**Лабораторна робота №13. Розробка програм із використанням віртуальних функцій**

**Мета:** Засвоєння навичок розробки програм з використанням віртуальних функцій

**Порядок виконання роботи**

1. Ознайомитися з теоретичними основами розробки віртуальних функцій.
2. Продовжити працювати над програмою лабораторної роботи №10.
   1. Створити через просте спадкування похідний до *Person к*лас *Teacher*, в якому створити:
3. приватні члени-дані : предмет, кафедра;
4. необхідні методи;
5. перевантажити методи базового класу: введення даних в інтерактивному режимі та метод виведення даних на консоль (використати віртуальні функції);
6. дружні функції 1) виведення даних, попередньо введених в інтерактивному режимі, до файлу; 2) введення даних з файлу та заповнення масиву об’єктів.
   1. В головній програмі (функції main()):
7. включити виклик нових методів та функцій до відповідних позицій меню (створити позиції для *Teacher*, якщо їх не було);
8. створити екземпляр класу *Teacher* та заповнитийого даними та вивести на консоль.
9. Виведення до файлу та введення даних з файлу щодо студентів виконувати за відпрацьованими алгоритмами на ЛР№№3-4, а саме: інформація записується до файлу, де дані атрибутів відділяємо знаком ";", а кінець позначаємо знаком "#", після якого записуємо поточну дату у вигляді РРММДД (де РР – дві останні цифри року, ММ – номер місяця, ДД – номер дня). Відповідно, введені дані потім потрібно буде аналізувати та "розбирати" на складові, щоб присвоїти значення відповідним членам класу. На цьому етапі ім’я файлу надаєте літералом у форматі FТ\_**<Прізвищеанглійською>.tхt.**
10. Для виведення даних класу *Teacher* створити позицію меню "Formation of file objects *Teacher* " (Формування файлу об’єктів *Teacher*), а для введення – "Read the *Teacher* object file" (Читання файлу об’єктів *Teacher*).
11. Встановити аналіз виклику нових функцій, зокрема, виконувати виведення в разі наявності заповнених об’єктів класу *Teacher*.
12. Протестувати програму щодо коректності роботи методів введення даних в інтерактивному режимі та виведення даних на консоль для всіх класів, створивши нові записи з прізвищем Test, які дозаписати до існуючих файлів. Всі файли з даними надіслати разом з програмою.
13. В першому рядку програми та заголовкового файлу повинні бути записаними в коментарі номер групи та прізвище, а також номер ЛР (через кому до попередньої).
14. Результати надсилати на електронну адресу викладача

[**t.i.lumpova@gmail.com**](mailto:t.i.lumpova@gmail.com)у вигляді заголовкового файлу та cpp-файлу з іменем у форматі

Для заголовкового файлу

**<Прізвище англійською>.h**

Наприклад, Ivanov.h

Для cpp-файлу:

**<Номер групи> <Номер лабораторної><Прізвище англійською>.cpp**

Наприклад, 31-01Ivanov.cpp.

Тему в заголовку листа записати

**ООП<Номер групи>-ЛР <Номер лабораторної>-<Прізвищеанглійською>**

**Строк відсилки ЛР ІПЗ-31 28.11.2020**

**ІПЗ-32 23.11.2020**

Всі запитання, що виникнуть, надсилайте на електронну адресу викладача, тему в заголовку листа записати

**ООП<Номер групи>-Запитання-<Прізвище англійською>**.

**Контрольні запитання**

1. Що таке віртуальні функції?
2. Що таке поліморфізм?
3. Для чого призначені віртуальні функції?
4. У чому полягає раннє зв’язування?
5. Наведіть приклад раннього зв’язування.
6. У чому полягає пізнє зв’язування?
7. Наведіть приклад пізнього зв’язування.
8. Наведіть приклад віртуальної функції.
9. Чим відрізняються віртуальні функції від перевантаження функцій?
10. Які операції можна використовувати з об’єктами базового та похідного класу?
11. Особливості роботи з масивами об’єктів базового та похідного класів.
12. Дайте визначення поняття поліморфізму.
13. Наведіть приклади прояву поліморфізму в мові С++.
14. Поясніти поняття статичного і динамічного поліморфізму.
15. У чому значення перевантаження функцій, на чому заснований механізм перевантаження функцій?

**16.** Приведіть формат запису операторної функції.

**17.** Вкажіть повну і скорочену форми виклику операторної функцій.

**18.** Вкажіть обмеження, що накладаються на перевантаження стандартних операцій.

**19.** Дайте визначення і вкажіть достоїнства і недоліки раннього зв’язування.

**20.** Дайте визначення і вкажіть переваги і недоліки пізнього зв’язування.

**21.** У чому значення віртуальних функцій?

**22.** Як здійснюється виклик віртуальних функцій?

23. Що на Вашу думку визначає цей запис int (\*fcnPtr)(int) = boo;?

Зауваження до №23

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | // Прототипы функций  int boo();  double doo();  int moo(int a);  // Присваивание значений указателям на функции  int (\*fcnPtr1)() = boo; // ок  int (\*fcnPtr2)() = doo; // не ок: типы возврата не совпадают!  double (\*fcnPtr4)() = doo; // ок  fcnPtr1 = moo; // не ок: fcnPtr1 не имеет параметров, но moo() имеет  int (\*fcnPtr3)(int) = moo; // ок |

C++ **неявно конвертує** функцію в покажчик на функцію, якщо це необхідно (тому не потрібно використовувати оператор & для отримання адреса функції). Але, C++ не буде неявно конвертувати покажчик на функцію в покажчик типа void або навпаки.

**Теоретичні відомості.**

Віртуальна функція (virtual function) є компонентною функцією базового класу, яка перевизначається в похідному класі. Використовування віртуальних функцій, на відміну від перевантаження функцій, забезпечує динамічний поліморфізм, реалізовуваний на етапі виконання програми.

При оголошенні віртуальної функції в базовому класі перед її ім'ям указується 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 указується.

Віртуальну функцію можна викликати як будь-яку іншу компонентну функцію. Проте для підтримки динамічного поліморфізму віртуальні функції викликають через покажчик базового класу, використовуваний як посилання на об'єкт похідного класу. Якщо об'єкт похідного класу, що адресується таким чином, містить віртуальну функцію і віртуальна функція 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.

Якщо є декілька похідних класів від базового класу, що містить віртуальну функцію, 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 відмінності від механізму перевантаження функцій. По-перше, **віртуальна функція, що перевизначається, повинна мати ті ж тип, число параметрів і тип значення, що повертається, тоді як перенавантажувана функція повинна мати відмінності в типі і числі параметрів**. По-друге, віртуальна функція повинна бути компонентом класу, що не відноситься до перенавантажуваних функцій.

Щоб прояснити значення використовування віртуальних функцій, наведемо нижче приклад, що ілюструє різницю між звичними і віртуальними функціями наведений нижче.

**#include <iostream>**

**#include <Windows.h>**

**using namespace std;**

**class parent{**

**//** оголошення базового класу

**public:int k;**

**parent(int a){ //** конструктор класу

**k=a;};**

**virtual void fv(){ //** віртуальна функція

**cout <<"call fv() of base class: cout " <<k << endl; };**

**void f(){ //** звичайна функція

**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() //** звична функція

**{**

**cout <<"call f() of derived class childl: ";**

**cout <<(k-1) <<endl;**

**}**

**};**

**int main () {system("color F0");**

**parent \*p;//** покажчик на базовий клас

**parent ez(5);//** оголошення об'єкту базового класу

**childl chezl(15); //** оголошення об'єкту похідного класу

**p=&ez;//** покажчик посилається на об'єкт базового класу

**p->fv(); //** виклик функції fv() базового класу

**p->f(); //** виклик функції f() базового класу

**p=&chezl;//** покажчик посилається на об'єкт похідного класу

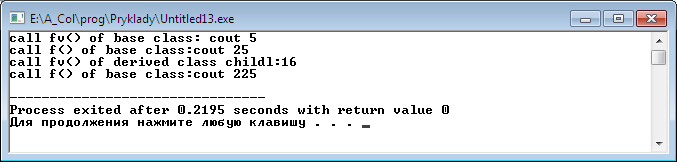
**p->fv(); //** виклик віртуальної функції fv() похідного класу

**p->f(); //** виклик звичної функції f() похідного класу

**return 0;**

**}**

При виконанні наведеної програми на екран будуть видані наступні результати:



Одержані результати показують, що при зверненні до віртуальної функції fv() похідного класу відбувається виклик функції похідного класу. Звернення до звичної функції f() похідного класу приводить до виклику однойменної функції базового класу, а не похідного.

Розглянуті вище статичний і динамічний варіанти поліморфізму (перевантаження функцій і використовування віртуальних функції відповідно) співвідносять з двома наступними поняттями: раннє зв’язування і пізнє зв’язування.

Раннє зв’язування торкається подій етапу компіляції програми, таких як: настройка при виклику звичних функцій, перенавантажених функцій, невіртуальних компонентних функцій і дружніх функцій. При виклику перерахованих функцій вся необхідна адресна інформація відома при компіляції. Перевагою раннього зв’язування є висока швидкодія одержуваних здійснимих програм. Недоліком раннього зв’язування є зниження гнучкості програм.

Пізнє зв’язування відноситься до 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 функції через покажчик базового класу при виконанні програми визначається тип засланого об'єкту і вибирається версія віртуальної функції для виклику. Перевагою пізнього зв’язування є висока гнучкість виконуваної програми, можливість реакції на події. Недоліком є відносно низька швидкодія програми.

**Специфікатори override та final**

Ці специфікатори не є ключовим, вони дозволяють вирішувати деякі специфічні проблеми. Специфікатор **оverride** є контекстно-залежним, він повинен стояти після визначення функції-члена, інакше він не розглядається як специфікатор. Він застосовується для запобігання випадковому спадкуванню, допомагає уникнути помилок.

Приклад 1.

class BaseClass

{

virtual void funcA();

virtual void funcB() const;

virtual void funcC(int = 0);

void funcD();

};

class DerivedClass: public BaseClass

{

virtual void funcA(); // ok, works as intended

virtual void funcB(); // DerivedClass::funcB() is non-const, so it does not

// override BaseClass::funcB() const and it is a new

// member function

virtual void funcC(double = 0.0); // DerivedClass::funcC(double) has a

// different parameter type than

// BaseClass::funcC(int), so

// DerivedClass::funcC(double) is a new

// member function

};

При використанні **оverride** компілятор видає помилки замість створення нових функцій.

class BaseClass

{

virtual void funcA();

virtual void funcB() const;

virtual void funcC(int = 0);

void funcD();

};

class DerivedClass: public BaseClass

{

virtual void funcA() override; // ok

virtual void funcB() override;

*// compiler error: DerivedClass::funcB() does not override*

*// BaseClass::funcB()const*

virtual void funcC( double = 0.0 ) override;

*// compiler error: DerivedClass::funcC(double) does not*

*// override BaseClass::funcC(int)*

void funcD() override;

*// compiler error: DerivedClass::funcD() does not override the*

*// non-virtual BaseClass::funcD()*

};

Специфікатор **final** використовується тоді, коли потрібно заборонити перевизначити віртуальну функцію або спадкувати клас.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | Приклад 3.  class A  {  public:  virtual const char\* getName() { return "A"; }  };  class B : public A  {  public:  // final в конце означает, что метод переопределить уже нельзя  virtual const char\* getName() override final { return "B"; } // всё хорошо, //переопределение A::getName()  };  class C : public B  {  public:  virtual const char\* getName() override { return "C"; } // ошибка компиляции: //переопределение метода B::getName(), который является final  }; |

Приклад 4.

class A

{

public:

virtual const char\* getName() { return "A"; }

};

class B final : public A // обратите внимание на модификатор final здесь

{

public:

virtual const char\* getName() override { return "B"; }

};

class C : public B // ошибка компиляции: нельзя наследовать класс final

{

public:

virtual const char\* getName() override { return "C"; }

};