з моменту його

написання до нового специфікатора або до кінця опису класу.В соответствии с синтаксисом языка C++ каждый компонент класса обладает статусом доступа. Таких статуса три: общедоступный, собственный и защищенный. В ***качестве спецификаторов доступа*** используются ключевые слова ***public*** *(*общедоступный*),* ***private*** (собственный), ***protected*** (защищенный), за которыми следует двоеточие. Действие спецификатора на компоненты класса начинается с момента его написания до нового спецификатора или до конца описания класса.

Специфікатор доступу *private* використовується в основному для завдання статусу доступу до елементів даних класу, що дозволяє розв'язати проблему захисту даних. Власні дані є доступними тільки для методів свого класу. Специфікатор доступу *public* часто використовується для завдання загальнодоступного доступу методам класу, які організовують зв'язок об'єкту цього класу із зовнішнім світом. Статус захищений *(protected)* використовується в класах при застосуванні механізму спадкоємства класів. За відсутності спадкоємства специфікатор *protected* эквива-лентен специфікатору *private.*

Усі компоненти класу, введені за допомогою ключових слів ***struct* і *union,*** є за умовчанням *загальнодоступними,* а за допомогою ключового слова ***class*** — *власними,* тобто недоступними для звернень ззовні. Для зміни статусу компонентів класів, описаних за допомогою ключових слів *class* і *struct,* необхідно використати специфікатори доступу. Класи, описані за допомогою ключового слова *union,* не можуть використовуватися як базові класи при спадкоємстві. Крім того, у об'єктів, оголошених на основі подібного класу, для елементів даних виділяється загальне місце в пам'яті. Статус компонентів у таких класів змінити не можна.Все компоненты класса, введенные с помощью ключевых слов ***struct* и *union,*** являются по умолчанию *общедоступными,* а с помощью ключевого слова ***class*** — *собственными,* т.е. недоступными для обращений извне.

Приклад 1. Опис класу.**Пример 1.** Описание класса.

Розглянемо опис класу *Sum,* який забезпечує підсумовування двох цілих чисел. Компонентами класу є: два доданків *x* і *y,* сума *s* і методи *getx(), gety(), summa(),* які призначені для ініціалізації компонентних даних *x* і у, а також для отримання і виводу на екран комп'ютера результату.

Для того, щоб в класі *Sum* елементи даних визначити *власними,* а методи *загальнодоступними,* опис класу можна записати таким чином: class Sum

```
{
    int x, y, s;            // за умовчанням privateint x,y,s;            // по умолчанию private
    public:
    void getx(int xl){  x=xl;  } // опис методуvoid getx(int xl) { x=xl; } // описание метода
    void gety(int yl){ y=y1; } // опис методу
    void summa();          // прототип методу
};
```

```
// Опис методу :
void Sum::summa()
   {
      s=x+y;
      cout << "Сума  ″  << x << " і " << y << " рівна :"  <<  s;cout << "\n
Сумма ″<< x << " и " << y << " равна:" << s;
   }
```

У приведеному класі *Sum* компонентні дані *x, y* і s є *власними* за умовчанням, а методи *getx(), gety()* і *summa() загальнодоступними.*

У описі класу методи *getx()* і *gety()* представлені повністю, а метод *summa()* — своїм прототипом. Методи *getx()* і *gety()* забезпечують введення компонентних даних *x* і *y* відповідно, оскільки доступ до елементів даних класу можна забезпечити тільки за допомогою методів класу *Sum.* Іншим функціям компонентні дані недоступні, оскільки дані *x* і *y* мають статус доступу *private.* Описи методів *getx()* і *gety()* розміщені усередині класу. Така форма опису робить метод вбудованим *(inline)* за умовчанням. В цьому випадку тіло методу буде розміщено в самому класі у вигляді макророзширення. Цим досягається заощадження часу реалізації методу при виклику функції і виході з неї. Цю форму опису слід використати лише для невеликих функцій. Другий спосіб опису методу полягає в тому, що усередині класу записується прототип, а опис методу розміщується в довільному місці програми поза тілом класу. У наведеному прикладі таким чином описаний метод *summa().*Описания методов *getx()* и *gety()* размещены внутри класса.

2.2 Создание и использование объектов

После описания класса можно объявить один или несколько объектов как экземпляр этого класса. Для приведенного примера объявление объектов будет выглядеть следующим образом:

Sum k, z;

Здесь объявлены объекты *k* и *z* типа *Sum.*

Для доступа к компонентам объекта можно использовать два способа указания имени объекта:

1. Непосредственное указание имени объекта.
2. Косвенное задание имени объекта с помощью указателя и операции косвенного выбора (->).

При первом способе задания обращение к компонентам объекта имеет следующий формат:

имя_объёкта.имя_класса::обращение к компоненту

Например, запись *k.Sum*::*x* означает обращение к компонентному данному *x* объекта *k* типа (класса) *Sum.*

Обращение к компонентам объекта в ряде случаев можно выполнить без указания имени класса, к которому принадлежит объект, в следующем формате:

имя_объекта.обращение_к_компоненту

```
z.gety(y2);
```

имя_конструктора имя_объекта (список_ аргументов);

В обоих случаях осуществляется создание объекта указанного класса и инициализация его элементов данных. Возможность второго варианта объясняется тем, что имя конструктора совпадает с именем класса.

Например, конструктор *Sum()* можно вызвать двумя способами: *Sum A=Sum(1,2);* и *Sum A(1,2).* В обоих случаях создается объект *A*, элементы данных *x* и *y* которого получают начальные значения *1* и *2* соответственно. Для обсуждаемого примера вызов конструктора можно осуществить без указания первого аргумента, т.е. *SumA(2);*. В этом случае элементу данных *x* будет присвоено нулевое значение по умолчанию (x2=0). В классе могут быть несколько конструкторов.

Деструкторы уничтожают объекты класса и освобождают занимаемую этими объектами память. Деструктор представляет собой метод с именем, совпадающим с именем класса, перед которым стоит символ тильда (~). Деструктор не должен иметь ни параметров, ни типа возвращаемого значения. Описание деструктора имеет следующий формат:

~имя класса() (операторы_тела_деструктора)

Например**,** для класса *Sum* описание деструктора выглядит следующим образом:

~Sum() {}

Деструктор вызывается *явно* или *неявно.* Деструктор вызывается ***явно*** (как обычный вызов функции) при необходимости уничтожения объекта. Вызов деструктора выполняется ***неявно*** (автоматически) для локального объекта тогда, когда перестает быть активным блок, в котором локальный объект объявлен. Если значения указателей объектов выходят за пределы области действия объявления объекта, то неявный вызов деструктора не происходит, а для разрушения такого объекта необходимо явным образом выполнить операцию *delete.*

Пример 1. Использование конструктора и деструктора.

Пусть требуется составить программу, реализующую вычисления по следующей формуле: *s=axb+cxk+axc.*

```
#include <iostream.h>
struct Pro
  {
   private:
   int x,y,z;
   public:
   // Прототипы методов:
   Pro (int, int);         // конструктор
   int putx();             // доступ к x
   int puty();             // доступ к y
   int putz();             // доступ к z
   void proizv();          // произведение
   ~Pro();                 // деструктор
  );
```

```
// Описания методов:
Pro::Pro(int xl,int yl) { x=xl; y=yl;}
int Pro::putx() { return x;}
int Pro::puty() { return y;}
int Pro::putz() ( return z;}
void Pro::proizv() {z=x*y;}
Pro::~Pro() { )
void main()
  (
    int s,a,b,c,k;
    cout << "\n Введите a,b,c и k\n";
    cin >> a >> b >> c >> k;
    Pro D = Pro(a,b);     //создание и инициализация объекта D
    Pro E(c,k);           //создание и инициализация объекта E
    Pro F(a,c);           //создание и инициализация объекта F
    D.proizv();           //получение произведения a*Ь
    E.proizvf);           //получение произведения c*k
    F.proizv();           //получение произведения a*c
    cout << "\n D.a=" << D.putx();
    cout << "\t D.b=" << D.puty();
    cout << "\t D.z=" << D.putz();
    s=D.putz()+E.putz()+F.putz();
    cout << "\n s=" << s;
    F.Pro::~Pro();      //уничтожение объекта F
    E.Pro::~Pro();      //уничтожение объекта E
    D.Pro::~Pro();      //уничтожение объекта D
  }
```

В примере для получения произведения создан класс *Pro,* компонентными данными которого являются сомножители *x* и *y,* а также элемент z, предназначенный для хранения произведения. Для доступа к компонентным данным используются три метода *putx(), puty()* и *putz(),* а для получения произведения *proizv(),* Кроме того, для класса определены конструктор *Рго()* и деструктор *~Pro ()* — в классе содержатся их прототипы.

Описания всех методов, в том числе конструктора и деструктора, вынесены за пределы описания класса. В результате трех явных вызовов конструктора созданы и инициализированы объекты Д, *E и R.* В результате работы метода *proizv()* получены три значения произведений для своих объектов. Результатом работы программы является получение суммы этих значений. В конце программы три раза вызван деструктор для уничтожения созданных ранее объектов.

2.4. Пример создания и использования класса

В качестве комплексного примера рассмотрим класс *Point,* который позволяет сформулировать точку на экране компьютера. Поместим описание класса в файл с именем *point.h:*

```
// Файл point.h
#ifndef POINT_H
#define POINT_H 1
class Point              //класс для определения точки на экране
  {
   protected:           //защищенный статус доступа к элементам данных
   int x;              //координата x точки
   int y;              //координата y точки
   // Прототипы методов:
   public:              //общедоступный статус доступа
   Point (int, int);     //конструктор
   int putx();          //доступ к x
   int puty();          //доступ к y
   void show(); //изобразить точку на экране
   void move (int, int);  //переместить точку
   private:             //собственный статус доступа
   void hide();          //убрать изображение точки
  };
#endif
```

Поскольку описание класса *Point* планируется использовать при описании других классов, то для предотвращения недопустимого дублирования описания класса в текст включены три директивы препроцессора *#ifndef POINT_H, #define POINTJI 1* и *#endif.*

Компонентами класса *Point* являются два элемента данных *x* и *y* с защищенным статусом доступа, пять общедоступных методов и один метод с собственным статусом доступа. Методы в описании класса представлены своими прототипами.

Выполним внешнее описание методов класса, разместив описания в файле *point.cpp:*

```
//Файл point.cpp - описание методов
#ifndef POINT_CPP
#define POINT_CPP 1
#include <graphics.h>// прототипы функций графической библиотеки
#include "f:\POS_C\PRIMER\point.h"   // описание класса Point
// Конструктор:
Point::Point (int xl=0, int y1=0)
  (
   x=xl;
   y=yl;
  }
//Метод доступа к x:
int Point::putx()
  {
   return x;
```

```
  }
//Метод доступ к y:
int Point::puty()
  (
    return y;
  }
//Метод изображения точки на экране:
void Point::show(void)
  {
   putpixel(x,y,getcolor()) ;
  }
// Метод удаления точки с экрана:
void Point::hide(void)
  {
   putpixei(x,y,getbkcolor());
  } // Метод перемещения точки в новое место экрана:
void Point::move(int xn=0, int yn=0)
  {
   hide();
   x=xn;
   y=yn;
   show();
  }
#endif
```

Для получения изображения точки на экране в программу должен быть включен заголовочный файл *graphics.h. В* данном файле находятся прототипы графических функций.

Достоинство *внешнего описания методов класса* состоит в том, что оно позволяет при необходимости модифицировать содержание методов, причем эти изменения останутся незамеченными для остальных частей программы.

Программа, содержащая в своем составе главную функцию, будет выглядеть следующим образом:

```
// Точка на экране
#include <iostream.h>
#include <graphics.h>     //прототипы графических функций
#include <conio.h>        //прототип функции getch()
#include "f:\POS_C\PRIMER\point.cpp"' //описание класса Point
int main()
  {
   Point t(100,150);     //создана невидимая точка t(x=100,y=150)
   Point tl(200,200);    //создана невидимая точка tl(X=200,Y=200)
   // Инициализация графики
   int a=DETECT,b;
   initgraph(&a,&b,"F:\\BC5\\bgi")?
```

```
  t.show();            // показать точку t
  cout << "\n коор-ты точки t: x=" << t.putx() << ",y=" << t.puty();
  getch();             //ждать нажатия клавиши
  tl.show();           //показать точку tl
  cout << "\n координаты точки tl: x=" << tl.putx() << ",y=" << tl.puty();
  getch();
  t.move (150,300);      //переместить точку t (x=150,y=300)
  cout << "\n новые координаты t: x=" << t.putx() << ",y=" << t.puty();
  getch();
  closegraphO;         //закрыть графический режим
 }
```

В программе используется функция *getch(),* которая позволяет остановить выполнение программы до нажатия на клавиатуре любой клавиши. Прототип этой *функции* находится в файле *conio.h.*

Контрольные вопросы и задания

1. Какова цель инкапсуляции?
2. Дайте определение понятию «класс».
3. Укажите формат описания класса.
4. Что такое метод?
5. Назовите статусы доступа к компонентам класса.
6. Какую область имеют действия спецификаторов на компоненты класса?
.7. Укажите различие между компонентами класса, объявленными с помощью ключевых слов *strukt, union, class.*
8. Какие функции имеют доступ к собственным элементам данных класса?
9. Назовите варианты размещения описаний методов класса.
10. В чем заключается достоинство внешнего описания методов класса?
11. Каким образом можно обратиться к элементам данных объекта?
12. В каком случае обращение к компанентам объекта можновыполнить без указания имени класса?
13. В чем назначение конструктора?
14. В чем особенности вызова конструктора по сравнению с вызовом других методов?
15. Каково назначение деструктора?
16. Укажите формат описания деструктора.
17. Назовите варианты вызова деструктора.
18. Всегда ли вызывается конструктор при создании объекта?
19. Измените описанный выше класс *Point* таким образом, чтобы элементы данных *x* и *y* стали общедоступными.
20. Для класса *Point* создайте конструктор, у которого отсутствуют параметры по умолчанию.
21. Для измененного класса *Point,* у которого общедоступные данные, создайте объект этого класса и организуйте обращение к элементам данных *x* и *y,* не используя методы класса *Point.*

22. Составьте программу игры в морской бой. При этом предусмотреть 5 классов кораблей и класс поля боя для размещения кораблей одной из сторон.

23. Составьте программу, моделирующую поиск мышки кошкой в комнате с предметами. Педусмотреть классы подвижных и неподвижных объектов разных размеров.

3 СПАДКОЄМСТВО

Зазвичай класи не існують самі по собі. Вони виникають з введенням поняття *об'єкт,* як тип цього об'єкту. Кожен окремий об'єкт .взаємодіє з іншими частинами програми за допомогою повідомлень. У відповідь на передане йому повідомлення *об'єкт* виконує такі дії, які закладені в методах того класу, до якого належить об'єкт. Такими діями можуть бути або зміни внутрішнього стану об'єкту, тобто зміни компонентних даних об'єкту, або передача повідомлень інших частин програми.Обычно классы не существуют сами по себе. Они возникают с введением понятия *объект,* как тип этого объекта. Каждый отдельный объект .взаимодействует с другими частями программы с помощью сообщений. В ответ на переданное ему сообщение *объект* выполняет такие действия, которые заложены в методах того класса, к которому принадлежит объект. Такими действиями могут быть либо изменения внутреннего состояния объекта, т.е. изменения компонентных данных объекта, либо передача сообщений другим частям программы.

Об'єкти різних класів і самі класи можуть знаходитися у відношенні *наслідував ния,* відповідно до якого формується ієрархія об'єктів і ієрархія класів відповідно.Объекты разных классов и сами классы могут находиться в отношении *наследовал ния,* в соответствии с которым формируется иерархия объектов и иерархия классов соответственно.

Механізм спадкоємства дозволяє визначати нові класи на основі вже наявних. Клас, на основі якого створюється новий клас, називають ***базовим (батьківським)*** класом, а новий — ***похідним (спадкоємцем). Безпосереднім базовим класом*** називається такий клас, який входить в список базових класів при визначенні класу. Будь-який похідний клас може у свою чергу стати базовим для інших створюваних класів. Таким чином, формується спрямований граф ***ієрархії класів,*** а при оголошенні об'єктів і ***ієрархія об'єктів.*** У ієрархії об'єктів похідний об'єкт має можливість доступу до елементів даних і методів об'єктів, що типізуються базовим класом.Механизм наследования позволяет определять новые классы на основе уже имеющихся. Класс, на основе которого создается новый класс, называют ***базовым (родительским)*** классом, а новый — ***производным (наследником). Непосредственным базовым классом*** называется такой класс, который входит в список базовых классов при определении класса. Любой производный класс может в свою очередь стать базовым для других создаваемых классов. Таким образом, формируется направленный граф ***иерархии классов,*** а при объявлении объектов и ***иерархия объектов.*** В иерархии объектов

производный объект имеет возможность доступа к элементам данных и методам объектов, типизированных базовым классом.

У мові існує можливість поодинокого і множинного спадкоємства. При ***поодинокому спадкоємстві*** базовим є один клас, а при ***множинному спадкоємстві*** базовими класами має бути декілька класів.В языке существует возможность одиночного и множественного наследования. При ***одиночном наследовании*** базовым является один класс, а при ***множественном наследовании*** базовыми классами должны быть несколько классов.

3.1 Управління доступом похідних класів

При спадкоємстві важливу роль грає статус доступу до компонент класу. Нагадаємо, що в ієрархії класів використовуються наступні угоди про права доступу до компонент класів :

- *власні (private)* компоненти доступні тільки усередині того класу, де вони визначені;*собственные (private)* компоненты доступны только внутри того класса, где они определены;
- *захищені (protected)* методи і елементи даних доступні усередині класу, в якому вони визначені, і в усіх похідних класах;*защищенные (protected)* методы и элементы данных доступны внутри класса, в котором они определены, и во всех производных классах;
- *загальнодоступні (public)* компоненти класу видимі з будь-якої точки програми.*общедоступные (public)* компоненты класса видимы из любой точки программы.

Таким чином, для *об'єкту,* який *обмінюється сполученнями* з іншими об'єктами і обробляє їх, доступними є:Таким образом, для *объекта,* который *обменивается сообщениями* с другими объектами и обрабатывает их, доступными являются:

- загальнодоступні компоненти усіх об'єктів програми;
- захищені дані і методи об'єктів, що є представниками базових класів;
- власні компоненти об'єкту.

При описі похідного класу можна змінити статус доступу до успадковних компонент класу за допомогою ***модифікатора статусу доступу.*** Формат опису похідного класу виглядає таким чином:При описании производного класса можно изменить статус доступа к наследуемым компонентам класса с помощью ***модификатора статуса доступа.*** Формат описания производного класса выглядит следующим образом:

class | struct ім'я_похідного_класу : [модифікатор] ім'я_базового_класу {компоненти_класу};class | struct имя_производного_класса: [модификатор] имя_базового_класса {компоненты_класса};

Як модифікатор статусу доступу використовуються ключові слова ***private, protected, public,*** В таблиці. 10.1 наводяться значення статусу доступу до компонент похідного класу залежно від статусу доступу до компонент базового класу і значення модифікатора статусу доступу.В качестве модификатора статуса доступа используются ключевые слова ***private, protected, public,*** В табл. 10.1

приводятся значения статуса доступа к компонентам производного класса в зависимости от статуса доступа к компонентам базового класса и значения модификатора статуса доступа.

Таблиця 10.1 - Статуси доступу похідних класів

Статус доступу у базовому класі	Модифікатор доступу	Статус доступу в похідному класі	
		struct	class
Public	—	public	private
protected	—	public	private
private	-	недоступні	недоступні
public	public	public	public
protected	public	protected	protected
private	public	недоступні	недоступні
public	protected	protected	protected
protected	protected	protected	protected
private	protected	недоступшпл	недоступні
public	private	private	private
protected	private	private	private
private	private	недоступні	недоступні

З таблиці видно, що в похідних класах статус доступу до компонент класу може бути тільки посилений. Щоб уникнути помилок доцільно завжди явно вказувати статус доступу для кожного компонента класу незалежно від установок за умовчанням.

3.2. Поодиноке спадкоємство

Нагадаємо, що при поодинокому спадкоємстві базовим для похідного класу служить один клас. Формат опису похідного класу при поодинокому спадкоємстві приведений в попередньому підрозділі. Розглянемо приклад поодинокого спадкоємства.

Приклад 1. Поодиноке спадкоємство класів.**Пример 1.** Одиночное наследование классов.

Вимагається скласти програму, яка дозволяє отримати на екрані коло. Для ілюстрації механізму спадкоємства на основі класу *Point* побудуємо похідний клас *Circle* (коло). Для похідного класу з класу *Point* виберемо наступні елементи даних і методи :Требуется составить программу, которая позволяет получить на экране окружность. Для иллюстрации механизма наследования на основе класса *Point* построим производный класс *Circle* (окружность). Для производного класса из класса *Point* выберем следующие элементы данных и методы:

- *intx* — координата сточування;*intx* — координата сточки;
- *inty* — координатах/точки;*inty* — координатах/точки;

- *intputx*() — доступ до x;*intputx*() — доступ к x;
- *intputy()* — доступ до y.*intputy()* — доступ к y.

Додатково для класу *Circle* введемо наступні компоненти:Дополнительно для класса *Circle* введем следующие компоненты:

- *int radius* — радіус кола;*int radius* — радиус окружности;
- *int vis* — індикатор видимості кола на екрані;*int vis* — индикатор видимости окружности на экране;
- *voidshpw()* — зображувати коло на екрані;*voidshpw()* — изобразить окружность на экране;
- *void hide()* — прибрати зображення кола;*void hide()* — убрать изображение окружности;
- *voidmove()* — перемістити коло на нове місце екрану;*voidmove()* — переместить окружность на новое место экрана;
- *void vary ()* — змінити розмір кола;*void vary ()* — изменить размер окружности;
- *int putradius()* — забезпечити доступ до радіусу кола.*int putradius()* — обеспечить доступ к радиусу окружности.

Відповідно до викладеного створимо клас *Circle* і його опис помістимо у файл *circle.h:*В соответствии с изложенным создадим класс *Circle* и его описание поместим в файл *circle.h:*

// Файл circle.h - опис похідного класуФайл circle.h - описание производного класса

#include "f: POS_C.cpp" // опис класу Pointinclude "f:POS_C\PRIMER\point.cpp" // описание класса Point

class Circle: public Point //клас для визначення колаclass Circle: public Point //класс для определения окружности

{ // модифікатор public дозволяє зробити елементи x і y класумодификатор public позволяет сделать элементы x и y класса

// Point в описі класу Circle захищеними (protected)Point в описании класса Circle защищенными (protected)

protected: //захищений статус доступу до елементів данихprotected: //защищенный статус доступа к элементам данных

int radius; //радіус колаint radius; //радиус окружности

int vis; //видимість кола на екраніint vis; //видимость окружности на экране

//Прототипи методів :

public: //загальнодоступний статус доступуpublic: //общедоступный статус доступа

Circle(int/int, int); //конструкторCircle(int/int,int); //конструктор

~Circle(); //деструкціяCircle(); //деструктор

void show(); //зображувати коло на екраніvoid show(); //изобразить окружность на экране

void hide(); //прибрати зображення колаvoid hide(); //убрать изображение окружности

void move (int/int); //перемістити коловoid move (int/int); //переместить окружность

void vary(int); //змінити розмір колаvoid vary(int); //изменить размер окружности

int putradius(); // доступ до радіусуint putradius(); // доступ к радиусу

};

У створеному класі *Circle* явно визначені конструктор *CircleQ* і деструкція *~Circle().* З класу *Point* наслідують два методи *putx()* і *puty(),* оскільки методи *show()* і *move()* замінюються однойменними методами класу *Circle,* а метод *hide()* не наслідує, оскільки має статус доступу *private.*В созданном классе *Circle* явно определены конструктор *CircleQ* и деструктор *~Circle().* Из класса *Point* наследуются два метода *putx()* и *puty(),* так как методы *show()* и *move()* заменяются одноименными методами класса *Circle,* а метод *hide()* не наследуется, поскольку имеет статус доступа *private.*

Використовуємо зовнішні описи методів, помістивши їх в окремий файл з ім'ям *circle.cpp:*Используем внешние описания методов, поместив их в отдельный файл с именем *circle.cpp:*

//Файл circle.cppФайл circle.cpp

#include <graphics.h> // прототипи функцій графічної бібліотекиinclude <graphics.h> // прототипы функций графической библиотеки

#include "f:_C.h" // опис класу Circleinclude "f:\POS_C\PRIMER\circle.h" // описание класса Circle

//Описи методів класу Circle :Описания методов класса Circle:

//Конструктор

Circle::Circle (int xc, int yc, int radc=0) : Point(xc, yc)

(

radius=radc;

vis=l;

}

//Зображувати коло на екрані

void Circle::show(void)void Circle::show(void)

{

circle(x, y, radius); //малювання колаcircle(x,y,radius); //рисование окружности

)

//Прибрати коло з екрану

void Circle::hide(void)void Circle::hide(void)

(

unsigned infc col; //оголошення змінної для поточного кольоруunsigned infc col; //объявление переменной для текущего цвета

if (vis^O) return; //кола немаєif (vis^O) return; //окружности нет

col=getcolor(); //запам'ятовування поточного кольоруcol=getcolor(); //запоминание текущего цвета

```
setcolor(getbkcolor());                 //виявлення поточного кольору фонуsetcolor(getbkcolor());     //выявление текущего цвета фона
circle(x, y, radius);     //малювання кола кольором фонуcircle(x,y,radius); //рисование окружности цветом фона
vis=0;vis=0;
setcolor(col);                 //відновлення поточного кольоруsetcolor(col); //восстановление текущего цвета
}
// Перемістити коло в нове місце екрану
void Circle::move(int xn, int yn)
{
  hide();              //стирання старого колаhide();              //стирание старой окружности
  x=xn;x=xn;
  y=yn;y=yn;
  show();              //малювання нового колаshow();              //рисование новой окружности
}
// Змінити розмір кола
void Circle::vary(int d)void Circle::vary(int d)
(
  hide();              //стирання старого колаhide();              //стирание старой окружности
  radius+=d;           //зміна радіусуradius+=d;           //изменение радиуса
  if(radius<0)
    radius=0;
  show();              //малювання нового колаshow();              //рисование новой окружности
}
// Доступ до радіусу
int Circle::putradius()int Circle::putradius()
{
  return radius;
}
//Деструкція
Circle::~Circle()
{
   hide();             //стирання колаhide();            //стирание окружности
}
```

Конструктор *Circle()* має три параметри: координати центру *(хс, вус)* і радіус кола *radc.* При створенні об'єкту класу *Circle* спочатку викликається конструктор класу *Point,* який по значеннях фактичних параметрів визначає центр кола. Ця точка створюється як об'єкт класу *Point* без імені. Потім виконується тіло конструктора *Circle().*Конструктор *Circle()* имеет три

параметра: координаты центра *(xc, yc)* и радиус окружности *radc.* При создании объекта класса *Circle* вначале вызывается конструктор класса *Point,* который по значениям фактических параметров определяет центр окружности. Эта точка создается как объект класса *Point* без имени. Затем выполняется тело конструктора *Circle().*

Деструкція - *Circle()* знищує коло і викликає деструкцію базового класу - *Point(),* який, незважаючи на відсутність його опису в класі, формується компілятором за умовчанням.Деструктор *-Circle()* уничтожает окружность и вызывает деструктор базового класса *-Point(),* который, несмотря на отсутствие его описания в классе, формируется компилятором по умолчанию.

Тепер можна скласти програму для роботи з об'єктами класу *Circle*Теперь можно составить программу для работы с объектами класса *Circle,*

#include <iostream.h>include <iostream.h>

tinclude <graphics.h> //прототипи графічних функційtinclude <graphics.h> //прототипы графических функций

ftinclude <conio.h> //прототип функції getch()ftinclude <conio.h> //прототип функции getch()

#include "f:_C.cpp" //опис класу Circleinclude "f:\POS_C\PRIMER\circle.cpp" //описание класса Circle

int main()

{

// Ініціалізація графікиИнициализация графики

int a=DETECT, b;

initgraph(&a,&b, "F:\\BC5\\bgi");

Circle A(150,250,30); //створено невидиме коло ACircle A(150,250,30); //создана невидимая окружность A

Circle B(300,100,50); //створено невидиме коло BCircle B(300,100,50); //создана невидимая окружность B

Cout<<"A : x=" <<A.putx()<<", y=" << A.puty() <<", радіус" Cout<<"\n A: x=" <<A.putx()<<", y=" << A.puty() <<", радиус"

<< A.putradius();

cout<< "B: x=" << B.putx() <<", y=" << B.puty() <<cout<< "\n B: x=" << B.putx() <<", y=" << B.puty() <<

", радиус=" << B.putradius();радиус=" << B.putradius();

getchf); //чекати натиснення клавішіgetchf); //ждать нажатия клавиши

A.show(); //показати на екрані коло AA.show(); //показать на экране окружность A

getch(); //жцать натиснення клавішіgetch(); //жцать нажатия клавиши

B. show(); //показати на екрані колоB.show(); //показать на экране окружность

У getch(); //чекати натиснення клавішіB getch(); //ждать нажатия клавиши

```
        Point  З(300,100);                    //створена  точка  ЗPoint  C(300,100);
//создана точка С
        C. show();              // показати на екрані точкуC.show();              //
показать на экране точку С
        getch();              // чекати натиснення клавішіgetch();              //
ждать нажатия клавиши
        A.move(150,150);          // перемістити коло АA.move(150,150);          //
переместить окружность А
        Cout << "\n A : x=" << A.putx() << ", y=" << A.puty() <<
          ", радиус=" << A.putradius();радиус=" << A.putradius();
        getch();getch();
        B. vary(20);              // змінити розміри колаB.vary(20);              //
изменить размеры окружности В
        Cout << "\n B : x=" << B.putx() << ", y=" << B.puty() <<
          ", радиус=" << B.putradius();
        getch();
        A.Circle::~Circle();          // знищення об'єкту АA.Circle::~Circle();          //
уничтожение объекта А
        B. Circle::~Circle ();        // знищення об'єкту ВB.Circle::~Circle ();        //
уничтожение объекта В
        Closegraph();              // закрити графічний режимClosegraph();              //
закрыть графический режим
    )
```

В результаті виконання програми спочатку конструктором *Circle()* створюються два невидимі об'єкти: коло *А з* координатами (150,250) і радіусом 30 і коло *В з* координатами (300,100) і радіусом 50. В результаті двократної роботи методу *show()* об'єкти *А* і *В* стають видимими. При явному виклику конструктора *Point()* створюється об'єкт *Із* з координатами центру кола *В. В результаті* роботи методу *show()* класу Point точка С стає видимою. Метод *move()* стирає старе коло А, змінює в об'єкті В значення захищених координат об'єкту (150,150) (функції, які не належать до класів *Point і Circle,* цього зробити не можуть) і малює коло колишнього радіусу на новому місці екрану. Метод *vary()* стирає коло об'єкту *В* з екрану (при цьому об'єкт *В* не знищується) і після зміни значення радіусу малює коло з іншим радіусом.В результате выполнения программы вначале конструктором *Circle()* создаются два невидимых объекта: окружность *А с* координатами (150,250) и радиусом 30 и окружность *В с* координатами (300,100) и радиусом 50. В результате двукратной работы метода *show()* объекты *А* и *В* становятся видимыми. При явном вызове конструктора *Point()* создается объект *С* с координатами центра окружности *В. В* результате работы метода *show()* класса Point точка С становится видимой. Метод *move()* стирает старую окружность А, изменяет в объекте В значения защищенных координат объекта (150,150) (функции, которые не принадлежат к классам *Point и Circle,* этого сделать не могут) и рисует окружность прежнего радиуса на новом месте экрана. Метод *vary()* стирает окружность объекта *В* с экрана (при этом

объект *B* не уничтожается) и после изменения значения радиуса рисует окружность с другим радиусом.

3.3 Множинне спадкоємство

Множинне спадкоємство є таким спадкоємством, при якому створення похідного класу грунтується на використанні декількох безпосередніх базових класів. При множинному наследованиие використовується наступний формат опису класу :*Множественное наследование* представляет собой такое наследование, при котором создание производного класса основывается на использовании нескольких непосредственных базовых классов. При множественном наследованиие используется следующий формат описания класса:

class I struct иьдя__похідного_класу : [модифікатор] ім'я_базового_класу_1,., [модифікатор] ім'я_базового_класу_п { компоненти_класу };class I struct иьдя__производного_класса: [модификатор] имя_базового_класса_1 ,.., [модификатор] имя_базового_класса_п { компоненты_класса };

Як приклад множинного спадкоємства розглянемо похідний клас *Reccircle* — «прямокутник і коло». Для опису класу *Reccircle* створимо новий безпосередній базовий клас *Rectangle* — «прямокутник» і використовуємо раніше створений клас *Circle* — «коло». Опис класу *Rectangle* помістимо в окремий файл *rectang.h:*В качестве примера множественного наследования рассмотрим производный класс *Reccircle* — «прямоугольник и окружность». Для описания класса *Reccircle* создадим новый непосредственный базовый класс *Rectangle* — «прямоугольник» и используем ранее созданный класс *Circle* — «окружность». Описание класса *Rectangle* поместим в отдельный файл *rectang.h:*

//Файл rectang.h - опис класу прямокутникФайл rectang.h - описание класса прямоугольник

#include <graphics.h> //прототипи функцій графічної бібліотекиinclude <graphics.h> //прототипы функций графической библиотеки

#include "f:_C.cpp" //опис класу Pointinclude "f:\TOS_C\PRIMER\point.cpp" //описание класса Point

class Rectangle: public Point //клас для визначення прямокутникаclass Rectangle: public Point //класс для определения прямоугольника

{

protected //захищений статус доступу до елементів данихprotected //защищенный статус доступа к элементам данных

int lx; //довжина по горизонталіint lx; //длина по горизонтали

int ly; - //довжина по вертикаліint ly; • //длина по вертикали

// Прототипи методів :

void risrec(); //промальовування прямокутникаvoid risrec(); //прорисовка прямоугольника

public: //загальнодоступний статус доступуpublic: //общедоступный статус доступа

Rectangle(int, int, int, int) // конструкторRectangle(int,int,int,int) // конструктор

void show (void); //промальовування прямокутника.void show (void); //прорисовка прямоугольника.

void hide (void); //прибрати зображення прямокутника з екрануvoid hide (void); //убрать изображение прямоугольника с экрана

);

Безпосереднім базовим класом для класу *Rectangle* є клас *Point,* Окрім успадковних елементів даних *x* і *y* (координати центру прямокутника) клас *Rectangle* у своєму складі має два елементи даних *lx* — довжина по горизонталі і /у — довжина по вертикалі. Чотири методи класу дозволяють створювати об'єкти, їх ініціалізувати, отримувати зображення на екрані і прибирати його у разі потреби. Описи методів класу *Rectangle* розмістимо в окремому файлі *rectang.cpp:*Непосредственным базовым классом для класса *Rectangle* является класс *Point,* Кроме наследуемых элементов данных *x* и *y* (координаты центра прямоугольника) класс *Rectangle* в своем составе имеет два элемента данных *lx* — длина по горизонтали и /у — длина по вертикали. Четыре метода класса позволяют создавать объекты, их инициализировать, получать изображение на экране и убирать его в случае необходимости. Описания методов класса *Rectangle* разместим в отдельном файле *rectang.cpp:*

//Файл rectang.cpp - описи методів класу прямокутникФайл rectang.cpp - описания методов класса прямоугольник

#include <graphics.h> // прототипи функцій графічної бібліотекиinclude <graphics.h> // прототипы функций графической библиотеки

#include "f:_C.h" // опис класу Pointinclude "f:\POS_C\PRIMER\rectang.h" // описание класса Point

// Описи методів :

void Rectangle::risrecf) // промальовування прямокутникаvoid Rectangle::risrecf) // прорисовка прямоугольника

```
(
  int g=lx/2;
  int v=ly/2;
  line(x - g, y - v, x+g, y - v);
  line(x - g, y+v, x+g, y+v);
  line(x - g, y - v, x - g, y+v);
  line(x+g, y - v, x+g, y+v);
}
```

// Конструктор:Конструктор:

```
Rectangle :: Rectangle (int xi, int yi, int lxi=0, int lyi=0);
Point(xi.yi)
  {
  lx=lxi;
```

```
  ly=lyi;ly=lyi;
 }
// Зображувати прямокутник на екрані:
void Rectangle::show(void)void Rectangle::show(void)
 {
  risrecO; // промальовування прямокутникаrisrecO; // прорисовка прямоугольника
 }
// Прибрати зображення прямокутника з екрану:
void Rectangle::hide(void)
 {
  int b, c;
  b=getbkcolor(); // запам'ятовування поточного кольору фонуb=getbkcolor(); // запоминание текущего цвета фона
  c=getcolor(); // запам'ятовування кольору зображенняс=getcolor(); // запоминание цвета изображения
  setcolor (b); // встановити колір зображенняsetcolor (b); // установить цвет изображения
  risrec(); // намалювати прямокутник кольором фонуrisrec(); // нарисовать прямоугольник цветом фона
  setcolor (c); // відновлення кольору зображення }setcolor (c); // восстановление цвета изображения }
```

Тепер можна створити клас «прямокутник і коло» на підставі двох безпосередніх базових класів. Опис класу *Recdrcle* помістимо в окремий файл *reccirc.h:*Теперь можно создать класс «прямоугольник и окружность» на основании двух непосредственных базовых классов. Описание класса *Recdrcle* поместим в отдельный файл *reccirc.h:*

```
//Файл reccirc.cpp - опис класу "прямокутник і коло"Файл reccirc.cpp - описание класса "прямоугольник и окружность"
#include "f:_C.cpp" //опис класу Rectangleinclude "f:\POS_C\PRIMER\rectang.cpp" //описание класса Rectangle
#include "f:_C.cpp" // опис класу Circleinclude "f:\POS_C\PRlMER\circle.cpp" // описание класса Circle
// Клас "прямокутник і коло" :Класс "прямоугольник и окружность":
class Reccircle: public Rectangle, public Circle
 {
  public:
// Конструктор:Конструктор:
  Reccircle(int xi, int yi, int ri, int lxi, int lyi);
  Rectangle(xi,yi,Ixi,lyi), // Явный вызов конструкторов
  Circle(xi,yi,ri) // базовых классов
 { }
// Изобразить прямоугольник и окружность на экране:
void show(void)
```

```
  {
   Rectangle :: show();     // изобразить прямоугольник
   Circle    :: show();     // изобразить окружность
  }
   // Убрать изображение с экрана:
void hide(void)
  {
   Rectangle::hide();      // убрать изображение прямоугольника
   Circle::hide();         // убрать изображение окружности
  }
}
```

У класі описаний три методи: конструктор, який явним чином викликає конструктори безпосередніх базових класів, і методи зображення на екрані і стирання з екрану прямокутника і кола.

Тепер можна скласти програму, яка дозволить працювати з об'єктами класу *Reccircle.*Теперь можно составить программу, которая позволит работать с объектами класса *Reccircle.*

```
// Приклад множинного спадкоємства
#include <iostream.h>        // прототипи функцій введення-виводуinclude <iostream.h>   // прототипы функций ввода-вывода
#include <graphics.h>    // прототипи графічних функційinclude <graphics.h> // прототипы графических функций
#include <conio.h>         // прототип функції getch()include <conio.h>         // прототип функции getch()
#include "f:_C.cpp"                  // опис класу Reccircleinclude "f:\POS_C\PRIMER\reccirc.cpp"   // описание класса Reccircle
void main()
  {
// Ініціалізація графікиИнициализация графики
   int a=DETECT, b;
   initgraph(&a,&b, "F:\\BC5\\bgi");
   Reccircle A(250,100,100,50,50);  // створений невидимий об'єкт AReccircle A(250,100,100,50,50);  // создан невидимый объект A
   Reccircle B(400,300,50,120,140); // створений невидимий об'єкт BReccircle B(400,300,50,120,140); // создан невидимый объект B
   Cout<<"\u1087?              A:              x="<<A.Rectangle::putx()<<", "<<A.Rectangle::puty();Cout<<"\п              A:              x="<<A.Rectangle::putx()<<", "<<A.Rectangle::puty();
   Cout<<"B:  x="<<B.Circle::putx()<<",  y="<<B.Circle::puty();Cout<<"\n  B: x="<<B.Circle::putx()<<", y="<<B.Circle::puty();
   getch();              // чекати натиснення клавішіgetch();              // ждать нажатия клавиши
   A.show();            // показати на екрані об'єкт AA.show();            // показать на экране объект A
```

getch(); // чекати натиснення клавішіgetch(); // ждать нажатия клавиши
B. show(); // показати на екрані об'єктB.show(); // показать на экране объект B
getch(); // чекати натиснення клавішіgetch(); // ждать нажатия клавиши
A.hide(); // стерти об'єкт AA.hide(); // стереть объект A
getch(); // чекати натиснення клавішіgetch(); // ждать нажатия клавиши
A.Circle::show(); // показати на екрані коло об'єкту AA.Circle::show(); // показать на экране окружность объекта A
getch(); // чекати натиснення клавішіgetch(); // ждать нажатия клавиши
B. hide(); // стерти об'єктB.hide(); // стереть объект B
getch(); // чекати натиснення клавішіgetch(); // ждать нажатия клавиши
B. Rectangle :: show(); // показати на екрані прямокутник об'єктуB.Rectangle :: show(); // показать на экране прямоугольник объекта B
getch(); // чекати натиснення клавішіgetch(); // ждать нажатия клавиши
closegraph(); // закрити графічний режимclosegraph(); // закрыть графический режим
}

У програмі за допомогою виклику конструктора формуються два об'єкти *A і B з* координатами центрів (250,100) і (400,300) відповідно. В результаті роботи методів *A^how()* і *B.show()* на екрані висвічуються об'єкти А і В, кожен з яких складається з прямокутника і кола. Розміри прямокутника для об'єкту А: 50 — по горизонталі, 50 — по вертикалі, а для *B:* 120 — по горизонталі, 140 — по вертикалі. Радіус кола об'єкту *A* — 100, а *B* — 50. Після стирання з екрану об'єкту *A за допомогою* оператора *A.Circle::show();* на екрані висвічується коло об'єкту Л. Подібним же чином висвічується на екран прямокутник об'єкту *B*.В программе посредством вызова конструктора формируются два объекта *A и B с* координатами центров (250,100) и (400,300) соответственно. В результате работы методов *A^how()* и *B.show()* на экране высвечиваются объекты А и В, каждый из которых состоит из прямоугольника и окружности. Размеры прямоугольника для объекта А: 50 — по горизонтали, 50 — по вертикали, а для *B:* 120 — по горизонтали, 140 — по вертикали. Радиус окружности объекта *A* — 100, а *B* — 50. После стирания с экрана объекта *A с* помощью оператора *A.Circle::show();* на экране высвечивается окружность объекта Л. Подобным же образом высвечивается на экран прямоугольник объекта *B*.

Контрольні питання і завдання

1. Яким чином можна змінити статус доступу до успадковних компонент класу?

2. Назвіть і охарактеризуйте ключові слова, що використовуються як модификаторов статус доступу.

3. Назвіть угоди про права доступу до компонент класу.

4. У базовому класі встановлений статус доступу *public,* модифікатор доступу має значення *private.* Назвіть який статус доступу в похідному класі?В базовом классе установлен статус доступа *public,* модификатор доступа имеет значение *private.* Назовите каков статус доступа в производном классе?

5. У базовому класі встановлений статус *wcrymptiblic.* Яке треба вибрати значення модифікатора доступу, щоб в похідному класі отримати статус *dостуnу protected?*В базовом классе установлен статус *wcrymptiblic.* Какое нужно выбрать значение модификатора доступа, чтобы в производном классе получить статус *dостуnа protected?*

6. Який клас називається базовим?

7. У чому відмінність між базовим і безпосереднім базовим класами?

8. Які особливості поодинокого і множинного спадкоємства?

9. Приведіть формат опису класу, використовуваний при множинному спадкоємстві.

10. Створіть клас, що описує квадрат, використовуючи як базового клас *Point.*Создайте класс, описывающий квадрат, используя в качестве базового класс *Point.*

11. На основі створених класів створіть клас «коло і текст», який дозволить зображувати коло і писати текст усередині кола або за її межами.

12. Використовуючи існуючі класи, створіть клас «прямокутник і текст», який дозволить зображувати прямокутник і писати текст усередині прямокутника і за його межами.

4 ПОЛІМОРФІЗМ

Дослівно *поліморфізм* в перекладі з грецького означає різноманіття форм *(poly* — багато, *morphos* — форм). ***Поліморфізм*** можна визначити як властивість, що дозволяє використати одне ім'я для позначення дій, загальних для споріднених класів. При цьому конкретизація вьшолняемых дій здійснюється залежно від типу обрабатыва-емьк даних. До найважливіших форм поліморфізму в С++ можна віднести наступне:Дословно *полиморфизм* в переводе с греческого означает многообразие форм *(poly* — много, *morphos* — форм). ***Полиморфизм*** можно определить как свойство, позволяющее использовать одно имя для обозначения действий, общих для родственных классов. При этом конкретизация вьшолняемых действий осуществляется в зависимости от типа обрабатыва-емьк данных. К важнейшим формам полиморфизма в С++ можно отнести следующее:

- перевантаження функцій і операцій;
- віртуальні функції;
- узагальнені функції, або шаблони.

Перевантаження функцій і операцій можна визначити як *статичний (static)* поліморфізм, оскільки він підтримується на етапі компіляції. Віртуальні функції

відносяться до *динамічного (run time)* поліморфізму, оскільки він реалізується при виконанні програм.Перегрузку функций и операций можно определить как *статический (static)* полиморфизм, поскольку он поддерживается на этапе компиляции. Виртуальные функции относятся к *динамическому (run time)* полиморфизму, поскольку он реализуется при выполнении программ.

Гідністю поліморфізму є те, що дозволяє використати багаторазово один раз складені алгоритми і, як наслідок, забезпечує зменшення надмірного коду.

4.1 Перевантаження функцій

У С++ допускається наявність декількох *однойменних* функцій, що виконують аналогічні дії над даними *різних типів.* Наприклад, нехай в програмі визначені дві функції з прототипами:В С++ допускается наличие нескольких *одноименных* функций, выполняющих аналогичные действия над данными *разных типов.* Например, пусть в программе определены две функции с прототипами:

int max(int, int);int max(int,int);

float max(float, float);float max(float,float);

В цьому випадку в програмі допустима вказівка наступних операторів :

float x, y;float x,y;

cout<<"max="<<max(5,8)<<endl;

x=12.5; y=24.8;

cout<<"max="<<max(x, y);cout<<"max="<<max(x, y);

В процесі компіляції програми при зверненні до функцій *max()* залежно від типу і числа аргументів здійснюватиметься завантаження необхідного екземпляра функції. Описаний механізм називається перевантаженням функцій.В процессе компиляции программы при обращении к функциям *max()* в зависимости от типа и числа аргументов будет осуществляться загрузка требуемого экземпляра функции. Описанный механизм называется перегрузкой функций.

Якщо в програмі описані декілька однойменних функцій, то при компіляції можливі наступні ситуації:

1. Якщо *повертані значення і сигнатури* (тип і число параметрів) декількох функцій *співпадають,* то друге і подальші оголошення трактуються як переобъявления першого. Наприклад:Если *возвращаемые значения и сигнатуры* (тип и число параметров) нескольких функций *совпадают,* то второе и последующие объявления трактуются как переобъявления первого. Например:

extern double max(double a, double b);

double max(double c, double d);

2. Якщо *сигнатури* декількох однойменних функцій *співпадають,* але *повертані значення різні,* то друге і подальші оголошення при компіляції розглядаються як помилкові. Наприклад, за наявності двох однойменних функцій з наступними прототипамиЕсли *сигнатуры* нескольких одноименных функций *совпадают,* но *возвращаемые значения различны,* то второе и последующие объявления при компиляции рассматриваются как ошибочные. Например, при наличии двух одноименных функций со следующими прототипами

unsigned int max (unsigned, unsigned);unsigned int max (unsigned, unsigned);
extern long max(unsigned, unsigned);

при компіляції буде видано повідомлення про помилку. Причина його в тому, що при визначенні можливості перезавантаження функції тип повертаного значення не береться до уваги, а сигнатури в даному випадку співпадають.

3. Якщо *сигнатури* декількох однойменних функцій *різні* (параметри мають відмінності по типах або кількості), тоді функція вважається *перевантажуваною* за умови оголошення її екземплярів в одній зоні видимості. Наприклад:Если *сигнатуры* нескольких одноименных функций *различны* (параметры имеют различия по типам или количеству), тогда функция считается *перегружаемой* при условии объявления ее экземпляров в одной области видимости. Например:

```
#include <iostream.h>
int max(int, int);
float max(float, float);
void main(){
}
float max(float c, float d)
{
  if (Od) return (c);
  return(d);
}
int max(int a, int b)
(
  if (a>b) return(a);
  if (a<=b) return(b);
}
```

В даному випадку ми маємо два екземпляри перевантажуваної функції *max()*.В данном случае мы имеем два экземпляра перегружаемой функции *max()*.

4.2 Вибір екземпляра функції

При *виборі необхідного екземпляра функції* здійснюється порівняння типів і числа параметрів і аргументів. При цьому можливі три наступні варіанти:При *выборе требуемого экземпляра функции* осуществляется сравнение типов и числа параметров и аргументов. При этом возможны три следующие варианта:

1. Має місце *точна відповідність типів* аргументів параметрам одного з екземплярів функції. В цьому випадку здійснюється виклик цього екземпляра функції.Имеет место *точное соответствие типов* аргументов параметрам одного из экземпляров функции. В этом случае осуществляется вызов этого экземпляра функции.

2. *Відповідність* аргументів параметрам *може бути досягнута* шляхом перетворення типів аргументів до типів параметрів *тільки для одного екземпляра функції.* В цьому випадку компілятор намагається спочатку виконати відповідні стандартні перетворення типів даних, а потім перетворення, визначені

користувачем.*Соответствие* аргументов параметрам *может быть достигнуто* путем преобразования типов аргументов к типам параметров *только для одного экземпляра функции.* В этом случае компилятор пытается сначала выполнить подходящие стандартные преобразования типов данных, а затем преобразования, определенные пользователем.

3. *Відповідність* може бути досягнута *більш ніж для одного екземпляра* функції. У цій ситуації слід мати на увазі, що перетворення типів, визначені користувачем, мають менший пріоритет порівняно із стандартними перетвореннями. Якщо відповідність типів досягається шляхом равноприоритетных перетворень, то компілятором видається повідомлення про помилку. Це пов'язано з тим, тут неможливо визначити однозначно необхідний екземпляр функції.*Соответствие* может быть достигнуто *более чем для одного экземпляра* функции. В этой ситуации следует иметь в виду, что преобразования типов, определенные пользователем, имеют меньший приоритет в сравнении со стандартными преобразованиями. Если соответствие типов достигается путем равноприоритетных преобразований, то компилятором выдается сообщение об ошибке. Это связано с тем, здесь невозможно определить однозначно требуемый экземпляр функции.

Наприклад, нехай є наступні прототипи перевантажуваних экземшбфов функцій :

void fl(char);

void fl(unsigned int);

void fl(char*);

В цьому випадку *точні відповідності* матимуть місце при викликахВ этом случае *точные соответствия* будут иметь место при вызовах

fl('b'); //точна відповідність з fl(char)fl('b'); //точное соответствие с fl(char)

f1("D"); //точна відповідність з fl(char*)f1("D"); //точное соответствие с fl(char*)

fl(52u); //точна відповідність з fl(unsigned)fl(52u); //точное соответствие с fl(unsigned)

Точна відповідність має місце у разі, коли аргументи точно відповідають параметрам одного з оголошених екземплярів перевантажуваної функції. Зауваження.Точное соответствие имеет место в случае, когда аргументы точно соответствуют параметрам одного из объявленных экземпляров перегружаемой функции. Замечания.

- Аргументи типу *char* і *short* і константа *0* вважаються як що точно відповідають параметру типу *int*Аргументы типа *char* и *short* и константа *0* считаются как точно соответствующие параметру типа *int*
- Аргумент типу *floaf* вважається таким, що точно відповідає параметру типу *double.*Аргумент типа *floaf* считается точно соответствующим параметру типа *double.*

Якщо точна відповідність не має місця, то компілятор намагається виконати ***відповідні стандартні перетворення.*** Наприклад, нехай є прототипиЕсли точное соответствие не имеет места, то компилятор пытается выполнить

подходящие стандартные преобразования. Например, пусть имеются прототипы

```
void f2(long);
void f2(char*);
```

Тоді в т . д.д. *д.2(10);* буде соответствовт. д.д.д. д.му екземпляру функції, оскільки аргумент зі значенням 10 типу int перетвориться до типу long. *Зауваження.*Тогда в т . д.д. *д.2(10);* будет соответствовт. д.д.д.. д.му экземпляру функции, так как аргумент со значением 10 типа *int* преобразуется к типу *long.* Замечания.

- Аргумент будь-якого числового типу можна перетворити стандартним шляхом до будь-якого числового типу.
- Константа 0 може бути перетворена стандартним шляхом до покажчика.
- Будь-який покажчик може бути перетворений до покажчика на *void.*Любой указатель может быть преобразован к указателю на *void.*

Перетворення, ***визначені користувачем,*** застосовуються компілятором за відсутності точної відповідності і відповідних стандартних перетворень. Наприклад, нехай є описи:Преобразования, ***определенные пользователем,*** применяются компилятором при отсутствии точного соответствия и подходящих стандартных преобразований. Например, пусть имеются описания:

```
class stringclass string
  {
   char *str; int zf;char *str; int zf;
   public:
   operator int();          //Приведення типу string в тип intoperator int(); //Приведение типа string в тип int
  );
string p;
void f3(char*);
void f3(int);void f3(int);
```

Тоді виклик *f3(p);* відповідатиме екземпляру функції *f3(int),* тому що є перетворення з *string* в *int,* визначене користувачем в класі *string :* метод *operator int()* визначає операцію приведення з типу *string* в тип *int.*Тогда вызов *f3(p);* будет соответствовать экземпляру функции *f3(int),* потому что есть преобразование из *string* в *int,* определенное пользователем в классе *string:* метод *operator int()* определяет операцию приведения из типа *string* в тип *int.*

Якщо немає точної відповідності аргументів параметрам і немає відповідних перетворень (стандартних або користувача), то видається повідомлення про помилку.

Зауваження.

Якщо у компілятора немає можливості віддання переваги між двома перетвореннями — різні стандартні перетворення призводять до різних екземплярів перевантажуваних функцій або визначено більше за одне призначене для користувача перетворення, що призводить до однієї мети, — те фіксується помилка «двозначність». Наприклад:Если у компилятора нет возможности отдания предпочтения между двумя преобразованиями — различные стандартные

преобразования приводят к различным экземплярам перегружаемых функций или определено более одного пользовательского преобразования, приводящего к одной цели, — то фиксируется ошибка «двусмысленность». Например:

void f4(long);void f4(long) ;
void f4(double);void f4(double);
f4 ('a'); //Помилка - двозначністьf4 ('a'); //Ошибка - двусмысленность

В даному випадку точної відповідності для аргументу типу *char* немає. Для досягнення відповідності используются стандартні перетворення. Тип *char* можна перетворити до типу *long* і до типу *double.* Зважаючи на равноприоритетности цих перетворень виникає помилка «двозначність».В данном случае точного соответствия для аргумента типа *char* нет. Для достижения соответствия используются стандартные преобразования. Тип *char* можно преобразовать к типу *long* и к типу *double.* Ввиду равноприоритетности этих преобразований возникает ошибка «двусмысленность».

4.3 Перевантаження стандартних операцій

Перевантаження стандартних операцій можна розглядати як різновид перевантаження функцій, при якому залежно від типів даних, таких, що беруть участь у виразах, викликається необхідний екземпляр операції. Наприклад, стандартна операція «+» призначена для складання звичайних числових даних (з фіксований^ ний або плаваючою точкою). С++ дозволяє розширити область застосовності цієї операції, наприклад, для складання даних комплексного типу або для конкатенації символьних рядків. Для цього треба перевизначити нову поведінку стандартної операції «+» стосовно нових типів даних. Це допускається, якщо хоч би один з операндів операції є об'єктом визначеного користувачем класу.***Перегрузку стандартных операций*** можно рассматривать как разновидность перегрузки функций, при которой в зависимости от типов данных, участвующих в выражениях, вызывается требуемый экземпляр операции. Например, стандартная операция «+» предназначена для сложения обычных числовых данных (с фиксирован^ ной или плавающей точкой). С++ позволяет расширить область применимости этой операции, например, для сложения данных комплексного типа или для конкатенации символьных строк. Для этого нужно переопределить новое поведение стандартной операции «+» применительно к новым типам данных. Это допускается, если хотя бы один из операндов операции является объектом определенного пользователем класса.

Перевизначення, і як наслідок, перевантаження може виконуватися для наступних стандартних операцій і вбудованих функціональних викликів :

+, -, *, /, **=, <, >,** +=, -=, *=, /=, <<, >>, <<=, >>=, ==, !, **<=, >=,**
++, --, %, &, ^, !, &=, ^=, |=, &&, ||, %=, [], (), new, delete.

Не можуть бути перевизначені наступні операції: .,.*,?:, ::, sizeof. Перевизначення операції виконується за допомогою визначення так званої ***операторної функції*** наступного формату :Не могут быть переопределены следующие операции: .,.*,?:, ::, sizeof. Переопределение операции выполняется с

помощью определения так называемой ***операторной функции*** следующего формата:

тип_результату operator знак_операції(список параметрів)тип_результата operator знак_операции(список параметров)

{оператори тіла операторної функції)

Тут *тип_результату* визначає тип повертаного значення при виконанні перевизначеної операції; у круглих дужках задається список типів параметрів, за наявності яких у аргументів звернення до операції з цим знаком робитиметься перевантаження (виклик) саме цього екземпляра операції.Здесь *тип_результата* определяет тип возвращаемого значения при выполнении переопределённой операции; в круглых скобках задаётся список типов параметров, при наличии которых у аргументов обращения к операции с данным знаком будет производиться перегрузка (вызов) именно этого экземпляра операции.

Операторна функція може бути компонентною функцією класу, в цьому випадку її зовнішнє (за рамками опису класу) визначення має наступний формат:

тип_результату клас :: operator знак_операції(список параметрів)тип_результата класс :: operator знак_операции(список параметров)

(оператори тіла операторної функції}

При необхідності можна вказувати прототип операторної функції наступного формату :

тип_результату operator знак_операції(список параметрів);тип_результата operator знак_операции(список параметров);

Приклад 1. Перевантаження операцій.**Пример 1.** Перегрузка операций.

Визначимо клас комплексних даних і для цього класу перевизначимо стандартні операції ==, < і >.

```
#include <iostream.h>
#include <math.h>
#define FALSE 0
#define TRUE 1
class Complex
  (
    private:
    double real;
    double image;
    public:
    Complex (double r){real=r;image=0;} //конструктор для одного параметраComplex (double r) {real=r;image=0;} //конструктор для одного параметра
    Complex (double r, double i)  (real=r;image=i;) //конструктор для 2Complex (double r,double i) (real=r;image=i;) //конструктор для 2
            // параметрів
    -Complex(void){ }
    int operator == (Complex&); //прототип операторної функції ==int operator == (Complex&); //прототип операторной функции ==
```

```
int operator > (Complex&); //прототип операторної функції >int operator > (Complex&); //прототип операторной функции >
int operator < (Complex&); //прототип операторної функції <int operator < (Complex&); //прототип операторной функции <
};
int Complex::operator == (Complex &c) //визначення операторної функції ==int Complex::operator == (Complex &c) //определение операторной функции ==
{
  return ((sqrt(real*real+iroage*image)==
   sqrt(c. reai*c.real - t - c.image*c.image))? TRUE: FALSE);sqrt(c.reai*c.real-t-c.image*c.image)) ? TRUE:FALSE);
}
int Complex::operator >(Complex &C) //визначення операторної функції >int Complex::operator >(Complex &C) //определение операторной функции >
{
  return((sqrt(real*real+image*image)>
   sqrt(C.real*C.real+C.image*C.image))? TRUE: FALSE);
}
int Complex::operator < (Complex &C) //визначення операторної функції <int Complex::operator < (Complex &C) //определение операторной функции <
{
  return ((sqrt (real^real+image^image)<
   sqrt(C.real*C.real+C.image*C.image))?TRUE: FALSE);
}
main()
{
  Complex cl(5.2,10.1)/c2(5.2,10.1);
  if (cl==c2) cout<<"cl equal c2"<<endl;
  if(cl < c2) cout<<"cl < c2"<<endl;
  if(cl > c2) cout<<"cl > c2"<<endl;
}
```

У наведеному прикладі ми виконали перевизначення стандартних операцій < і > (стандартно використовуваних для чисельних даних)забезпечивши можливість їх застосування для комплексних даних. Виклик необхідних екземплярів операцій здійснюватиметься залежно від типу операндів (у нашому прикладі операнди є комплексними).В приведенном примере мы выполнили переопределение стандартных операций ", < и > (стандартно используемых для численных данных), обеспечив возможность их применения для комплексных данных. Вызов требуемых экземпляров операций будет осуществляться в зависимости от типа операндов (в нашем примере операнды являются комплексными).

Використання знаку операції є *скороченою формою запису* виклику операторної функції, яка може викликатися як будь-яка інша функція. Наприклад:Использование знака операции представляет собой *сокращенную форму записи* вызова операторной функции, которая может вызываться как любая другая функция. Например:

```
c1==c2;                //скорочена форма виклику
```

cl.operator==(c2); //повна форма виклику функції-операціїcl.operator==(c2); //полная форма вызова функции-операции

На виконання перевантаження стандартних операцій накладаються наступні обмеження:

- не можуть бути переобтяжені наступні символи препроцесора : # і ##;
- оператори ^*[ ], ( ) і* -> можуть бути переобтяжені тільки як нестатичні компонентні функції. Ці оператори не можуть бути переобтяжені для перераховуваних типів *enum;*операторы ^ *[ ], ( ) и* -> могут быть перегружены только как нестатические компонентные функции. Эти операторы не могут быть перегружены для перечисляемых типов *enum;*
- звичайні пріоритети операцій і порядок відповідності параметрів і аргументів (правила асоціації) залишаються незмінними при використанні переобтяженої операції;
- переобтяжена операція не може змінити поведінку по відношенню до вбудованих типів даних.

Приклад 2. Використання перевантаження операцій.**Пример 2.** Использование перегрузки операций.

Розглянемо приклад, що ілюструє переваги об'єктно-орієнтованого підходу для обчислювальних завдань, заснованих на матричних перетвореннях. Визначимо клас *matrix* (динамічний двовимірний масив). Для вказаного класу перевантажимо стандартні операції множення, ділення, складання і віднімання. Таке перевантаження операторів дозволить надалі писати програми з матричними обчисленнями на C++ подібно до того, як пишуться програми для обробки простих числових даних. Приклад використання класу *matrix* міститься у кінці програми.Рассмотрим пример, иллюстрирующий преимущества объектно-ориентированного подхода для вычислительных задач, основанных на матричных преобразованиях. Определим класс *matrix* (динамический двумерный массив). Для указанного класса перегрузим стандартные операции умножения, деления, сложения и вычитания. Такая перегрузка операторов позволит в дальнейшем писать программы с матричными вычислениями на C++ подобно тому, как пишутся программы для обработки простых числовых данных. Пример использования класса *matrix* содержится в конце программы.

```
// Файл matrix.hppФайл matrix.hpp
#include <iostream.h>
#include <math.h>
#include <stdlib.h>
const int ERR_EXIT = - 1;
// якщо число менше IS ZERO/если число меньше IS ZERO/
const double IS_ZERO = 1E-5;          // воно вважається нульовимconst double IS_ZERO = 1E-5;          // оно считается нулевым
class matrix
  {
   private:
    unsigned int
```

ROWS, // число рядківROWS, // число строк
COLS; // число стовпцівCOLS; // число столбцов
float **point; // власне ці матриціfloat **point; // собственно данные матрицы
protected:protected:
float Det2x2(); // особливий випадок - матриця 2x2float Det2x2(); // особый случай - матрица 2x2
public:
matrix (unsigned int rows, unsigned int cols); // конструкторmatrix (unsigned int rows,unsigned int cols); // конструктор
matrix(const matrix &REF); // конструктор копіюванняmatrix(const matrix &REF); // конструктор копирования
-matrix(); // деструкціяmatrix(); // деструктор
matrix operator + (matrix &B);
matrix operator - (matrix &B);
matrix operator * (matrix &B);
matrix operator !(); // транспонуванняmatrix operator !(); // транспонирование
matrix operator ^(matrix &B); // рівністьmatrix operator ^(matrix &B); // равенство
inline matrix operator * (float a); // Множення на скалярinline matrix operator * (float a); // Умножение на скаляр
inline matrix operator / (float a); // Ділення на скалярinline matrix operator / (float a); // Деление на скаляр
inline matrix operator * (int a);
inline matrix operator / (int a);
// обнулення
inline matrix set_ZERO();
inline matrix swap_rows(unsigned int rowl, unsigned int row2);
inline matrix swap_cols(unsigned int col1, unsigned int col2);
matrix Menor (unsigned int row, unsigned int col);// обчисленняmatrix Menor (unsigned int row,unsigned int col);// вычисление
// мінору
void Out(); // Виведення вмісту matrix void Out(); // Вывод содержимого matrix
input(); // введення значеньinput(); // ввод значений
public:
friend float Det(matrix &B); // визначникfriend float Det(matrix &B); // определитель

```
friend matrix Adj(matrix &B);      // приведенняfriend matrix Adj(matrix &B);    // приведение
friend matrix Inv (matrix &B);     // інвертуванняfriend matrix Inv (matrix &B);    // инвертирование
};
//===================================================================
// конструктор
matrix: rmatrix(unsigned int rows, unsigned int cols)
 (
  unsigned int i;
  ROWS=rows;COLS=cols;
  if ( !( point = new float*[ROWS] ) )
   {
    cerr << "Неможливо розмістити матрицю в пам'яті";cerr << "Невозможно разместить матрицу в памяти";
    exit(ERR_EXIT);
   }
   for (i=0;i<ROWS;i++)
    if (!( point [i]- new float[COLS] ) )
     cerr << "Неможливо розмістити матрицю в пам'яті";cerr << "Невозможно разместить матрицу в памяти";
     exit(ERR_EXIT);exit(ERR_EXIT);
 }
}
//==========================================================
// Конструктор копіювання
matrix::matrix(const matrix &REF)matrix::matrix(const matrix &REF)
 (
  unsigned int i, j;
  // Розміщуємо нову матрицю
  ROWS=REF.ROWS;COLS=REF.COLS ;
 if ( !( point = new float*[ROWS] ) )
   {
     cerr << "Неможливо розмістити матрицю в пам'яті";
     exit (ERR_EXIT);
   }
  for (i=0;i<ROWS;i++)
    if ( !( point [i]=new float[COLS] ) )(
      cerr << "Неможливо розмістити матрицю в пам'яті";cerr << "Невозможно разместить матрицу в памяти";
      exit(ERR_EXIT);exit(ERR_EXIT);
    }
  // Копіюємо матрицю в нове розташування
```

```
    for (j=0;j<COLS;j++)
     for (i=0;i<ROWS;i++)
        point[i][j]=REF.point[i][j];
  }
//=============================================================
====
/* Привласнення (=) */Присваивание (=) */
matrix matrix::operator^(matrix &B)
  {
    unsigned int i, j;
    for (i=0;i<COLS;i++)
       delete(point[i]);
    delete (point);
    ROWS=B.ROWS; COLS=B.COLS ;
   if ( !( point = new float*[ROWS] ) ){
      cerr << "Неможливо розмістити матрицю в пам'яті";cerr << "Невозможно разместить матрицу в памяти";
      exit(ERR_EXIT);
   }
   for (i=0;i<ROWS;i++)
     if ( !( point [i]=new float[COLS] ) ){
        cerr << "Неможливо розмістити матрицю в пам'яті";cerr << "Невозможно разместить матрицу в памяти";
        exit (ERR_EXIT);
     }
   for (j=0;j<B.ROWS;j++)
      for (i=0;i<B.COLS;i++)
          point[i][j]=B.point[i][j];
   return *this;
  }
//=============================================================
===
/* Деструкція */Деструктор */
matrix::~matrix()
  {
    unsigned int i;
    for (i=0;i<ROWS;i++)
       delete(point[i]);
    delete (point);
  }
//=============================================================
====
/* Транспонування (!) */Транспонирование (!) */
matrix matrix::operator !()
```

```
 {
  unsigned int i, j;
  matrix TEMP(ROWS, COLS);
  for (j=0;j<COLS;j++)
   for (i=0;i<ROWS;i++)
     TEMP.point[j][i]=point[i][j];
  return TEMP;
 }
//============================================================
====
/* Обнулення */Обнуление */
inline matrix matrix::set_ZERO()
 {
  unsigned int i, j;
  for (j=0;j<COLS;j++)
   for (i=0;i<ROWS;i++)
     point[i][j]=0;
  return *this;
 )
//============================================================
====
/*   Операція віднімання  (-) */Операция вычитания  (-) */
matrix matrix::operator -(matrix &B)
 {
  unsigned int i, j;
  if ((COLS!=B.COLS)||(ROWS!=B.ROWS)) (
     cerr << "Матриця не сумісна з операцією -".;cerr << "Матрица не
совместима с операцией -.";
     exit (ERR_EXIT);
  )
  matrix З(ROWS, COLS);matrix C(ROWS,COLS);
  for (j=0;j<COLS;j++)
   for (i=0;i<ROWS;i++)(
      C.point[i] [j]==point[i] [ j]- B.point [i] [j];
   }
  return C;
}
//============================================================
====
/*  Операція складання (+)  */Операция сложения (+)  */
matrix matrix::operator+(matrix &B)
 {
  unsigned int i, j;
  if ((COLS!=B.COLS)||(ROWS!=B.ROWS))(
```

```
    cerr « »Matrix aren't compatible bellow +«.;
    exit (ERR^EXIT);
  }
  matrix C(ROWS, COLS);
  for (j=0;j<COLS;j++)
   for (i⁼=0;i<ROWS;i++){for (i⁼=0;i<ROWS;i++) {
    C.point[i][j]=B.point[i][j]+point[i][j];
   }
  return C;
 }
//===================================================================
====
/* Операція множення (*) */Операция умножения (*) */
matrix matrix::operator * (matrix &B)
 {
  unsigned int i, j, k;
  if (COLS!=B.ROWS)(
   cerr << "Матриця не сумісна з операцією *".;
   exit (ERR_EXIT);
  }
  matrix M (ROWS, B. COLS);matrix M (ROWS, B.COLS);
  M.set_ZERO();        // обнуляємо M
  // множення:
  for (j=0;j<B.COLS;j++)
   for (i=0;i<ROWS;i++)
    for (k=0;k<B.ROWS;k++)
     M.point[i][j]=M.point[i][j]+point[i][k]*B.point[k][j];
  return M;
 }
//===================================================================
====
/* Інвертування Inv(M) */
matrix Inv(matrix &B)
 {
  unsigned int i, j;
  float det;
  matrix InvM(B.ROWS, B.COLS);
  det=Det(B);
  if (det==0){
   cerr<<"Матриця не має зворотної";
   exit(ERR_EXIT);
  }
  // інвертуємо
  InvM=(Adj(!B))/det;
```

```
    return InvM;
   }
  //======================================================================
====
  /* Детермінант матриці 2x2 (Det2x2) */
  float matrix::Det2x2()
   {   // особливий випадокособый случай
    float del;
    det=point[0][0]*point[l][1]- point[0][1]*point[l][0];
    return del;
   )
  //======================================================================
====
  /* Детермінант матриці Det (М) */Детерминант матрицы Det (М) */
  float Det(matrix &B)
   (
    unsigned int n;
    int signo;
    float det=0;
    if (B.ROWS!=B. COLS)(if (B.ROWS!=B.COLS) (
      cегг<<"Матриця повинна бытй квадратною"!;
      exit(ERR_EXIT);
    }
    else
     if (B.ROWS==1)
       return B.point[0][0];
    else
      if (B.ROWS=2)
        return B.Det2x2();
      else
        for (n=0;n<B.COLS;n++)(
            // перевірка на парність, для парних стовпців
            // знак = +, непарних — знак = -
           (п&1)==0 ? (signo=l):(signo=-l);п&1)==0 ? (signo=l):(signo=-l);
           det=det+signo*B.point[0][n]*Det(B.Menor(0, n));
        }
    return det;return det;
   }
  //======================================================================
====
  /* Множення матриці на скаляр */
  matrix matrix::operator*(float a)matrix matrix::operator*(float a)
   {
     unsigned int i, j;
```

```
        for (j=0;j<COLS;j++)
          for (i=O;i<ROWS;i++) point[i][j]=a*point[i][j];
            return *this;return *this;
    }
  /* Множення матриці на цілочисельний скаляр */
  matrix matrix::operator*(int a)matrix matrix::operator*(int a)
    {
      unsigned int i, j;
      for (j=0;j<COLS;j++)
        for (i=0;i<ROWS;i++)
          point[i][j]=a*point[i][j];
      return *this;
    }
  //=================================================================
====
  /* Ділення матриці на скаляр */
  matrix matrix::operator /(float a)matrix matrix::operator /(float a)
    {
      unsigned int i, j;
      for (j=0;j<COLS;j++)
        for (i=0;i<ROWS;i++)
          point[i][j]=point[i][j]/a;
      return *this;
    }
  /* Ділення матриці на цілочисельний скаляр */
  matrix matrix::operator /(int a)matrix matrix::operator /(int a)
    {
       unsigned int i, j;
       for (j=0;j<COLS;j++)
         for (i=0;i<ROWS;i++)
           point [i] [j]=point[i] [j]/a/return *this;
    }
  //=================================================================
====
  /* Обчислення мінору матриці */Вычисление минора матрицы */
  matrix matrix::Menor(unsigned int a, unsigned int b)
    (
      unsigned int i, j, p, q;
      matrix MEN(ROWS - 1, COLS - 1);
      for (j=0, q=0;q<MEN.COLS;j++, q++)
        for (i=0, p=0;p<MEN.ROWS;i++, p++){
          if (i==a) i++;
          if (j==b) j++;
          MEN.point[p][q]=point[i][j];
```

```
        }
        return MEN;
      }
    //=====================================================================
====
    /* Приведення матриці */Приведение матрицы */
    matrix Adj (matrix &B)
      (
        unsigned int i, j;
        float signo;
        matrix ADJ(B.ROWS, B.COLS);
        for (j=0;j<B.COLS;j++)
          for (i - 0;i<B.ROWS;i++)(
            // перевірка на парність, для парних знак=+, непарних, - знак=-
            ((i+jl&li==O ? (signo=l) : (signo=-l);
            ADJ.point[i] [j]==signo*Det(B.Menor(i, j));
            }
        return ADJ;
      }
    //=====================================================================
====
    /* Введення даних */Ввод данных */
    matrix matrix::input ()
      {
        unsigned int i, j;
        cout << "Введіть матрицю"";cout << "Введите матрицу"";
        cout <<ROWS << 'X'<< COLS <<" :"<<";cout <<ROWS << 'X'<< COLS
<<" :"<<'\n';
        flush(cout);
        for (i=0;i<ROWS;i++)for (i=0;i<ROWS;i++)
          for (j=0;j<COLS;j++){
            ci>>point[i][j];
        }
        return *this;
      )
    //=====================================================================
====
    /* Висновок */Вывод */
    void matrix::0ut()
      {
        unsigned int i, j;
        cout << endl;
        for (i=0;i<ROWS;i++){
          for (j=0;j<COLS;j++)
```

```
        cout<< point [i][j]  << " ";cout<< point [i][j] << " ";
        cout<< endl;
      }
    flush(cout);
  )
//=================================================================
====
/* Перестановка рядків і стовпців */
matrix matrix::swap_rows(unsigned int rowl, unsigned int row2)
  {
    unsigned int j;
    float *FILA=new (float);
    if ( (rowl>ROWS)||(row2>ROWS) ){
      cегг<<"Неможливо переставити неіснуючі рядки"!;
      exit(ERR^EXIT);
    }
    FILA=point[rowl];
    point[rowl]=point[row2];
    point[row2]=FILA;
    return *this;
  }
matrix matrix::swap_cols(unsigned int coil, unsigned int col2)
  {
    unsigned int i;
    float COLUMN;
    if ((coll>COLS) || (col2>COLS) ){
      cerr<<"Неможливо переставити неіснуючі стовпці"!;
      exit(ERR_EXIT);
    }
    for (i=0;i<ROWS;i++){
      COLUMN=point[i][coil];
      point[i][coll]=point[i][col2];
      point[i][col2]=COLUMN;
    }
    return *this;
  )
//=================================================================
====
// Приклад роботи програми
//=================================================================
====
#include "matrix.hpp"include "matrix.hpp"
#include <stdlib.h>
void main ()
```

```
{
 unsigned int i, j;
 matrix A(4,4), B(4,4), З(4, 4);matrix A(4,4), B(4,4), C(4, 4);
 A.input();
 Cout<<"A= \n";
 A.Out();
 B=Inv(A);
 Cout<<"B = \n";
 B. Out();B.Out();
 З=A*B;
 Cout<<"Результат множення A*B: \п";
 C.Out();
 cout<<"Введіть множник i = \u1087?";cout<<"Введите множитель i = \п";
 cin >> i;cin >> i;
 З=З*i;
 Cout<<"Результат множення : ";Cout<<"Результат умножения: \n";
 C.Out();C.Out();
 З=!З;
 Cout<<"Результат транспонування: \п";
 C.Out();
 Cout<< "Детермінант A : "<<Det(A) <<"\п";
}
```

Як випливає з наведеного прикладу, на основі використання визначеного нами класу робота з матрицями (порівняно з визначенням звичайних процедур і функцій з передачею параметрів) виявляється дуже простою і може принести задоволення.

4.4 Віртуальні функції

Віртуальна функція *(virtual function)* є компонентною функцією базового класу, яка перевизначається в похідному класі. Використання віртуальних функцій, на відміну від перевантаження функцій, забезпечує *динамічний поліморфізм,* що реалізовується на етапі виконання програми.***Виртуальная функция*** *(virtual function)* представляет собой компонентную функцию базового класса, которая переопределяется в производном классе. Использование виртуальных функций, в отличие от перегрузки функций, обеспечивает *динамический полиморфизм,* реализуемый на этапе выполнения программы.

При оголошенні віртуальної функції у базовому класі перед її ім'ям вказується ключове слово ***virtual.*** У похідному класі віртуальна функція перевизначається. Кожне таке перевизначення *(overriding)* віртуальної функції в похідному класі означає створення конкретного методу. При перевизначенні віртуальної функції в похідному класі ключове слово *virtual* не вказується.При объявлении виртуальной функции в базовом классе перед ее именем указывается ключевое слово ***virtual.*** В производном классе виртуальная функция переопределяется. Каждое такое переопределение *(overriding)* виртуальной

функции в производном классе означает создание конкретного метода. При переопределении виртуальной функции в производном классе ключевое слово *virtual* не указывается.

Віртуальну функцію можна викликати як будь-яку іншу компонентну функцію. Проте для підтримки динамічного поліморфізму віртуальні функції викликають через *покажчик базового класу,* що використовується як посилання на об'єкт похідного класу. Якщо об'єкт похідного класу, що адресується таким чином, містить віртуальну функцію і віртуальна функия викликається за допомогою цього покажчика, то при компіляції визначається версія віртуальної функції, що викликається, з урахуванням типу об'єкту, на який посилається покажчик. Визначення конкретної версії віртуальної функції здійснюється в процесі виконання програми.Виртуальную функцию можно вызывать как любую другую компонентную функцию. Однако для поддержки динамического полиморфизма виртуальные функции вызывают через *указатель базового класса,* используемый в качестве ссылки на объект производного класса. Если адресуемый таким образом объект производного класса содержит виртуальную функцию и виртуальная функия вызывается с помощью этого указателя, то при компиляции определяется версия вызываемой виртуальной функции с учетом типа объекта, на который ссылается указатель. Определение конкретной версии виртуальной функции осуществляется в процессе выполнения программы.

Якщо є декілька похідних класів від того, що містить віртуальну функцію базового класу, то при посиланні покажчика базового класу на різні об'єкти цих похідних класів виконуватимуться різні версії віртуальної функції.

Перевизначення віртуальної функції в похідному класі має істотні відмінності від механізму перевантаження функцій. Передусім, перевизначувана віртуальна функція повинна мати ті ж тип, число параметрів і тип повертаного значення, тоді як перевантажувана функція повинна мати відмінності в типі і/або числі параметрів. По-друге, віртуальна функція має бути компонентом класу, що не відноситься до перевантажуваних функцій.*Переопределение* виртуальной функции в производном классе имеет существенные отличия от механизма перегрузки функций. Прежде всего, переопределяемая виртуальная функция должна иметь те же тип, число параметров и тип возвращаемого значения, тогда как перегружаемая функция должна иметь отличия в типе и/или числе параметров. Во-вторых, виртуальная функция должна быть компонентом класса, что не относится к перегружаемым функциям.

Щоб прояснити сенс використання віртуальних функцій, наведемо приклад, що ілюструє різницю між звичайними і віртуальними функціями.

Приклад. Віртуальні і звичайні функції.**Пример.** Виртуальные и обычные функции.

```
#include <iostream.h>
class parent
 (                              //оголошення базового класуобъявление базового класса
   public:
```

```
            int k;
            parent(int a)                // конструктор класуparent(int a)                //
конструктор класса
            {
             k=a;
            }
            virtual void fv()          // віртуальна функція
            {
              cout << "call fv() of base class: ";
              cout << k << endl;
            }
            void f()                   //звичайна функціяvoid f()                   //обычная
функция
             (
              cout << "call f() of base class: ";
              cout << k*k << endl;
             }
         };
        class childl: public parent
          {
           public:
            childl(int x) : parent(x)
            {  }
           void fv()
            {
             cout << "call fv() of derived class childl: ";
             cout << (k + 1) << endl;
            }
           void f() //звичайна функціяvoid f() //обычная функция
            (
             cout << "call f() of derived class childl: ";cout << "call f() of derived class
childl: ";
             cout << (k - 1) << endl;
            }
          );
        int main()
          {
           parent *p; // покажчик на базовий класparent *p; // указатель на базовый
класс
           parent ez(5); // оголошення об'єкту базового класуparent ez(5); //
объявление объекта базового класса
           childl chezl(15); // оголошення об'єкту похідного класуchildl chezl(15); //
объявление объекта производного класса
```

p = &ez; // покажчик посилається на об'єкт базового класур = &ez; // указатель ссылается на объект базового класса

p ->fv(); // виклик функції fv() базового класур->fv(); // вызов функции fv() базового класса

p ->f(); // виклик функції f() базового класур->f(); // вызов функции f() базового класса

p = &chezl; // покажчик посилається на об'єкт похідного класур = &chezl; // указатель ссылается на объект производного класса

p ->fv(); // виклик віртуальної функції fv() похідного класур->fv(); // вызов виртуальной функции fv() производного класса

p ->f(); // виклик звичайної функції f() похідного класур->f(); // вызов обычной функции f() производного класса

return 0;return 0;

)

При виконанні приведеної програми на екран будуть видані наступні результати:

call fv() of base class: 5

call f() of base class: 25

call fv() of derived class childl: 16

call f() of base class: 225

Отримані результати показують, що при зверненні до віртуальної функції *fv()* похідного класу відбувається виклик функції похідного класу. Звернення до звичайної функції *f()* похідного класу призводить до виклику однойменної функції базового класу, а не похідного.Полученные результаты показывают, что при обращении к виртуальной функции *fv()* производного класса происходит вызов функции производного класса. Обращение к обычной функции *f()* производного класса приводит к вызову одноименной функции базового класса, а не производного.

У мові C++ існує поняття ***чистих*** *(pure)* ***віртуальних функцій.*** Такі функції використовуються у разі, коли у віртуальній функції базового класу *відсутня значима дія.* При цьому в кожному похідному класі від заданого класу така функція має бути обов'язково перевизначена.В языке C++ существует понятие ***чистых*** *(pure)* ***виртуальных функций.*** Такие функции используются в случае, когда в виртуальной функции базового класса *отсутствует значимое действие.* При этом в каждом производном классе от заданного класса такая функция должна быть обязательно переопределена.

Чисті віртуальні функції у базовому класі не визначаються, замість цього поміщаються їх прототипи з наступною формою запису :

virtual тип ім'я_функції (список параметрів) = 0;virtual тип имя_функции (список параметров) = 0;

Привласнення імені функції значення 0 вказує на відсутність тіла функції. При такому оголошенні віртуальної функції у базовому класі в кожному похідному класі повинне виконуватися її перевизначення, інакше при компіляції програми буде виявлена помилка.

Розглянуті нами *статичний* і *динамічний* варіанти поліморфізму (перевантаження функцій і використання віртуальних функції відповідно) співвідносять з двома наступними поняттями: раннє зв'язування і пізнє зв'язування.Рассмотренные нами *статический* и *динамический* варианты полиморфизма (перегрузка функций и использование виртуальных функции соответственно) соотносят с двумя следующими понятиями: раннее связывание и позднее связывание.

Раннє зв'язування торкається подій етапу компіляції програми, таких як налаштування при виклику звичайних функцій, перевантажуваних функцій, невіртуальних компонентних функцій і дружніх функцій. При виклику перерахованих функкий уся необхідна адресна інформація відома при компіляції. *Гідністю* раннього зв'язування є висока швидкодія отримуваних здійснимих програм. *Недоліком* раннього зв'язування є зниження гнучкості програм.***Раннее связывание*** касается событий этапа компиляции программы, таких как настройка при вызове обычных функций, перегружаемых функций, невиртуальных компонентных функций и дружественных функций. При вызове перечисленных функкий вся необходимая адресная информация известна при компиляции. *Достоинством* раннего связывания является высокое быстродействие получаемых выполнимых программ. *Недостатком* раннего связывания является снижение гибкости программ.

Пізнє зв'язування торкається подій, що відбуваються в процесі виконання програми. При виклику функцій з використанням пізнього зв'язування адреса функції, що викликається, до початку виконання програми невідома. Зокрема, об'єктом пізнього зв'язування є віртуальні функції. При доступі до віртуальної фукциї через покажчик базового класу при виконанні програми визначається тип засиланого об'єкту і вибирається версія віртуальної функції для виклику. *Гідністю* пізнього зв'язування є висока гнучкість виконуваної програми, можливість реакції на події. *Недоліком* є відносно низька швидкодія програми.***Позднее связывание*** касается событий, происходящих в процессе выполнения программы. При вызове функций с использованием позднего связывания адрес вызываемой функции до начала выполнения программы неизвестен. В частности, объектом позднего связывания являются виртуальные функции. При доступе к виртуальной фукции через указатель базового класса при выполнении программы определяется тип ссылаемого объекта и выбирается версия виртуальной функции для вызова. *Достоинством* позднего связывания является высокая гибкость выполняемой программы, возможность реакции на события. *Недостатком* является относительно низкое быстродействие программы.

Контрольні питання і завдання

1. Дайте визначення поняття поліморфізму.
2. Назвіть приклади прояву поліморфізму в мові С++.
3. Поясніть поняття статичного і динамічного полимирфизма.

4. У чому сенс перевантаження функцій, на чому заснований механізм перевантаження функцій?

5. Приведіть формат запису операторної функції.

6. Вкажіть повну і скорочену форми виклику операторної функцій.

7. Вкажіть обмеження« що накладаються на перевантаження стандартних операцій.

8. Дайте визначення і вкажіть достоїнства і недоліки раннього зв'язування.

9. Дайте визначення і вкажіть достоїнства і недоліки пізнього связвания.

10. У чому сенс віртуальних функцій?

11. Як здійснюється виклик віртуальних функцій?

12. Перерахуйте відмінності віртуальних і перевантажуваних функцій.

13. Вкажіть, які результати будуть виведені на екран, якщо в головній функції (Приклад 1 підрозділу 11.4) помістити наступні рядки:

```
parent ob(10);
childl chob(8);
p = & ob;
P -> fv();
P -> f();
p = & chob;
P  ->  fv();P -> fv();
p -> f();p -> f();
```

14. Порівняйте роботу з динамічними масивами на основі класу *matrix* з підрозділу 11.3 і приклад роботи з динамічними масивами, розглянутий в четвертому розділі 4. Якої з них простіше для неодноразового використання?Сравните работу с динамическими массивами на основе класса *matrix* из подраздела 11.3 и пример работы с динамическими массивами, рассмотренный в четвертом разделе 4. Какой из них проще для неоднократного использования?

15. Скласти програму використання перевантаження функцій, засновану на відмінності типів аргументів *int і float*Составить программу с использование перегрузки функций, основанную на различии типов аргументов *int и float*

16. Скласти програму використання перевантаження стандартних операцій складання і множення для речових і комплексних змінних.

17. Складіть програму з використанням віртуальних функцій для вичислень за наступною формулою:

z= 2+a, якщо $x>0;$z= 2+a, если $x>0;$

z= $a*a,$ якщо $x<=O,$ За умови, що значення x вводиться при виконанні програми.z= $a*a,$ если $x<=O,$ При условии, что значение x вводится при выполнении программы.

5 ОСНОВИ ОРГАНІЗАЦІЇ ВВЕДЕННЯ-ВИВОДУ

Практично будь-яка програма обмінюється інформацією із ***зовнішніми*** облаштуваннями ПЕВМ, які використовуються для введення даних в програму і

виведення результатів. Управляти введенням-виводом можна з програми і на рівні операційної системи (за рамками мови програмування). Ми розглянемо способи роботи з облаштуваннями введення-виводу з програми і відповідні засоби мови С++.Практически любая программа обменивается информацией с ***внешними*** устройствами ПЭВМ, которые используются для ввода данных в программу и вывода результатов. Управлять вводом-выводом можно из программы и на уровне операционной системы (за рамками языка программирования). Мы рассмотрим способы работы с устройствами ввода-вывода из программы и соответствующие средства языка С++.

5.1. Класифікація засобів введення-виводу

У основі класифікації лежить характер використання засобів введення-виведення мови, яка визначається трьома головними чинниками :

- рівнем абстрагування від деталей організації взаємодії із зовнішніми пристроями;
- стилем програмування, який підтримують засоби введення-виводу;
- принципами управління зовнішніми пристроями і організацією файлової системи.

Можна виділити три рівні абстрагування :

- високий, або рівень потоків і об'єктів типу «файл» *(TFile);*высокий, или уровень потоков и объектов типа «файл» *(TFile);*
- середній, або рівень стандартних системних облаштувань введення-виводу (консоль — клавіатура і дисплей, системний принтер);
- низький, або рівень операційної системи при роботі з двійковими файлами і портами введення-виводу.

З позицій ***стилю програмування*** в мові С++ можна виділити *об'єктно-орієнтовану систему введення-виводу* і три успадковані *процедурно-орієнтовані системи введення-виведення* мови : *потокові засоби стандарту ANSI* З, *систему введення-виведення типу UNIX* і *засоби низькорівневого введення-виводу.*С позиций ***стиля программирования*** в языке С++ можно выделить *объектно-ориентированную систему ввода-вывода* и три унаследованные *процедурно-ориентированные системы ввода-вывода* языка С: *потоковые средства стандарта ANSI* С, *систему ввода-вывода типа UNIX* и *средства низкоуровневого ввода-вывода.*

Усі засоби введення-виводу в мові С++ реалізовані у вигляді ***бібліотечних*** переобтяжених операцій << і >> (витягання і вставки), маніпуляторів, класів потоків, констант, глобальних змінних, функцій і типів даних.Все средства ввода-вывода в языке С++ реализованы в виде ***библиотечных*** перегруженных операций << и >> (извлечения и вставки), манипуляторов, классов потоков, констант, глобальных переменных, функций и типов данных.

Базові ***об'єктно-орієнтовані*** засоби введення-виводу, що забезпечують роботу з потоками *(streams),* оголошені у файлі ***iostream.h.*** При підключенні файлу ***fstream.h*** стають доступні об'єктно-орієнтовані засоби обміну даними між програмою і файлами через потоки. У файлі ***sstream.h (strstrea.h*** — в ранніх

версіях Borland C++) оголошені засоби для обміну даними з областю пам'яті, яка розглядається як масив символів або як рядок типу *string.*Базовые ***объектно-ориентированные*** средства ввода-вывода, обеспечивающие работу с потоками *(streams),* объявлены в файле ***iostream.h.*** При подключении файла ***fstream.h*** становятся доступны объектно-ориентированные средства обмена данными между программой и файлами через потоки. В файле ***sstream.h (strstrea.h*** — в ранних версиях Borland C++) объявлены средства для обмена данными с областью памяти, которая рассматривается как массив символов или как строка типа *string.*

Процедурно-орієнтовані засоби високого рівня реалізовані за допомогою бібліотеки стандартного введення-виводу (ANSI C) і стають доступними при підключенні заголовного файлу ***stdio.h.Процедурно-ориентированные*** средства высокого уровня реализованы с помощью библиотеки стандартного ввода-вывода (ANSI C) и становятся доступными при подключении заголовочного файла ***stdio.h.***

У багатьох реалізаціях C++ засобу роботи із стандартними системними облаштуваннями введення-виводу середнього ***рівня для*** DOS доступні при підключенні файлу conio.***h (***від англ. console — клавіатура і монітор).Во многих реализациях C++ средства работы со стандартными системными устройствами ввода-вывода ***среднего уровня*** для DOS доступны при подключении файла ***conio.h*** (от англ. *console* — клавиатура и монитор).

Низькорівневі засоби введення-виводу оголошені у файлах ***bios.h, direct.h, dirent.h, dos.h, io.h*** і ряду інших. Приклади організації введення-виводу на низькому рівні можна знайти в системі допомоги (Help) інтегрованого середовища програмирования.***Низкоуровневые средства*** ввода-вывода объявлены в файлах ***bios.h, direct.h, dirent.h, dos.h, io.h*** и ряде других. Примеры организации ввода-вывода на низком уровне можно найти в системе помощи (Help) интегрированной среды програмирования.

Надалі ми розглядатимемо *стандартні засоби* ***высокоуровнего*** введення-виводу. Їх використання припускає засвоєння двох найважливіших абстрактних понять «Потоку і »файл« і принципів організації обміну даними програми з файлами через потоки.В дальнейшем мы будем рассматривать *стандартные средства* ***высокоуровнего*** ввода-вывода. Их использование предполагает усвоение двух важнейших абстрактных понятий «потока и «файл» и принципов организации обмена данными программы с файлами через потоки.

Відмітимо, що в основі усіх абстрактних понять введення-виводу на мові C++ лежать принципи організації файлової системи і порядок роботи з файловими пристроями.

5.2 Принципи роботи з потоками і файлами

Потік *(stream)* можна визначити як абстрактний канал зв'язку, який створюється в програмі для обміну даними з файлами. Це поняття введене для того, щоб можна було не враховувати деталі фізичної організації каналу зв'язку між джерелом і приймачем інформації.***Поток*** *(stream)* можно определить как

абстрактный канал связи, который создается в программе для обмена данными с файлами. Это понятие введено для того, чтобы можно было не учитывать детали физической организации канала связи между источником и приемником информации.

У усіх створюваних в програмі потоків є загальні поведінкові властивості. Тому однакові засоби (операції і функції) можуть застосовуватися для роботи з різними потоками. Відмінності потоків визначаються специфікою їх створення і використання.

Другим найважливішим поняттям є файл. ***Файл*** *(file)* є пойменованою сукупністю даних, що знаходиться на зовнішньому пристрої і має певні атрибути (характеристики). Правила іменування файлів визначаються файловою системою.Вторым важнейшим понятием является файл. ***Файл*** *(file)* представляет собой поименованную совокупность данных, находящуюся на внешнем устройстве и имеющую определенные атрибуты (характеристики). Правила именования файлов определяются файловой системой.

Зазвичай файл розглядається як *послідовність* байтів або символів, тому файл має початок (перед першим байтом), кінець (після останнього байта) і при просуванні від початку до кінця кожен байт знаходиться в певній *позиції.* Початок файлу і перший байт мають нульову позицію, кожен подальший байт має позицію на одиницю більше за попередній. Позиція кінця файлу дорівнює розміру файлу у байтах.Обычно файл рассматривается как *последовательность* байтов или символов, поэтому файл имеет начало (перед первым байтом), конец (после последнего байта) и при продвижении от начала к концу каждый байт находится в определенной *позиции.* Начало файла и первый байт имеют нулевую позицию, каждый последующий байт имеет позицию на единицу больше предыдущего. Позиция конца файла равна размеру файла в байтах.

Високорівневі засоби введення-виведення С++ дозволяють працювати з *текстовими і бінарними файлами* після того, як в програмі встановлений їх взаємозв'язок з *текстовими* і *бінарними потоками* відповідно. Файл, що розглядається як послідовність рядків символів, разделеных *непробільними* символами, називається ***текстовим.*** Окрім символу «пропуск» пробільними символами є спеціальні символи '', ', '*\u1087?*', '', '' — табуляція (горизонтальна і вертикальна), новий рядок, повернення каретки і переклад формату (сторінки) відповідно. Серед пробільних символів виділяється символ «*новий рядок»,* який використовується для стандартного розділення сукупностей рядків на «файлові рядки» («лінії» — *line),* у файлі він представляється двома символами ' ' і ' '. *Рядки символів* в *текстових файлах* є послідовностями непробільних символів, які можуть інтерпретуватися при введенні як дані певного типу, представлені в деякому форматі. Наприклад, послідовність символів 125, обмежена з двох сторін пробільними символами, може бути перетворена при введенні в дійсне число типу *double* або ціле число типу типу *int.*Высокоуровневые средства ввода-вывода С++ позволяют работать с *текстовыми и бинарными файлами* после того, как в программе установлена их взаимосвязь с *текстовыми* и *бинарными потоками* соответственно. Файл, рассматриваемый как последовательность строк

символов, разделеных *непробельными* символами, называется ***текстовым.*** Кроме символа «пробел» пробельными символами являются специальные символы '*t*' , '*v*', '*n*', '\r', '*f*' — табуляция (горизонтальная и вертикальная), новая строка, возврат каретки и перевод формата (страницы) соответственно. Среди пробельных символов выделяется символ "*новая строка",* который используется для стандартного разделения совокупностей строк на «файловые строки» («линии» — *line),* в файле он представляется двумя символами ' \r' и ' \n'. *Строки символов* в *текстовых файлах* представляют собой последовательности непробельных символов, которые могут интерпретироваться при вводе как данные определенного типа, представленные в некотором формате. Например, последовательность символов 125, ограниченная с двух сторон пробельными символами, может быть преобразована при вводе в вещественное число типа *double* или целое число типа типа *int.*

Файл називається ***бінарним,*** якщо з ним працюють як з послідовністю байтів або символів. Бінарні потоки і файли зазвичай використовуються при перевантаженні операцій витягання (>>) і вставки (<<) для призначених для користувача типів даних або реалізації специфічних методів введення-виводу для призначених для користувача класів.Файл называется ***бинарным,*** если с ним работают как с последовательностью байтов или символов. Бинарные потоки и файлы обычно используются при перегрузке операций извлечения (>>) и вставки (<<) для пользовательских типов данных или реализации специфических методов ввода-вывода для пользовательских классов.

В якості файлів розглядаються не лише файли на дисках, але і будь-які пристрої (файлові), з якими можна здійснювати операції введення-виводу. Так, файлами є клавіатура і дисплей, а також модем, принтер і інші підключені зовнішні пристрої.

Не усі файли мають однакові поведінкові властивості. Це пов'язано з призначенням і фізичною реалізацією файлу. Наприклад, для файлів на CD ROM коректні тільки операції введення. Тому при *зв'язуванні* потоку з певним файлом *потік набуває властивість* цього файлу. Така поведінка потоку дозволяє говорити про запис в потік і читання з потоку, що еквівалентно запису у файл і читанню з файлу. Якщо властивості файлу конфліктують з властивостями зв'язуваного потоку, то виникає виняткова ситуація — помилка відкриття файлу.Не все файлы имеют одинаковые поведенческие свойства. Это связано с назначением и физической реализацией файла. Например, для файлов на CD ROM корректны только операции ввода. Поэтому при *связывании* потока с определенным файлом *поток приобретает свойства* этого файла. Такое поведение потока позволяет говорить о записи в поток и чтении из потока, что эквивалентно записи в файл и чтению из файла. Если свойства файла конфликтуют со свойствами связываемого потока, то возникает исключительная ситуация — ошибка открытия файла.

При роботі з потоками і файлами розрізняють уведення-виведення, що ***буферизує* і *не буферизує*** (без використання буферів). ***Буфер*** *(buffer)* є областю оперативної пам'яті, яку використовують засоби введення-виводу для проміжного

зберігання даних, що передаються між програмою і зовнішнім пристроєм. Буфер може бути системним. Область пам'яті, де розміщуються *системні буфери,* належить операційній системі, а не програмі.При работе с потоками и файлами различают ***буферизированный*** **и** ***небуферизированный*** (без использования буферов) ввод-вывод. ***Буфер*** *(buffer)* представляет собой область оперативной памяти, которую используют средства ввода-вывода для промежуточного хранения данных, передаваемых между программой и внешним устройством. Буфер может быть системным. Область памяти, где размещаются *системные буферы,* принадлежит операционной системе, а не программе.

Виведення даних в потік з буфером призводить до висновку цих даних у відповідний файл тільки після заповнення буфера. Виведення даних в потік, що не буферизує, призводить до негайного виводу у файл. Використання буферів дозволяє прискорити роботу потоку шляхом поблочного (не побайтного) обміну даними. В той же час, наявність буфера призводить до накладних витрат. По-перше, при введенні даних слід враховувати можливість помилок і, отже, вимагається контролювати і забезпечувати коректність утримуваного буфера перед виконанням чергової операції введення. По-друге, в умовах збоїв і можливості примусового (аварійного) завершення програми при виведенні даних необхідно забезпечувати своєчасне збереження утримуваного буфера у файлі. І нарешті, якщо виникає необхідність синхронізації роботи потоків, то наявність буферів також повинна враховуватися.Вывод данных в поток с буфером приводит к выводу этих данных в соответствующий файл только после заполнения буфера. Вывод данных в небуферизованный поток приводит к немедленному выводу в файл. Использование буферов позволяет ускорить работу потока путем поблочного (не побайтного) обмена данными. Вместе с тем, наличие буфера приводит к накладным расходам. Во-первых, при вводе данных следует учитывать возможность ошибок и, следовательно, требуется контролировать и обеспечивать корректность содержимого буфера перед выполнением очередной операции ввода. Во-вторых, в условиях сбоев и возможности принудительного (аварийного) завершения программы при выводе данных необходимо обеспечивать своевременное сохранение содержимого буфера в файле. И наконец, если возникает необходимость синхронизации работы потоков, то наличие буферов также должно учитываться.

Розглянемо детальніше види і загальні властивості потоків, які дозволяють створювати стандартні засоби С++.

По *напряму передачі даних* розрізняють наступні потоки:**По *направлению передачи данных*** различают следующие потоки:

- *вхідні,* з яких читаються (витягаються) дані в змінні програми — *потоки введення (input stream);входные,* из которых читаются (извлекаются) данные в переменные программы — *потоки ввода (input stream);*
- *вихідні,* в які записуються (вставляються) значення з програми, — *потоки виводу (output stream);выходные,* в которые записываются (вставляются) значения из программы — *потоки вывода (output stream);*
- *двонаправлені,* допускаюче читання і запис — *потоки введення-виводу (input — output stream).двунаправленные,* допускающие чтение и запись — *потоки ввода-вывода (input-output stream).*

Вхідний потік не може бути пов'язаний з файлом, який передбачає тільки запис. Прикладом такого файлу є принтер або монітор. *Вихідний потік* не може бути пов'язаний з файлом, який має атрибут «тільки для читання».*Входной поток* не может быть связан с файлом, который предусматривает только запись. Примером такого файла является принтер или монитор. *Выходной поток* не может быть связан с файлом, который имеет атрибут «только для чтения».

Двонаправлені потоки мають властивості як вхідних, так і вихідних потоків. Потоки такого класу має сенс використати для файлів, до яких можна організувати *довільний доступ. З* файлами, що зберігаються на дискетах або жорстких дисках і не мають атрибуту «тільки для читання», можна працювати через потік таким чином. Перед операцією запису-читання чергової порції даних можна установить необхідну *позицію* у файлі, починаючи з якої і буде осуществлятся черговий обмін даними. Таке *позиціонування* забезпечує доступ до довільного байта, а не тільки до чергового наступного байта, як це має місце при звичайному *послідовному доступі до* файлу. Враховуючи, що потік і пов'язаний з ним файл представляють в програмі єдиний об'єкт, зазвичай говорять тільки про *позиціонування (seek) потоку.*Двунаправленные потоки имеют свойства как входных, так и выходных потоков. Потоки такого класса имеет смысл использовать для файлов, к которым можно организовать *произвольный доступ. С* файлами, хранящимися на дискетах или жестких дисках и не имеющими атрибута «только для чтения», можно работать через поток следующим образом. Перед операцией записи-чтения очередной порции данных можно устанавить требуемую *позицию* в файле, начиная с которой и будет осуществлятся очередной обмен данными. Такое *позиционирование* обеспечивает доступ к произвольному байту, а не только к очередному следующему байту, как это имеет место при обычном *последовательном доступе к* файлу. Учитывая, что поток и связанный с ним файл представляют в программе единый объект, обычно говорят только о *позиционировании (seek) потока.*

За *способом створення* розрізняють потоки наступних видів :**По *способу создания*** различают потоки следующих видов:

- *автоматично створювані* потоки, які пов'язані із стандартними системними облаштуваннями введення-виводу **(*стандартні*** *потоки введення і виводу);автоматически создаваемые* потоки, которые связаны со стандартными системными устройствами ввода-вывода ***(стандартные*** *потоки ввода и вывода);*
- *явно створювані* потоки для організації обміну даними з файлами;*явно создаваемые* потоки для организации обмена данными с файлами;
- *явно створювані* потоки для обміну даними з рядком в оперативній пам'яті.*явно создаваемые* потоки для обмена данными со строкой в оперативной памяти.

Уведення-виведення, що виконується з використанням стандартних потоків, зазвичай називають ***консольним*** введенням-виводом.Ввод-вывод, выполняемый с использованием стандартных потоков, обычно называют ***консольным*** вводом-выводом.

При підключенні файлу ***iostream.h*** в програмі на початку її виконання автоматично створюються 4 ***стандартні потоки:***При подключении файла ***iostream.h*** в программе в начале ее выполнения автоматически создаются 4 ***стандартных потока:***

- ***cin*** — потік, що буферизує, для введення даних із стандартного пристрою (за умовчанням — з клавіатури);***cin*** — буферизированный поток для ввода данных со стандартного устройства (по умолчанию — с клавиатуры);
- ***cout*** — потік, що буферизує, для виведення даних на стандартний пристрій (за умовчанням — на монітор);***cout*** — буферизированный поток для вывода данных на стандартное устройство (по умолчанию — на монитор);
- ***cerr*** **—** потік, що не буферизує, для стандартного висновку повідомлень про помилки (за умовчанням — на монітор);***cerr*** **—** не буферизированный поток для стандартного вывода сообщений об ошибках (по умолчанию — на монитор);
- ***clog*** — потік, що буферизує, для стандартного висновку повідомлень про помилки (за умовчанням — на монітор).***clog*** — буферизированный поток для стандартного вывода сообщений об ошибках (по умолчанию — на монитор).

Зв'язку цих стандартних потоків з файловими пристроями можна змінювати (перепризначувати) на рівні операційної системи або в програмі.

Наприклад, при виконанні команди DOSНапример, при выполнении команды DOS

tstpgm < tstpgm.in > tstpgm.outtstpgm < tstpgm.in > tstpgm.out

будуть перевизначені зв'язки стандартних потоків введення-виводу. Під час виконання програми з файлу tstpgm.*exe* введення через cin буде робиться з файлу tstpgm.*in,* а вивід через cout — *у* файл tstpgm.*out.*будут переопределены связи стандартных потоков ввода-вывода. Во время выполнения программы из файла *tstpgm.exe* ввод через *cin* будет производится из файла *tstpgm.in,* а вывод через *cout — в* файл *tstpgm.out.*

Перенаправлення стандартних потоків можна виконувати також безпосередньо в програмі.*Перенаправление стандартных потоков* можно выполнять также непосредственно в программе.

Використання потоків *cerr* і *clog* спрощує створення програм на мові C++ для середовищ, в яких є спеціальні зовнішні пристрої для негайного висновку повідомлень про помилки (*errors)* через cerr і збереження цих повідомлень у файлі реєстрації помилок (*errors log).*Использование потоков *cerr* и *clog* упрощает создание программ на языке C++ для сред, в которых имеются специальные внешние устройства для немедленного вывода сообщений об ошибках *(errors)* через cerr и сохранения этих сообщений в файле регистрации ошибок *(errors log).*

Потоки для роботи з іншими ***файловими пристроями*** треба в програмі створювати явно, забезпечуючи необхідні їх властивості. Для більшості завдань ці потоки можуть бути об'єктами стандартних класів, оголошених у файлах

fstream.h і *sstream.h (strstrea.h).* Помітимо, що базові властивості усіх потоків містить клас ios, оголошений у файлі *iostream.h.*Потоки для работы с другими ***файловыми устройствами*** нужно в программе создавать явно, обеспечивая необходимые их свойства. Для большинства задач эти потоки могут быть объектами стандартных классов, объявленных в файлах *fstream.h* и *sstream.h (strstrea.h).* Заметим, что базовые свойства всех потоков содержит класс ios, объявленный в файле *iostream.h.*

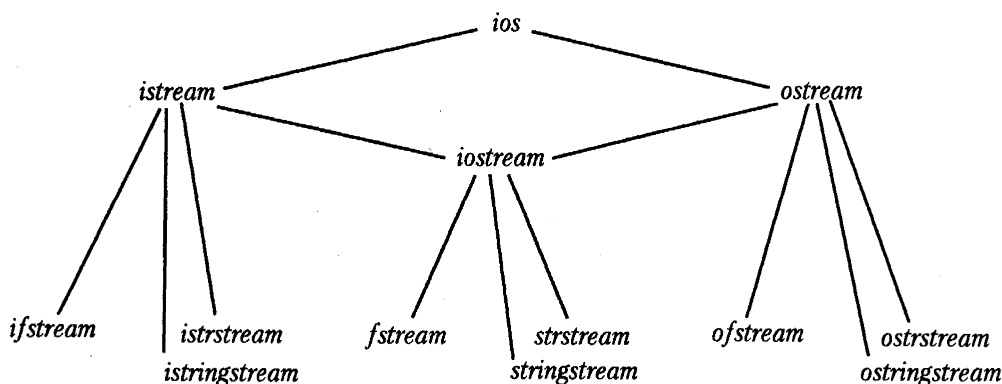
Якщо в програмі вимагається працювати з файлами через потоки, то необхідно підключати файл *fstream.h.* Тоді можна створювати потоки наступних трьох класів :Если в программе требуется работать с файлами через потоки, то необходимо подключать файл *fstream.h.* Тогда можно создавать потоки следующих трех классов:

- ***ifstream*** — для введення з файлів;***ifstream*** — для ввода из файлов;
- ***ofstream*** — для виводу у файл;***ofstream*** — для вывода в файл;
- ***fstream*** — для обміну з файлом в двох напрямах.***fstream*** — для обмена с файлом в двух направлениях.

У файлі ***sstream.h (strstrea.h)*** оголошені класи потоків для обміну даними не із зовнішніми пристроями, а з ***оперативною пам'яттю,*** в якій виділена спеціальна область. Ця область розглядається як масив символів або рядок типу *string.* При підключенні файлу *sstream.h (strstrea.h)* в програмі можна створювати потоки 3+3 аналогічних класів :В файле ***sstream.h (strstrea.h)*** объявлены классы потоков для обмена данными не с внешними устройствами, а с ***оперативной памятью,*** в которой выделена специальная область. Эта область рассматривается как массив символов или строка типа *string.* При подключении файла *sstream.h (strstrea.h)* в программе можно создавать потоки 3+3 аналогичных классов:

- ***istrstream, istringstream;***
- ***ostrstream, ostringstream;***
- ***strstream, stringstream***

для роботи з масивом символів і рядком типу *string* відповідно. На мал. 12.1 приведений фрагмент ієрархії стандартних класів потоків.для работы с массивом символов и строкой типа *string* соответственно. На рис. 12.1 приведен фрагмент иерархии стандартных классов потоков.



Малюнок 12.1 - Стандартні класи потоків

Процес роботи з файлом через потоки можна розбити на 4 етапи:

1. Створення потоку (оголошення змінної).

2. Зв'язування потоку з файлом і відкриття *(open)* файлу для роботи в певному режимі.Связывание потока с файлом и открытие *(open)* файла для работы в определенном режиме.

3. Обмін даними з файлом через потік: запис в потік; читання з потоку; управління станом потоку.

4. Розривши зв'язки потоку з файлом: звільнення буфера («флэширование» — *flush),* закриття *(close)* файлу і розривши його зв'язки з потоком.Разрыв связи потока с файлом: освобождение буфера («флэширование» — *flush),* закрытие *(close)* файла и разрыв его связи с потоком.

При роботі із стандартними потоками дії, вказані в пунктах 1, 2, 4 виконуються автоматично. Стандартні потоки cin*, cout, cerr* і clog називають також **зумовленими** *потоками,* оскільки зв'язок їх з файловими пристроями визначений до початку виконання першого оператора програми. Зазвичай потік свявляется об'єктом спеціального класу istream_*withassign,* а не класу istream*,* як можна бьюо б чекати. Відповідно потоки cout*, cerr,* і clog — це об'єкти класу ostream_*withassign.* Класи стандартних потоків є спадкоємцями класів istream і ostream відповідно.При работе со *стандартными* потоками действия, указанные в пунктах 1, 2, 4 выполняются автоматически. Стандартные потоки *cin, cout, cerr* и *clog* называют также ***предопределенными*** *потоками,* поскольку связь их с файловыми устройствами определена до начала выполнения первого оператора программы. Обычно поток *св*является объектом специального класса *istream_withassign,* а не класса *istream,* как можно бьюо бы ожидать. Соответственно потоки *cout, cerr,* и *clog* — это объекты класса *ostream_withassign.* Классы стандартных потоков являются наследниками классов *istream* и *ostream* соответственно.

За умовчанням передбачається, що при коректному завершенні програми або при виході з області видимості потоку звільнення буфера і закриття файлу здійснюється автоматично. Але з цього не слід робити однозначний висновок про надмірність пункту 4. Подібна надмірність підвищує надійність програм при роботі з файлами.

При роботі з файлами багато що залежить від користувача, який «схильний» припускатися помилки, та і самі файлові пристрої найменш надійний елемент архітектури ЕОМ. Тому при виконанні пунктів 2, 3, 4 слід контролювати наявність помилок введення-виводу. Засоби мови С++ передбачають подібний контроль.

Кожен потік, як і всякий об'єкт, у будь-який момент часу характеризується станом, що визначає поведінкові властивості потоку і залежним від попередньої роботи з потоком. ***Стан*** представляється набором трьох груп змінних :Каждый поток, как и всякий объект, в любой момент времени характеризуется состоянием, определяющим поведенческие свойства потока и зависящим от предшествующей работы с потоком. ***Состояние*** представляется набором трех групп переменных:

- *прапорів* (ознак) *стану* потоку і покажчика, що визначає зв'язок потоку з іншим потоком;*флагов* (признаков) *состояния* потока и указателя, определяющего связь потока с другим потоком;
- *прапорів і змінних форматування,* що виконується в потоці;*флагов и переменных форматирования,* выполняемого в потоке;
- *змінною,* що зберігає *поточну позицію* потоку і *прапори режимів* роботи з файлами.*переменной,* хранящей *текущую позицию* потока и *флаги режимов* работы с файлами.

Стан потоку визначається також утримуванним буфера потоку.

Аналіз ***прапорів стану*** дозволяє встановити наявність помилок введення-виводу і можливість подальшої роботи з потоком. Для цього використовуються функції-члени класу *ios.*Анализ ***флагов состояния*** позволяет установить наличие ошибок ввода-вывода и возможность дальнейшей работы с потоком. Для этого используются функции-члены класса *ios.*

При аналізі стану потоку важливим поняттям являється подія ***кінець файлу*** *(End Of File, EOF).* Ця подія виникає при спробі читання з потоку, тоді як поточна позиція потоку дорівнює розміру файлу. Тобто операція введення виконується після того, як вже був лічений останній байт файлу або файл порожній. При цьому встановлюється відповідний прапор стану. Функції введення-виводу можуть повертати значення *EOF* не лише при виникненні події «кінець файлу», але і при виникненні помилок введення-виводу. Цю подію використовують для завершення прочитування даних з файлів.При анализе состояния потока важным понятием является событие ***конец файла*** *(End Of File, EOF).* Это событие возникает при попытке чтения из потока, в то время как текущая позиция потока равна размеру файла. То есть операция ввода выполняется после того, как уже был считан последний байт файла или файл пуст. При этом устанавливается соответствующий флаг состояния. Функции ввода-вывода могут возвращать значение *EOF* не только при возникновении события «конец файла», но и при возникновении ошибок ввода-вывода. Данное событие используют для завершения считывания данных из файлов.

Прапори форматування визначають поведінку потоку при ***форматованому*** введенні-виводі, організовуваному за допомогою операцій >> і << (витягання і вставки). *Форматування* є перетворення послідовності байтів потоку відповідно до встановлених правил. При виводі в потік форматування дозволяє набувати у файлі значення, що виводиться, в певному виді (форматі). При введенні форматування може дозволити прочитувати з потоку послідовність байтів як значення певного типу. *Прапори* і *змінні* форматування задають встановлені в потоці правила форматування, які діють при виконанні операцій >> і <<. Управляти прапорами і встановлювати значення змінних форматування можна за допомогою відповідних методів класу *ios* або за допомогою спеціальних функцій-маніпуляторів*,* частина з яких объявленна у файлі *iomanip.h,* а інші — в *iostream.h.* Відмітимо, що при створенні будь-яких потоків діють єдині правила форматування (встановлені за умовчанням).***Флаги форматирования*** определяют поведение потока при ***форматированном*** вводе-

выводе, организуемом с помощью операций >> и << (извлечения и вставки). *Форматирование* есть преобразование последовательности байтов потока в соответствии с установленными правилами. При выводе в поток форматирование позволяет получать в файле выводимое значение в определенном виде (формате). При вводе форматирование может позволить считывать из потока последовательность байтов как значение определенного типа. *Флаги* и *переменные* форматирования задают установленные в потоке правила форматирования, которые действуют при выполнении операций >> и <<. Управлять флагами и устанавливать значения переменных форматирования можно с помощью соответствующих методов класса *ios* или с помощью специальных функций—*манипуляторов,* часть из которых объявленна в файле *iomanip.h,* а остальные — в *iostream.h.* Отметим, что при создании любых потоков действуют единые правила форматирования (установленные по умолчанию).

Неформатоване уведення-виведення здійснюється за допомогою функцій-членів *get(), put(), read()* і *write()* відповідних класів *istream, ostream, iostream.* При цьому можна виділити два варіанти організації : ***строкоориентированный*** і ***символьний,*** Перший варіант зручний для роботи з текстовими файлами. Символьне уведення-виведення використовується, в першу чергу, для роботи з бінарними файлами.***Неформатированный*** ввод-вывод осуществляется с помощью функций-членов *get(), put(), read()* и *write()* соответствующих классов *istream, ostream, iostream.* При этом можно выделить два варианта организации: ***строкоориентированный*** и ***символьный,*** Первый вариант удобен для работы с текстовыми файлами. Символьный ввод-вывод используется, в первую очередь, для работы с бинарными файлами.

На мал. 12.2 приведені можливі варіанти організації введення-виводу. З усіх можливих варіантів ми розглянемо шість основних:

- консольне форматоване уведення-виведення базових типів;
- консольне форматоване уведення-виведення призначених для користувача типів;
- файлове форматоване уведення-виведення базових типів;
- файлове символьне уведення-виведення;
- файлове строко-ориентированный уведення-виведення;
- форматований обмін даними базових типів з рядком в пам'яті.

Малюнок 12.2 - Варіанти організації введення-виводу

У цих варіантах представлені усі класи потоків стандартної бібліотеки. Інші можливі варіанти можна реалізувати аналогічно, оскільки робота ведеться з однотипними об'єктами — потоками. Відмінності, в першу чергу, проявляються в іменах, використовуваних при створенні потоку і зв'язуванні його з файлом (рядком).

Відмітимо, що до справжнього розділу в усіх прикладах здійснювалося форматоване уведення-виведення через стандартні потоки cin і cout, при якому використовувалися параметри форматування за умовчанням. Тому спочатку розглянемо можливості форматування на прикладі консольного введення-виводу (з використанням стандартних потоків), а потім перейдемо до інших варіантів організації введення-виводу.Отметим, что до настоящего раздела во всех примерах осуществлялся форматированный ввод-вывод через стандартные потоки *cin* и *cout,* при котором использовались параметры форматирования по умолчанию. Поэтому сначала рассмотрим возможности форматирования на примере консольного ввода-вывода (с использованием стандартных потоков), а затем перейдем к другим вариантам организации ввода-вывода.

5.3. Форматоване уведення-виведення базових типів

Для виконання форматованого введення-виведення базових типів використовуються переобтяжені операції >> і << (правого і лівого зрушення), що називаються *операціями* ***витягання*** *(extraction)* і ***вставки*** *(insertion)* відповідно. Перевантаження проведене у файлі *iostream.h* так, щоб полегшити форматоване уведення-виведення значень змінних базових типів. Операції вставки і витягання розпізнають тип правого операнда і відповідно до встановлених правил форматування в потоці перетворять значення цього операнда.Для выполнения форматированного ввода-вывода базовых типов используются перегруженные операции >> и << (правого и левого сдвига), называемые *операциями* ***извлечения*** *(extraction)* и ***вставки*** *(insertion)* соответственно. Перегрузка проведена в файле *iostream.h* таким образом, чтобы облегчить форматированный ввод-вывод значений переменных базовых типов.

Операции вставки и извлечения распознают тип правого операнда и в соответствии с установленными правилами форматирования в потоке преобразуют значение этого операнда.

Приклад 1. Консольне форматоване уведення-виведення базових типів. Наступна програма показує властивості форматування при роботі із стандартними потоками cin і cout.**Пример 1.** Консольный форматированный ввод-вывод базовых типов. Следующая программа показывает свойства форматирования при работе со стандартными потоками *cin* и *cout.*

```
#include <iostream>
int main()
  {
   char з = 't';char c = 't';
   char s[7] = "string ";
   int i = 16;
   double df = 12.3456789;
   ostream *pcout = &cout;
   cout << з <<"<<s<<"<<i<<df <<"<< pcout<<";cout << c
<<'\n'<<s<<'\n'<<i<<df <<'\n'<< pcout<<'\n';
   return 0;return 0;
  }
```

В результаті виконання програми на екран будуть виведені наступні 5 рядків.

```
t
string
16
12.3457
Oxllefl020
```

Останній рядок виводу може відрізнятися, оскільки шістнадцятирична адреса розміщення області пам'яті, виділеної для потоку *cout* на вашому комп'ютері, може бути іншим. Крім того, формат виведення покажчиків за умовчанням залежить від реалізації C++.Последняя строка вывода может отличаться, поскольку шестнадцатеричный адрес размещения области памяти, выделенной для потока *cout* на вашем компьютере, может быть другим. Кроме того, формат вывода указателей по умолчанию зависит от реализации C++.

З наведеного прикладу стає зрозумілим, що потік *cout* є об'єктом класу *ostream.* Відповідно, потік *cin* — це об'єкт класу *istream.*Из приведенного примера становится понятным, что поток *cout* является объектом класса *ostream.* Соответственно, поток *cin* — это объект класса *istream.*

Відносно базових типів і символьних масивів за умовчанням встановлені наступні *стандартні правила форматування*:В отношении базовых типов и символьных массивов по умолчанию установлены следующие *стандартные правила форматирование*:

- символи і рядки символів виводяться в звичному виді;
- числа виводяться в десятковій системі числення;

- знак у позитивних чисел не виводиться;
- у цілих чисел виводяться тільки значущі цифри (незначущі старші нулі відкидаються);
- дійсні числа виводяться зі збереженням до 6 значущих цифр і вказівкою положення десяткової точки, якщо дробова частина не нульова; при отбрасывавании молодших значущих цифр робиться округлення.

Можна управляти форматом введення-виводу за допомогою *прапорів форматування* потоку, які оголошені в класі *ios.* Для установки прапорів форматування використовуються символічні константи, які приведені в таблицю. 12.1. В другому стовпці таблиці приведені правила форматування, які починають діяти при установці відповідних прапорів.Можно управлять форматом ввода-вывода с помощью *флагов форматирования* потока, которые объявлены в классе *ios.* Для установки флагов форматирования используются символические константы, которые приведены в табл. 12.1. Во втором столбце таблицы приведены правила форматирования, которые начинают действовать при установке соответствующих флагов.

Відмітимо, що прапор *boolalpha* відсутній в ранніх версіях С++; прапор *stdio,* навпаки, є в цих версіях, але не увійшов до стандарту С++.Отметим, что флаг *boolalpha* отсутствует в ранних версиях С++; флаг *stdio,* наоборот, имеется в этих версиях, но не вошел в стандарт С++.

Прапори форматування зберігаються в захищеному членові класу *ios* — змінній *x_flags* типу *long.* Встановлювати прапори форматування можна за допомогою функції *flags()* — члена класу *ios.* Ця функція оголошена в двох варіантах:Флаги форматирования хранятся в защищенном члене класса *ios* — переменной *x_flags* типа *long.* Устанавливать флаги форматирования можно с помощью функции *flags()* — члена класса *ios.* Эта функция объявлена в двух вариантах:

long flags();long flags();

long flags(long new_fmtflags);long flags(long new_fmtflags);

Нова сукупність прапорів *new_fmtflags* може бути отримана за допомогою операції | (побітного АБО) і констант, приведених в таблицю. 12.1. Функції повертають колишній набір прапорів.Новая совокупность флагов *new_fmtflags* может быть получена с помощью операции | (побитного ИЛИ) и констант, приведенных в табл. 12.1. Функции возвращают прежний набор флагов.

Приклад 2. Використання прапорів форматування.**Пример 2.** Использование флагов форматирования.

Розглянемо збереження прапорів, встановлених за умовчанням, і установку прапорів для виведення дійсних чисел в науковому форматі і цілих чисел в шістнадцятиричній системі числення.

long oldfmtflags = cout.flags();

cout.flags(oldfmtflags|ios::scientific|ios::hex);

cout << " <<1.2 << "; (dec) 255 = (hex)" << 255;cout << '\n' <<1.2 << "; (dec) 255 = (hex)" << 255;

В результаті виконання цього фрагмента на екрані може бути отриманий наступний рядок:

1.200000e+00; (dec) 255 == (hex) ffe+00; (dec) 255 == (hex) ff

Таблиця 12.1 Констант установки прапорів форматування

Константа	Спосіб форматування
Boolalpha	Виводити дані типу *bool* в символьному виді« наприклад, 0 — *false* і !(0) — *true*
Dec	Використати десяткове представлення
Fixed	Для дійсних чисел використати представлення з фіксованою точкою
Hex	Використати шістнадцятиричне представлення
Internal	Поміщати символ символ-заповнювач (за умовчанням пропуск) після знаку числа або символу-ознаки основи системи числення
Left	Притискати до лівого краю при виводі
Oct	Використати вісімкове представлення
Right	Притискати до правого краю
Scientific	Для дійсних чисел використати «наукове» представлення: мантиса і порядок, розділені символом *е* або *Е*
Showbase	Виводити ознаку системи числення
Showpoint	При виведенні дійсних чисел виводити десяткову точку, навіть якщо дробова частина нульова
Showpos	При виведенні позитивних чисел виводити знак «+»
Skipws	Пропускати пробільні символи при введенні
Stdio	Звільняти стандартні потоки *stdout, stderr* мови З після кожного виводу в потік
Unitbuf	Звільняти буфери (виводити вміст) усіх потоків після кожного включення (висновку) в потік
Uppercase	При виведенні чисел використати букви верхнього регістра

Зберігати значення прапорів, встановлених за умовчанням, необов'язково. Для відновлення початкових установок можна скористатися викликом *flags(O).* За допомогою функції *setf(),* можна встановлювати окремі прапори без використання змінної прапорів форматування.Сохранять значения флагов, установленных по умолчанию, необязательно. Для восстановления начальных установок можно воспользоваться вызовом *flags(O).* С помощью функции *setf(),* можно устанавливать отдельные флаги без использования переменной флагов форматирования.

Приклад 3. Установка окремих прапорів форматування.**Пример 3.** Установка отдельных флагов форматирования.

Наступний фрагмент програми

```
cout.setf(ios::fixed|ios::hex|ios::showbase);
cout <<'\n' <<12.0 << "; 255 = " << 127;
cout.setf(ios::showpos|ios::scientific|ios::oct);
cout <<" <<12.0 << "; 255 = " << 127;cout <<'\n' <<12.0 << "; 255 = " << 127;
```

дозволяє отримати на екрані рядки виду :

12.000000; 255 = Ox7f

+1.200000e+01; 255 = 0177

Функція *setf()* має наступні дві форми:Функция *setf()* имеет следующие две формы:

long setf(long new_fmtflags);

long setf(long new_fmtflags, long reset_mask_fmtflags);

У останньому варіанті другий операнд задає *маску поля прапорів форматів,* які встановлюються заново.В последнем варианте второй операнд задает *маску поля флагов форматов,* которые переустанавливаются.

Функція *setf()* повертає колишній набір прапорів.Функция *setf()* возвращает прежний набор флагов.

Не усі прапори форматування можна встановити одночасно. Інформацію про взаємозв'язок прапорів дають константи, використовувані при утворенні маски для функції *setf().* У класі *ios* оголошена три такі константиНе все флаги форматирования можно установить одновременно. Информацию о взаимосвязи флагов дают константы, используемые при образовании маски для функции *setf().* В классе *ios* объявлено три таких константы

basefield — поле прапорів *dec, oct, hex;basefield —* поле флагов *dec, oct, hex;*

adjustfield — поле прапорів *left, right, internal;adjustfield —* поле флагов *left, right, internal;*

floatfield — поле прапорів *scientific* і *fixed.floatfield —* поле флагов *scientific* и *fixed.*

Ці константи слід використати у разі, коли перед установкою прапора вимагається скинути усі альтернативні прапори. Наприклад, при виконанні наступного фрагмента

cout.setf(ios::dec, ios::basefield);cout.setf(ios::dec, ios::basefield);

cout <<'' <<Ox7B;cout <<'\n' <<Ox7B;

на екрані може бути отриманий наступний рядок:

123

Полі прапорів форматування, що залишилися, можна задати за допомогою порозрядних логічних операцій, наприклад, таким чином:

~ (ios::basefield | ios::adjustfield | ios::floatfield)

Реалізація другої форми функції *setf()* в класі *ios* може бути визначена таким чином:Реализация второй формы функции *setf()* в классе *ios* может быть определена следующим образом:

```
setf(long new_fmtflags, long reset_mask_fmtflags)
  {
   return ( fiags( flags() | ( new_fmtflags & reset_mask_fmtflags )))
  }
```

За допомогою функції ***unsetf()*** скидаються усі прапори, які помічені в параметрі. Функція повертає попереднє значення змінної *x_flags.*С помощью функции ***unsetf()*** сбрасываются все флаги, которые помечены в параметре. Функция возвращает предыдущее значение переменной *x_flags.*

Відмітимо, що не усі прапори діють на вхідні і на вихідні потоки. Облік властивостей прапорів може полегшити введення деяких специфічних значень.

Приклад 4. Введення в шестанадцатеричной системі числення.**Пример 4.** Ввод в шестанадцатеричной системе счисления.

Нехай в змінній *obj_state* типу *unsigned int* кожен біт відповідає певному стану деякого об'єкту. Тоді наступний фрагмент програми припускає, що значення вводитимуться в шестрадцатеричной системі числення.Пусть в переменной *obj_state* типа *unsigned int* каждый бит соответствует определенному состоянию некоторого объекта. Тогда следующий фрагмент программы предполагает, что значения будут вводиться в шестрадцатеричной системе счисления.

```
cin.setf(ios::hex);cin.setf(ios::hex);
cout << "\пВведите стан об'єкту : ";
cin >> obj_state;cin >> obj_state;
```

Тому, якщо *sizeof(int)==l6,* то для установки старшого біта змінною *bitstate* вимагається ввести з клавіатури рядок 8000, а введення FFFF забезпечить установку усіх бітів в одиничний стан.Поэтому, если *sizeof(int)==l6,* то для установки старшего бита переменной *bitstate* требуется ввести с клавиатуры строку 8000, а ввод FFFF обеспечит установку всех битов в единичное состояние.

При виведенні даних у вигляді таблиць часто вимагається визначати фіксовані розміри рядка або символьного представлення числа. В цьому випадку говорять про ***поле виводу*** або про число символів, які мають бути виведені. Якщо поле виводу більше числа символів в рядку, що виводиться, або числі, то незайняті місця заповнюються певними символами. В якості *символу-заповнювача* за умовчанням встановлюється "пропуск". Прапори *left, right, internal* задають правило розміщення числа або рядка в цьому потоці. Крім того, для дійсних чисел часто вимагається управляти і точністю представлення чисел, що виводяться.При выводе данных в виде таблиц часто требуется определять фиксированные размеры строки или символьного представления числа. В этом случае говорят о ***поле вывода*** или о числе символов, которые должны быть выведены. Если поле вывода больше числа символов в выводимой строке или числе, то незанятые места заполняются определенными символами. В качестве *символа-заполнителя* по умолчанию устанавливается "пробел". Флаги *left, right, internal* задают правило размещения числа или строки в этом потоке. Кроме того, для вещественных чисел часто требуется управлять и точностью представления выводимых чисел.

Значення ширини поля виводу, точність представлення і символу заповнювача зберігається в наступних захищених змінних класу *ios :*Значения ширины поля вывода, точности представления и символа заполнителя хранятся в следующих защищенных переменных класса *ios:*

int x_width; // задає мінімальну ширину поля виводуint x_width; // задает минимальную ширину поля вывода

int x_precision; // визначає максимальну кількість значущихint x_precision; // определяет максимальное количество значащих

// цифр дійсного числа

int x_fill; // задає символ-заповнювач поля виводу doint x_fill; // задает символ-заполнитель поля вывода до

// мінімальної ширини

Доступ до змінних форматування забезпечують функції-члени того ж класу, що мають наступні заголовки :

char fill (); // повертає використовуваний символ-заповнювачchar fill (); // возвращает используемый символ-заполнитель

char fill (char cf); // встановлює новий символ заповнювач ichar fill (char cf); // устанавливает новый символ заполнитель и

// повертає колишній символ

int precision (); // повертає використовуване значення змінноїint precision (); // возвращает используемое значение переменной

// x_precisionx_precision

int precision(int p); // встановлює нове значення iint precision(int p); // устанавливает новое значение и

// повертає колишнє значення

int width (); // повертає використовуваний розмір поля вьюодаint width (); // возвращает используемый размер поля вьюода

int width (int w); // встановлює ширину поля виводу і возвра-int width (int w); // устанавливает ширину поля вывода и возвра-

// щает колишнє значення

Зауваження.

Нове значення ширини поля виводу діє тільки на чергову операцію виводу, після якої відновлюється колишнє значення величини *x_mdth* = 0.Новое значение ширины поля вывода действует только на очередную операцию вывода, после которой восстанавливается прежнее значение величины *x_mdth* = 0.

Приклад 5. Управління параметрами форматування.**Пример** 5. Управление параметрами форматирования.

```
cout.flags(O);
cout.width(40);
cout.precision (10);
cout.fill('%');
cout.setf(ios::showpoint|ios::showpos|ios::left);
cout<<17.77777<< " = " << 1.77777el << ">" << 17<<'\n';
cout.setf(ios::internal);
cout.precision(7);
cout.width(40);
cout.fill('*');
cout<<17.77777<< "!= " << 1.77777el << "> " << 17<<";cout<<17.77777<< " != " << 1.77777el << " > " << 17<<'\n';
cout.setf(ios::right);
cout.unsetf(ios::showpos);
cout.precision(4);
```

cout.width(40);
cout. fill *('$')*;cout. fill *('$')*;
cout<<17.77777<< " = " << 1.77777el << "> " << 17<<'\n';

В результаті виконання вказаного фрагмента програми на екрані з'являться три рядки:

+17.77777000%%%%%%%%%%%%%%%%%%%%%%%%%%%%% = +17.77770000 > +17

+******************************17.77777 != +17.77770 > +17

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$17.78 = 17.78 > 17

У наведених прикладах можна виділити два стилі роботи з потоками:

- ***функціональний,*** коли для управління станом потоку використовуються виклики виду***функциональный,*** когда для управления состоянием потока используются вызовы вида

ім'я_потоку.функція();

- ***операциональный,*** який наочно відбиває порядок обміну даними, у виді:***операциональный,*** который наглядно отражает порядок обмена данными, в виде:

ім'я_потоку << виводиться_1_значення << 2_значення <<...<< n_значення;имя_потока << выводимое_1_значение << 2_значение <<.. .<< n_значение;

ім'я_потоку >> вводиться_1_значення >> 2_значення >>...>> n_значення;

Обидва стилі рівноправні, але другою з них представляється наочнішим і привабливішим. І засоби для широкого застосування цього стилю є — це маніпулятори. Розглянемо їх детальніше.

5.4 Маніпулятори

Маніпуляторами називають спеціальні функції, що дозволяють змінювати стан потоку і що використовуються спільно з операціями витягання і вставки в одному операторові введення або виведення даних. Відмінність маніпуляторів від звичайних функцій полягає в тому, що їх імена можна використати в якості правого операнда при виконанні форматованого обміну за допомогою операцій << і >>. В якості найлівішого операнда вираження з маніпуляторами і операторами обміну завжди використовується ім'я потоку.***Манипуляторами*** называют специальные функции, позволяющие изменять состояние потока и использующиеся совместно с операциями извлечения и вставки в одном операторе ввода или вывода данных. Отличие манипуляторов от обычных функций состоит в том, что их имена можно использовать в качестве правого операнда при выполнении форматированного обмена с помощью операций << и >>. В качестве самого левого операнда выражения с манипуляторами и операторами обмена всегда используется имя потока.

Наприклад, використовуючи маніпулятори *hex, oct* і *endl,* можна написати наступний оператор;Например, используя манипуляторы *hex, oct* и *endl,* можно написать следующий оператор;

cout << "(dec) 1023 = (hex)" << hex << тобто << "т.е. (oct) " << от.е.< 1023 << endl;cout << "(dec) 1023 = (hex)" << hex << т.е. << "т.е. (oct) " << от.е.< 1023 << endl;

В результаті виконання на екрані з'явиться рядок

(dec) 1023 = (hex) 3ff = (oct) 1777dec) 1023 = (hex) 3ff = (oct) 1777

Стандартні маніпулятори введення-виводу діляться на дві групи: маніпулятори з параметрами і маніпулятори без параметрів. Маніпулятори з параметрами оголошені у файлі **iomanip.h,** а без параметрів — у файлі ***iostream.h.*** У таблиці. 12.2 приведені стандартні маніпулятори без параметрів.*Стандартные манипуляторы* ввода-вывода делятся на две группы: манипуляторы с параметрами и манипуляторы без параметров. Манипуляторы с параметрами объявлены в файле **iomanip.h,** а без параметров — в файле ***iostream.h.*** В табл. 12.2 приведены стандартные манипуляторы без параметров.

Таблица12.2 Маніпулятори без параметрів

Маніпулятор	Дія в потоці
dec	Перетворення в десяткове представлення
hex	Перетворення в шістнадцятиричне представлення
oct	Перетворення у вісімкове представлення
endl	Вставка символу нового рядка і выгружение буфера
ends	Вставка в потік нульової ознаки кінця рядка
flush	Выгружение буфера вихідного потоку
ws	Витягання і ігнорування пробільних символів *
showbase	Вставка ознаки системи числення *
noshowbase	Вилучення ознаки системи числення *
skipws	Пропуск пробільних символів при введенні *
noskipws	Відміна пропуску пробільних символів при введенні *
uppercase	Використання символів верхнього регістра при виведенні чисел *
Nouppercase	Відміна використання символів верхнього регістра при виведенні чисел *
internal	Вставка заповнювачів між знаком і модулем числа, що виводиться *
left -	Вставка заповнювачів після значення в полі виводу *
right	Вставка заповнювачів перед значенням в полі виводу *
fixed	Використання для дійсних чисел формату *dddd.dd* *
scientific	Використання для дійсних чисел формату *i.dddd e dd* *
boolalpha	Виведення даних типу *bool* в символічному виді *
noboolalfa	Виведення даних типу *bool* як цілих чисел *

Зірочкою відмічені маніпулятори, відсутні в ранніх версіях C++. Відмітимо, що маніпулятори без параметрів управляють прапорами форматування потоку і буфером потоку.

Маніпулятор *endl* зручно використати для негайної видачі повідомлень програми.Манипулятор *endl* удобно использовать для немедленной выдачи сообщений программы.

Наприклад:

....

cout << "Идет процес моделювання. " << endl;cout << "\nИдет процесс моделирования. " << endl;

Висновок з таким використання маніпулятора *endl* позбавляє від необхідності очікування заповнення буфера до моменту закінчення процесу моделювання.Вывод с таким использование манипулятора *endl* избавляет от необходимости ожидания заполнения буфера до момента окончания процесса моделирования.

Щоб уникнути подібних ситуацій при висновку повідомлень про помилки, потік cerr *не* буферизирован. Тому повідомлення про помилки доцільно направляти в потік сегг. *При цьому* ім'ям потоку відзначається місце обробки помилок в програмі.Чтобы избежать подобных ситуаций при выводе сообщений об ошибках, поток *cerr не* буферизирован. Поэтому сообщения об ошибках целесообразно направлять в поток *сегг.* При этом именем потока отмечается место обработки ошибок в программе.

Приклад 1. Висновок повідомлень про помилки в потік cerr.**Пример 1.** Вывод сообщений об ошибках в поток *cerr.*

if (SizeFile > MaxSizeFile)if (SizeFile > MaxSizeFile)

(

cerr << "\u1087?Размер файлу результатів занадто великий".;cerr << "\пРазмер файла результатов слишком велик.";

... // оператори обробки помилки

)

cout << "Сохраняются результати моделювання". <<endl;cout << "\nСохраняются результаты моделирования." <<endl;

Якщо при виводі не вимагається починати новий рядок і треба забезпечити негайне звільнення буфера, то слід використати маніпулятор *flush*. НаприкладЕсли при выводе не требуется начинать новую строку и нужно обеспечить немедленное освобождение буфера, то следует использовать манипулятор *flush*. Например,

cout << "\u1087?Введите Tm =**" <<** flush;cout << "\пВведите Tm =**" <<** flush;

cin >> Tm;cin >> Tm;

чи

clog << "Завершение роботи. **" <<** flush;clog << "\nЗавершение работы. **"** **<<** flush;

Для управління точністю, шириною поля виводу і іншими параметрами форматування *використовуються маніпулятори з параметрами,* приведені в таблицю 12.3.Для управления точностью, шириной поля вывода и другими параметрами форматирования *используются манипуляторы с параметрами,* приведенные в табл 12.3.

Відмітимо, що маніпулятори забезпечують усі можливості методів класу *ios* для управління форматуванням.Отметим, что манипуляторы обеспечивают все возможности методов класса *ios* для управления форматированием.

Приклад 2. Використання маніпуляторів з параметрами.**Пример 2.** Использование манипуляторов с параметрами.

```
#include <iostream.h>
#include <iomanip.h>
int main()
  {
    double darray[]=(6.7891, - 0.0089, 5.6789, - 0.0056, 4.5678);
    char col_name[] ="значення";char col_name[] ="значение";
cout << setw(sizeof(col_name))<<col_name<<endl;
cout << setprecision(3) << setiosflags(ios::fixed | ios: rshowpoint);
cout << setfil('x')    << setiosflags(ios::showpos | ios::left);
  for (int i=0;i<sizeof(darray)/sizeof(darray[0]);i++)
    cout << setw(sizeof(col_name))<<darray[i]<<endl;
  return 0;
  }
```

В результаті виконання програми на екрані з'явиться стовпець закруглених значень елементів масиву із заголовком.

Таблиця 12.3 Маніпуляторів з параметрами

Маніпулятор і тип параметра	Дія в потоці
Resetios flags (long fflags)	Скидання прапорів форматування
Setios flags (long fflags)	Установка прапорів форматування
Setbase (int b)	Установка системи числення із заданою основою
Setfill (int cf)	Установка символу-заповнювача
Setpresision (int p)	Установка точності виведення дійсних чисел
setw (int w)	Установка ширини поля для чергової операції виводу

5.5 Прапори стану потоку

При введенні даних з клавіатури дуже легко помилитися. Оскільки оператор << призначений для прочитування даних очікуваного типу в очікуваному форматі, то чергова операція введення може виконатися як нульова (безрезультатно). Щоб контролювати подібні ситуації, клас ios має захищену змінну *state* типу *int,* в якій зберігаються *прапори стану потоку.*При вводе данных с клавиатуры очень легко ошибиться. Поскольку оператор << предназначен для считывания данных ожидаемого типа в ожидаемом формате, то очередная операция ввода может выполниться как нулевая (безрезультатно). Чтобы контролировать подобные ситуации, класс ios имеет защищенную переменную *state* типа *int,* в которой хранятся *флаги состояния потока.*

Для роботи із станом потоку використовуються наступні чотири *константи,* що визначають *прапори стану :*Для работы с состоянием потока используются следующие четыре *константы,* определяющие *флаги состояния:*

- *badbit* — потік зіпсований;*badbit* — поток испорчен;
- *eofbit* — досягнутий кінець файлу;*eofbit* — достигнут конец файла;
- *failbit* — наступна операція не виконається;*failbit* — следующая операция не выполнится;
- *goodbit* — прапор, на відміну від інших, встановлений в нуль, якщо потік не зіпсований, не досягнутий кінець файлу і наступна операція може виконається, тобто потік «хороший» *(good).goodbit* — флаг, в отличие от остальных, установлен в ноль, если поток не испорчен, не достигнут конец файла и следующая операция может выполнится, т.е. поток «хороший» *(good)*.

Для аналізу стану потоку можна використати методи *good(), eof(), fail(), bad()* і *rdstate()* без параметрів і дві переобтяжені унарні операції ( ) і !( ).Для анализа состояния потока можно использовать методы *good(), eof(), fail(), bad()* и *rdstate()* без параметров и две перегруженные унарные операции ( ) и !( ).

Метод *good()* і операція ( ) повертають не нуль, якщо встановлений прапор *goodbit*. Метод *fail()* і операція 1( ) повертають не нуль, якщо встановлений прапор *failbit.* Метод *rdstate()* повертає значення змінної *state.*Метод *good()* и операция ( ) возвращают не ноль, если установлен флаг *goodbit*. Метод *fail()* и операция 1( ) возвращают не ноль, если установлен флаг *failbit.* Метод *rdstate()* возвращает значение переменной *state.*

Приклад. Аналіз стану потоку.**Пример.** Анализ состояния потока.

```
cout << "Введіть d = " <<flush;cout << "Введите d = " <<flush;
cin >> d;
if (cin.fail ())(
cerr << "0 шибка введення d";cerr << "0шибка ввода d";
exit(1);
)
```

Аналіз стану потоку за допомогою операції !() можна виконати таким чином:

```
if (! (cin >> d)){
cerr << "0 шибка введення";
exit(1);
}
```

Прапори стану можна встановлювати заново за допомогою методу *c1eaг() або* другої *форми* методу *rdstate(),* вказавши в якості фактичного параметра прапори, що скидаються або встановлювані, відповідно. Тут є аналогія з установкою і скиданням прапорів форматування. Виклик *c1 ear()* без параметрів еквівалентний виклику *rdstate(ios::goodbit).*Флаги состояния можно переустанавливать с помощью метода *c1eaг() или* второй *формы* метода *rdstate(),* указав в качестве фактического параметра сбрасываемые или устанавливаемые флаги соответственно. Здесь есть аналогия с установкой и сбросом флагов

форматирования. Вызов *c1ear()* без параметров эквивалентен вызову *rdstate(ios::goodbit).*

5.6 Зв'язування потоків

Зв'язування потоків є встановленням зв'язку між двома потоками за допомогою методу *tie()* класу *ios,* що виконується з метою *синхронізації* операцій введення і виводу для пов'язаних потоків. Пояснимо на прикладі, навіщо потрібно зв'язування двох потоків. Припустимо, виконується наступний фрагмент:***Связывание потоков*** представляет собой установление связи между двумя потоками с помощью метода *tie()* класса *ios,* выполняемое с *целью синхронизации* операций ввода и вывода для связанных потоков. Поясним на примере, зачем требуется связывание двух потоков. Допустим, выполняется следующий фрагмент:

cout << "Введіть будь-який символ :"
cin >> з;

Наявність буфера у потоку *cout* може привести до того, що повідомлення на екрані з'явиться пізніше, ніж буде виконана операція прочитування. Тому слід синхронізувати роботу потоків таким чином, що коли потрібний новий символ з одного потоку, то в другому потоці звільнявся б буфер. Рішення цієї задачі за допомогою маніпулятора endl явно занадто грубе. Використання маніпулятора flush *або методу flush()* доцільно тільки при одиничних синхронізаціях. Метод tie*() дозволяє* встановити подібну синхронізацію між потоками на необхідному інтервалі виконання таким чином:Наличие буфера у потока *cout* может привести к тому, что сообщение на экране появится позже, чем будет выполнена операция считывания. Поэтому следует синхронизировать работу потоков таким образом, что когда нужен новый символ из одного потока, то во втором потоке освобождался бы буфер. Решение этой задачи с помощью манипулятора endl явно слишком грубое. Использование манипулятора *flush или метода flush()* целесообразно только при единичных синхронизациях. Метод *tie()* позволяет установить подобную синхронизацию между потоками на требуемом интервале выполнения следующим образом:

cin.tie(&cout); //встановлення синхронизующей зв'язкуcin.tie(&cout); //установление синхронизующей связи
cout << "Введіть будь-який символ :"
cin >> з;
cin.tie (0); // розривши зв'язкиcin.tie (0); // разрыв связи

Список параметрів методу *tie()* може містити покажчик на потік, нульове значення або бути порожнім. Повертане значення є покажчик на потік, з яким пов'язаний цей потік. У кожен момент часу будь-який потік може бути пов'язаний тільки з одним потоком.Список параметров метода *tie()* может содержать указатель на поток, нулевое значение или быть пустым. Возвращаемое значение есть указатель на поток, с которым связан данный поток. В каждый момент времени любой поток может быть связан только с одним потоком.

Контрольні питання і завдання

1. Дайте визначення поняття «потік».
2. Чим відрізняються текстові і бінарні файли?
3. Назвіть основні класи потоків.
4. Намалюйте дерево спадкоємства властивостей для стандартних класів потоків.
5. Об'єктами яких класів є потоки *cin* і *cout?*Объектами каких классов являются потоки *cin* и *cout?*
6. У яких цілях використовуються потоки *cerr, clog?*В каких целях используются потоки *cerr, clog?*
7. Опишіть основні етапи роботи з файлами через потоки.
8. Що таке форматоване уведення-виведення?
9. Чим відрізняється форматоване уведення-виведення від строко-ориентированного і символьного введення-виводу?
10. Що загального між форматованим і строко-ориентированным введенням-виводом?
11. Які засоби використовуються для управлінням форматованим введенням-виводом?
12. Які є прапори і змінні форматування?
13. Назвіть прапори, які впливають тільки на потоки виводу.
14. Які прапори визначають тільки формат виводу?
15. У яких файлах оголошені засоби форматування?
16. Що таке маніпулятори і як вони використовуються?
17. У яких цілях використовуються маніпулятори *flush* і *endl?*В каких целях используются манипуляторы *flush* и *endl?*
18. Назвіть маніпулятори для управління змінними форматування.
19. Що визначають прапори стану потоку?
20. Навіщо і як аналізують прапори стану потоку?
21. У яких цілях можна використати метод *tie()?*В каких целях можно использовать метод *tie()?*
22. Перевантажте операції << і >> для форматованого введення об'єктів класу «матриця дійсних чисел».
23. У якому файлі оголошені класи файлових потоків?
24. Назвіть основні методи файлових потоків.
25. Коли вимагається явно розривати зв'язок потоку з файлом?

6 ДОДАТКОВІ МОЖЛИВОСТІ ВВЕДЕННЯ-ВИВОДУ

У справжньому розділі розглядається організація форматованого введення-виведення призначених для користувача типів« файлового введення-виведення призначених для користувача типів, файлового введення виводу, обміну з рядком в оперативній пам'яті і засобу покрокового введення-виведення мови С.

6.1 Форматоване уведення-виведення призначених для користувача типів

У разі створення нового призначеного для користувача типу даних для зручності роботи з ним доцільно перевантажити операції « і » (вставки і витягання). Розглянемо варіант перевантаження на наступному прикладі.

Приклад 1. Перевантаження операцій « і ». Розглянемо виконання перевантаження операцій « і » (вставки і витягання) для класу «точка тривимірного простору *chafxintxdouble^*. Відповідний фрагмент програми має наступний вигляд:Рассмотрим выполнение перегрузки операций « и » (вставки и извлечения) для класса «точка трехмерного пространства *chafxintxdouble^*. Соответствующий фрагмент программы имеет следующий вид:

```
class cid_point {class cid_point {
char CJP;char CJP;
int i_p;
double dJ?;
public:
```

Література

1. *Бабэ Б.* Просто и ясно о Borland С++: пер. с англ. — СПб.: Питер, 1997. — 464 с.
2. *Вод Т.* Объектно-ориентированное программирование в действии: пер. с англ. СПб.: Питер, 1997. - 464 с.
3. *Ирэ* Я. Объектно-ориентированное программирование с использование С++: пер. с англ. К.: НИПФ ДиаСофтЛтд, 1995. - 480 с.
4. *Подбельский В.В.* Язык С++. — М.: Финансы и статистика, 1966. — 558 с.
5. Программирование. Учебник под ред. Свердлика А.Н., МО СССР, 1992. — 608 с.
6. *Сван* Г. Программирование для Windows в Borland С++: пер. с англ. — М.: БИНОМ. - 480 с.
7. *Шамис ВА.* Borland С++ Builder. Программирование на С++ без проблем. — М.: Нолидж, 1997. - 266 с.
8. *Шшдт* Г. Теория и практика С++: пер. с англ. — СПб.: BHV — Санкт-Петербург, 1996.-416 с.
9. *Шилдт Г.* Самоучитель С++, 3-е издание: пер. с англ. — СПб.: BHV — Санкт-Петербург, 1998. - 688 с.