Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп’ютерної інженерії

Кафедра ПЗ

Лабораторна робота №5

з дисципліни «Операційні системи»

з теми « класи і об’єкти в С++ »

Виконали: ст. 1ПІ-21Б Миронюк О.В.

Гиренко В.В.

Коцюбняк В.А.

Максименко О.В.

Перевірив: Рейда О. М.

Вінниця – 2023

Мета роботи: отримати практичні навички реалізацій класів в С++.

Теоретичні відомості

Клас

Клас – фундаментальне поняття С++, він лежить в основі багатьох властивостей С++. Клас надає механізм для створення об’єктів. В класі відображені найважливіші концепції об’єктно-орієнтованого програмування: інкапсуляція, наслідування, поліморфізм.

З точки зору синтаксису, клас в С++ – це структурований тип, сформований на основі вже існуючих типів.

В цьому сенсі клас є розширенням поняття структури. В найпростішому випадку клас можна визначити за допомогою конструкції:

тип_класу ім’я_класу{список_членів_класу};

де тип_класу – одне з службових слів **class, struct, union**;

ім’я_класу – ідентифікатор;

список_членів_класу – визначення та опис типізованих даних та функцій, що належать класу.

Функції – це методи класу, що визначають операції над об’єктом.

Дані – це поля об’єкту, що формують його структуру. Значення полів визначають стан об’єкту.

До визначених об’єктів входять дані, які відповідають членам-даним класу. Функції-члени класу дозволяють обробляти дані конкретних об’єктів класу. Звертатися до даних об’єкта та викликати функції для

об’єкта можна двома способами. Перший за допомогою «кваліфікованих імен»:

ім'я_об’єкта.ім’я_даного

ім’я_об’єкта.ім’я_функції

Другий спосіб доступу використовує вказівник на об’єкт
вказівник_на_об’єкт–>ім’я_компонента

Доступність компонентів класу

В розглянутих вище прикладах класів компоненти класів є загальнодоступними. В будь-якому місті програми, де “видно” визначення класу, можна отримати доступ до компонентів об’єкта класу. Таким чином не виконується основний принцип абстракції даних – інкапсуляція (приховування) даних всередині об’єкта. Для зміни видимості компонентів у визначенні класу можна використовувати специфікатори доступу: **public, private, protected.**

Загальнодоступні (public) компоненти доступні в будь-якому місці програми. Вони можуть використовуватися будь-якою функцією як всередині даного класу, так і поза його межами. Доступ ззовні реалізується через ім’я об’єкта:

ім'я_об’єкта.ім’я_члена_класу

посилання_на_об’єкт.ім’я_члена_класу

вказівник_на_об’єкт–>ім’я_члена_класу

Власні (private) компоненти локалізовані в класі і не доступні ззовні. Вони можуть використовуватися функціями-членами даного класу і функціями-“друзями” того класу, в якому вони описані.

Захищені (protected) компоненті доступні всередині класу та в похідних класах.

Змінити статус доступу до компонентів класу можна і за допомогою використання при визначенні класу ключового слова class. В цьому випадку всі компоненти класу за замовчуванням є власними.

Конструктор

Недоліком розглянутих вище класів є відсутність автоматичної ініціалізації створюваних об’єктів. Для кожного створюваного об’єкту необхідно було викликати функцію типу **set** (як для класу complex) або явно присвоювати значення даним об’єкта. Однак для ініціалізації об’єктів класу в його визначення можна явно включити спеціальну компонентну функцію, яка називається конструктором. Формат визначення конструктора наступний:

ім'я_класу(список_форм_параметрів{оператори_тіла_конструктора}

Ім’я цієї компонентної функції за правилами мови C++ повинно співпадати з ім’ям класу. Така функція автоматично викликається при визначенні або розміщенні в пам’яті за допомогою оператора **new** кожного об’єкта класу.

Конструктор виділяє пам’ять для об’єкта та ініціалізує дані-члени класу.

Конструктор має ряд особливостей:

- для конструктора не визначається тип значення, що повертається, навіть тип void недопустимий;

- вказівник на конструктор не може бути визначений, відповідно не можна отримати адресу конструктора;

- конструктори не наслідуються;

- конструктори не можуть бути описані з ключовими словами virtual, static, const, mutuable, valatile.

Конструктор завжди існує для будь-якого класу, причому, якщо він не був визначений явно, створюється автоматично. За замовчуванням створюється конструктор без параметрів та конструктор копіювання. Якщо конструктор описаний явно, то конструктор за замовчуванням не створюється. За замовчуванням конструктори створюються загальнодоступними (public).

Параметром конструктора не може бути його власний клас, але може бути посилання на нього (T&). Без явної вказівки програміста конструктор завжди автоматично викликається при визначенні (створенні) об’єкта. В такому випадку викликається конструктор без параметрів. Для явного виклику конструктора використовуються дві форми:

ім'я_класу ім’я_об’єкта(фактичні_параметри);
ім’я_класу(фактичні_параметри);

Деструктор

Динамічне виділення пам’яті для об’єкта створює необхідність вивільнення цієї пам’яті після знищення об’єкта. Наприклад, якщо об’єкт формується як локальний всередині блоку, то доцільно, щоб при виході з блоку, коли об’єкт вже перестає існувати, було повернуто виділену для нього пам’ять. Бажано, щоб вивільнення пам’яті відбувалося автоматично. Таку можливість забезпечує спеціальний компонент класу – **деструктор** класу. Його формат:

~ім’я_класу(){оператори_тіла_деструктора}

Ім’я деструктора співпадає з іменем класу, але йому передує символ “~” (тильда)

. Деструктор не має параметрів та значення, що повертається. Виклик деструктора відбувається неявно (автоматично), як тільки об’єкт класу знищується.

Вказівники на компоненти-функції

Можна визначити вказівник на компоненти-функції:

тип_поверт_значення(ім’я_класу::*ім’я_вказівника_на_функцію) (специф_параметрів_функції);

1 Приклад класу через властивості (property):

1. Приклад визначення класу:

```
#include <iostream>
#include "SERVANT.h"

using namespace std;

   //Properties
   PROPERTY<char*> Name;
   PROPERTY<int> Age;
   PROPERTY<int> Experience;

   //Constructors
   SERVANT::SERVANT()
   {
      cout << "Empty constructor called " << endl;
```

```
};
    SERVANT::SERVANT(char* name, int age, int experience) {
   Name = name;
   Age = age;
   Experience = experience;
   cout << "Constructor called by object " << Name() << endl;
};
SERVANT::SERVANT(SERVANT& s) {
   Name = s.Name();
   Age = s.Age();
   Experience = s.Experience();
   cout << "Constructor called by object " << Name() << endl;
};
SERVANT::~SERVANT() {
   cout << "Destructor called by object " << this->Name() << endl;
};

//Functions
void SERVANT::skip_years()
{
   int years_amount;

   cout << "Enter amount of years to skip: ";
   cin >> years_amount;

   this->Age = this->Age() + years_amount;
   this->Experience = this->Experience() + years_amount;

   cout << "Years Passed!\n\n";
```

```
}

SERVANT* SERVANT::create_new_servant() {
    char* name = new char[100];
    int age;
    int experience;

    cout << "Enter servant's data: \n";
    cout << "Name:";
    cin >> name;
    cout << "Age:";
    cin >> age;
    cout << "Experience:";
    cin >> experience;
    cout << endl << endl;

    return new SERVANT(name, age, experience);
}

SERVANT* SERVANT::make_one_year_younger(SERVANT & s) {
    SERVANT* temp_servant = new SERVANT(s);
    temp_servant->Age = s.Age() - 1;
    cout << "Made servant younger!" << endl << endl;

    return temp_servant;
}

void SERVANT::print_servant(SERVANT* s) {
```

```
    cout << "Servant: \nName: " << s->Name() << ",\nAge: " << s->Age() << ",\nExperience: " << s->Experience() << ";\n\n";
  }
```

2. Приклад реалізації конструктора з виведенням повідомлення:

```
SERVANT::SERVANT(char* name, int age, int experience) {
    Name = name;
    Age = age;
    Experience = experience;
    cout << "Constructor called by object " << Name() << endl;
  };
```

3. Варто передбачити в програмі всі можливі способи виклику конструктора копіювання.

а) при використанні об’єкта для ініціалізації іншого об’єкта

```
SERVANT::SERVANT(SERVANT& s) {
    Name = s.Name();
    Age = s.Age();
    Experience = s.Experience();
    cout << "Constructor called by object " << Name() << endl;};
```

б) коли об’єкт передається функції за значенням

```
  void SERVANT::print_servant(SERVANT* s) {
    cout << "Servant: \nName: " << s->Name() << ",\nAge: " << s->Age() << ",\nExperience: " << s->Experience() << ";\n\n";
  }
```

в) при використанні тимчасового об’єкта як значення, що повертає функція

```
SERVANT* SERVANT::make_one_year_younger(SERVANT & s) {
```

```
    SERVANT* temp_servant = new SERVANT(s);
    temp_servant->Age = s.Age() - 1;
    cout << "Made servant younger!" << endl << endl;

    return temp_servant;
}
```

4. У програмі необхідно передбачити розміщення об’єктів як у статичній, так і в динамічній пам’яті, а також створення масивів об’єктів:

а) масив студентів розміщується в статичній пам’яті

```
SERVANT department[2];
char name[4];
strcpy(name, "Bob");
department[0] = *(new SERVANT(name, 33, 12));

char name1[4];
strcpy(name1, "Qwe");
department[1] = *(new SERVANT(name1, 28, 6));

(*print)(&department[0]);
(*print)(&department[1]);
```

б) масив студентів розміщується в динамічній пам’яті

```
SERVANT* SERVANT::create_new_servant() {
    char* name = new char[100];
```

int age;

int experience;

5. Приклад використання вказівника на компонентну функцію

SERVANT* SERVANT::create_new_servant() {

char* name = new char[100];

Програма використовує 4 файли:

- SERVANT.h -файл заголовку з визначенням класу;

- main.cpp - файл демонстративної програми

- PROPERTY.cpp -файл з реалізацією класу властивостей;

- SERVANT.cpp -. файл з реалізацією класу;

Завдання: написати програму, в якій створюються та знищуються об’єкти, визначеного користувачем класу. Дослідити виклик конструкторів та деструкторів.

Код програми

Модуль SERVANT.h

```
#include <iostream>
#include "PROPERTY.cpp"

class SERVANT {
public:
    //Properties
    PROPERTY<char*> Name;
    PROPERTY<int> Age;
    PROPERTY<int> Experience;

    //Constructors
    SERVANT();
```

```
    SERVANT(char* name, int age, int experience);
    SERVANT(SERVANT& s);

    //Destructor
    ~SERVANT();

    //Functions
    void skip_years();

    static SERVANT* make_one_year_younger(SERVANT &);

    static SERVANT* create_new_servant();

    static void print_servant(SERVANT* s);
};
```

Модуль SERVANT.cpp

```
#include <iostream>
#include "SERVANT.h"

using namespace std;

    //Properties
    PROPERTY<char*> Name;
    PROPERTY<int> Age;
    PROPERTY<int> Experience;

    //Constructors
    SERVANT::SERVANT()
    {
        cout << "Empty constructor called " << endl;
    };
      SERVANT::SERVANT(char* name, int age, int experience) {
        Name = name;
        Age = age;
        Experience = experience;
        cout << "Constructor called by object " << Name() << endl;
    };
    SERVANT::SERVANT(SERVANT& s) {
        Name = s.Name();
        Age = s.Age();
        Experience = s.Experience();
        cout << "Constructor called by object " << Name() << endl;
    };
    SERVANT::~SERVANT() {};

    //Functions
    void SERVANT::skip_years()
    {
        int years_amount;

        cout << "Enter amount of years to skip: ";
        cin >> years_amount;

        this->Age = this->Age() + years_amount;
        this->Experience = this->Experience() + years_amount;
```

```
        cout << "Years Passed!\n\n";
    }

    SERVANT* SERVANT::create_new_servant() {
        char* name = new char[100];
        int age;
        int experience;

        cout << "Enter servant's data: \n";
        cout << "Name:";
        cin >> name;
        cout << "Age:";
        cin >> age;
        cout << "Experience:";
        cin >> experience;
        cout << endl << endl;

        return new SERVANT(name, age, experience);
    }

    SERVANT* SERVANT::make_one_year_younger(SERVANT & s) {
        SERVANT* temp_servant = new SERVANT(s);
        temp_servant->Age = s.Age() - 1;
        cout << "Made servant younger!" << endl << endl;

        return temp_servant;
    }

    void SERVANT::print_servant(SERVANT* s) {
        cout << "Servant: \nName: " << s->Name() << ",\nAge: " << s->Age() <<
",\nExperience: " << s->Experience() << ";\n\n";
          }
```

Модуль main.cpp

```
#include <iostream>
#include <vector>
#include <string>
#include <string.h>
#include "SERVANT.h"

using namespace std;

int main()
{
    //Вказівник на екземпляр класу
    SERVANT* s = new SERVANT();

    // Вказівники на компоненти-функції
    SERVANT* (*create)() = &SERVANT::create_new_servant;
    void (*print)(SERVANT*) = &SERVANT::print_servant;
    void (SERVANT :: * pass_years)() = &SERVANT::skip_years;

    //Servant array in static memory
    cout << "Static array ---------------------" << endl;
    SERVANT department[2];
    char name[4];
    strcpy(name, "Bob");
```

```
    department[0] = *(new SERVANT(name, 33, 12));

    char name1[4];
    strcpy(name1, "Qwe");
    department[1] = *(new SERVANT(name1, 28, 6));

    (*print)(&department[0]);
    (*print)(&department[1]);

    //Servant array in dynamic memory
    cout << "Dynamic array ----------------------" << endl;
    SERVANT* dynamic_department = new SERVANT[2]{
    department[0],
    department[1]
    };

    dynamic_department[0] =
*(SERVANT::make_one_year_younger(dynamic_department[0]));
    dynamic_department[1] =
*(SERVANT::make_one_year_younger(dynamic_department[1]));
    (*print)(&dynamic_department[0]);
    (*print)(&dynamic_department[1]);

    //Виклики функцій через вказівники
    cout << "Solo Item ----------------------" << endl;
    s = (*create)();
    (*print)(s);
    (s->*pass_years)();

    s = SERVANT::make_one_year_younger(*s);
    (*print)(s);



    //Виклик деструктора, так як об'єкт створювався через оператор new
    delete s;
    delete[] department;
    delete[] dynamic_department;

    return 0;
                                        }
```

Модуль PROPERTY.cpp

```
template <typename T>
class PROPERTY {
public:
    virtual ~PROPERTY() = default; //C++11: use override and =default;
    virtual T& operator= (const T& f) { return value = f; }
    virtual const T& operator() () const { return value; }
    virtual explicit operator const T& () const { return value; }
    virtual T* operator->() { return &value; }
protected:
    T value;
};
```

Результат тестування роботи програми

```
Empty constructor called
Static array ---------------------
Empty constructor called
Empty constructor called
Constructor called by object Bob
Constructor called by object Qwe
Servant:
Name: Bob,
Age: 33,
Experience: 12;

Servant:
Name: Qwe,
Age: 28,
Experience: 6;

Dynamic array ---------------------
Constructor called by object Bob
Constructor called by object Qwe
Constructor called by object Bob
Made servant younger!

Constructor called by object Qwe
Made servant younger!

Servant:
Name: Bob,
Age: 32,
Experience: 12;

Servant:
Name: Qwe,
Age: 27,
Experience: 6;

Solo Item ---------------------
Enter servant's data:
Name:Alex
Age:19
Experience:1


Constructor called by object Alex
Servant:
Name: Alex,
Age: 19,
Experience: 1;

Enter amount of years to skip: 2
Years Passed!

Constructor called by object Alex
Made servant younger!

Servant:
Name: Alex,
Age: 20,
Experience: 3;
```

Рисунок 1.1 – Результат тесутвання програми