Лабораторна робота № 8

ШАБЛОНИ ФУНКЦІЙ І КЛАСІВ

Мета. Отримати практичні навички створення шаблонів і використання їх у програмах C++.

Основний зміст роботи.

Створити шаблон заданого класу і використовувати його для даних різних типів.

Короткі теоретичні відомості.

Шаблон функції.

Шаблон функції (інакше, параметрізована функція) визначає загальний набір операцій (алгоритм), які будуть застосовуватися для даних різних типів. При цьому тип даних, над якими функція повинна виконувати операції, передається їй у вигляді параметра на стадії компіляції.

У C ++ параметрізована функція створюється за допомогою ключового слова `template`. Формат шаблону функції:

```
template <class тип_даних> тіп_возвр_значенія
імя_функциі (спісок_параметров) {тіло_функціі}
```

Основні властивості параметрів шаблону функції.

- Імена параметрів шаблону повинні бути унікальними у всьому визначенні шаблону.
- Список параметрів шаблону не може бути порожнім.
- У списку параметрів шаблону може бути декілька параметрів, і кожному з них має передувати ключове слово `class`.
- Ім'я параметра шаблона має всі права імені типу у визначеної шаблоном функції.
- Визначена за допомогою шаблону функція може мати будь-яку кількість непараметрізованих формальних параметрів. Може бути параметризованим і значення, що повертається функцією.
- У списку параметрів прототипу шаблону імена параметрів не зобов'язані збігатися з іменами тих же параметрів у визначенні шаблону.
- При конкретизації параметрізованої функції необхідно, щоб при виклику функції типи фактичних параметрів, відповідні однаково параметрізованим формальним параметрам, були однакові.

Шаблон класу.

Шаблон класу (інакше параметрізований клас) використовується для побудови родового класу. Створюючи родової клас, ви створюєте ціле сімейство споріднених класів, які можна застосовувати до будь-якого типу даних. Таким чином, тип даних, яким оперує клас, вказується як параметр при створенні об'єкту, що належить до цього класу. Подібно до того, як клас визначає правила побудови і формат окремих об'єктів, шаблон класу визначає спосіб побудови окремих класів. У визначенні класу, що входить в шаблон, ім'я класу є не ім'ям окремого класу, а параметрізованим ім'ям сімейства класів.

Загальна форма оголошення параметрізованого класу:

```
template <class тип_даних> class ім'я_класу {. . . };
```

Основні властивості шаблонів класів.

- Компонентні функції параметрізованого класу автоматично є параметрізованими. Їх не обов'язково оголошувати як параметрізрвані за допомогою `template`.
- Дружні функції, які описуються в параметрізованому класі, не є автоматично параметрізованим функціями, тобто за замовчуванням такі функції є дружніми для всіх класів, які організовуються за даним шаблоном.
- Якщо `friend`-функція містить у своєму описі параметр типу параметрізованого класу, то для кожного створеного за даним шаблоном класу є власна `friend`-функція.
- В рамках параметрізованого класу не можна визначити friend-шаблони (дружні параметрізрвані класи).
- З одного боку, шаблони можуть бути похідними (наслідуваться) як від шаблонів, так і від звичайних класів, з іншого боку, вони можуть використовуватися в якості базових для інших шаблонів або класів.
- Шаблони функцій, які є членами класів, не можна описувати як `virtual`.
- Локальні класи не можуть містити шаблони в якості своїх елементів.

Компонентні функції параметрізованих класів.

Реалізація компонентної функції шаблону класу, яка знаходиться поза визначенням шаблону класу, повинна включати додатково наступні два елементи:

- Визначення повинно починатися з ключового слова `template`, за яким слідує такий же спісок_параметров_тіпов в кутових дужках.
- За іменем_класса, попереднім операції області видимості (: :), повинен слідувати спісок_імен_параметров шаблону.

```
template <список_типов> тип_возвр_значенія ім'я_класу <список_
імен_ параметрів>:: імя_функциі (список_параметров) {. . . }
```

Порядок виконання роботи.

1. Створити шаблон заданого класу. Визначити конструктори, деструктор, перевантажену операцію присвоювання ("=") та операції, що задані у варіанті завдання.
2. Написати програму тестування, в якій перевіряється використання шаблону для стандартних типів даних.
3. Виконати тестування.
4. Визначити користувальницький клас, який буде використовуватися в якості параметра шаблону. Визначити в класі необхідні функції і перевантажені операції.
5. Написати програму тестування, в якій перевіряється використання шаблону для користувальницького типу.
6. Виконати тестування.

Зміст звіту.

1. Титульний лист: назва дисципліни, номер та найменування роботи, прізвище, ім'я, по батькові студента, дата виконання.
2. Постановка завдання.
    a. Слід дати конкретну постановку, тобто, вказати шаблон якого класу повинен бути створений, які повинні бути в ньому конструктори, методи, перевантажені операції тощо.
    b. Те ж саме слід вказати для користувальницького класу.
3. Визначення шаблону класу з коментарями.
4. Визначення користувацького класу з коментарями.
5. Реалізація конструкторів, деструктора, операції присвоювання і операцій, які задані у варіанті завдання.
6. Те ж саме для користувальницького класу.
7. Результати тестування. Слід вказати для яких типів і які операції перевірені і які виявлені помилки (або не виявлені)

Питання для самоконтролю.

1. У чому сенс використання шаблонів?
2. Які синтаксис / семантика шаблонів функцій?
3. Які синтаксис / семантика шаблонів класів?
4. Що таке параметри шаблону функції?
5. Перелічите основні властивості параметрів шаблону функції.
6. Як записувати параметр шаблону?
7. Чи можна перевантажувати параметризрвані функції?

Методичні вказівки.

1. Клас реалізувати як динамічний масив. Для цього визначення класу повинно мати такі поля:

- Покажчик на початок масиву;
- Максимальний розмір масиву;
- Поточний розмір масиву.

2. Щоб у вас не виникало проблем, акуратно працюйте з константними об'єктами. наприклад:

• конструктор копіювання слід визначити так:

```
MyTmp (const MyTmp & ob);
```

• операцію присвоювання перевантажити так:

```
MyTmp & operator = (const MyTmp & ob);
```

3. Для шаблонів множин, списків, стеків і черг в якості стандартних типів використовувати символьні, цілі і дійсні типи.

4. Для шаблонів масивів в якості стандартних типів використовувати цілі і дійсні типи. Для користувальницького типу взяти клас "Комплексне число" complex.

```
class complex {
int re; // Дійсна частина
int im; // Уявна частина
public;
// Необхідні функції і перевантажені операції
};
```

5. Тестування повинне бути виконане для всіх типів даних і для всіх операцій.

Варіанти завдань.

1. Класс - одновимірний масив . Додатково перевантажити наступні операції:

<< вивід на консоль;

== порівняння.

2. Класс - одновимірний масив. Додатково перевантажити наступні операції:

[ ] Оператор індекса;

== порівняння.

3. Класс - одновимірний масив. Додатково перевантажити наступні операції:

<< вивід на консоль;

+ - об'єднати два масива;

4. Класс - одновимірний масив . Додатково перевантажити наступні операції:

[ ] Оператор індекса;

+ злиття двох масивів .

5. Класс - одновимірний масив. Додатково перевантажити наступні операції:

<< вивід на консоль;

= = - Перевірка на рівність масивів

6. Клас - однонаправлений список list. Додатково перевантажити наступні операції:

+ - Додати елемент в початок (list + item);

- - Видалити елемент з початку (--list);

= = - Перевірка на рівність.

7. Клас - однонаправлений список list. Додатково перевантажити наступні операції:

+ - Додати елемент в початок (list + item);

- - Видалити елемент з початку (--list);

! = - Перевірка на нерівність.

8. Клас - однонаправлений список list. Додатково перевантажити наступні операції:

+ - Додати елемент в кінець (list + item);

- - Видалити елемент з кінця (типу list--);

! = - Перевірка на нерівність.

9. Клас - однонаправлений список list. Додатково перевантажити наступні операції:

[] - Доступ до елементу в даній точці, наприклад:

int i; Type c;

list L;

c = L [i];

+ - Об'єднати два списки;

10. Клас - стек stack. Додатково перевантажити наступні операції:

+ - Додати елемент в стек;

- - Витягти елемент з стека;

11. Клас - черга queue. Додатково перевантажити наступні операції:

+ - Додати елемент;

- - Витягти елемент;

12. Клас - одновимірний масив. Додатково перевантажити наступні операції:

+ - Додавання масивів;

[] - Доступ за індексом;

+ - Скласти елемент з масивом.

ДОДАТОК

Приклади шаблонів

Приклад шаблона функції

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>
using namespace std;
//Визначення шаблонної функції.

       template <class X> void swapargs(X &a, X &b)
       {
       X temp;
       temp = a;
       a = b;
       b = temp;
       }

int main()
{
setlocale(0,"Russian");
int i = 10, j=20;
double x=10.1, y=23.3;
char a='x', b='z';
cout << "Початкові значення i, j: " << i << ' '<< j<<endl;
cout << "Початкові значення x, y: " << x << ' '<< y<<endl;
cout << "Початкові значення a, b: " << a << ' '<< b<<endl;
swapargs(i, j); // перестановка целых чисел
swapargs(x, y); // перестановка значений с плавающей точкой
swapargs(a, b); // перестановка символов
cout << "Після переустановки i, j: " << i << ' '<< j<<endl;
cout << "Після переустановки x, y: " << x << ' '<< y<<endl;
cout << "Після переустановки a, b: " << a << ' '<< b<<endl;
  getch();
  return 0;

}
```

Демонстрація використання узагальненого класу черзі.

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>
using namespace std;

// Демонстрація використання узагальненого класу черзі.
const int SIZE=100;
       // Створення узагальненого класу queue.
template <class QType>
class queue{
QType q[SIZE];
int sloc, rloc;
public:
queue() { sloc = rloc =0; }
void qput(QType i);
QType qget();
};
       //Внесення об'єкта в чергу.
template <class QType>
void queue<QType>::qput(QType i)
{
if(sloc==SIZE) {
cout<<"Черга заповнена."<<endl;
return;
}
sloc++;
```

```
q[sloc] = i;
}
// Вилучення об'єкта з черги.
template <class QType>
QType queue<QType>::qget()
{
if(rloc == sloc) {
cout<<"Черга пуста."<<endl;
return 0;
}
rloc++;
return q[rloc];
}
       int main()
{
setlocale(0,"Russian");
queue<int> a, b; //Створюємо дві черги для цілих чисел.
a.qput(10);
a.qput(20);
b.qput(19);
b.qput(1);
cout << a.qget() << " ";
cout << a.qget() << " ";
cout << b.qget() << " ";
cout << b.qget() <<endl;

queue<double> c, d; // Створюємо дві черги для double-значень.
c.qput(10.12);
c.qput(-20.0);
d.qput(19.99);
d.qput(0.986);
cout << c.qget() << " ";
cout << c.qget() << " ";
cout << d.qget()<< " ";
cout << d.qget() <<endl;
getch();
return 0;
}
```

Приклад створення і використання параметризованного безпечного масиву.

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>
using namespace std;

// Приклад створення і використання параметризованного безпечного масиву.
const int SIZE = 10;
template <class AType>
class atype {
AType a[SIZE];
public:
atype() {
register int i;
for(i=0; i<SIZE; i++)
       a[i] = i;
}
AType &operator[](int i);
};

// Забезпечення контролю границь для класу a type.
template <class AType>
AType &atype<AType>::operator[](int i)
{
if(i<0 || i> SIZE-1) {
cout << "Значення індексу ";
cout << i << " за межами границь масиву.";
}
```

```
return a[i];
}

int main()
{
	setlocale(0,"Russian");
atype<int> intob; // масив int-значень
atype<double> doubleob; // масив double-значень
int i;
cout << "Масив int-значень: ";
for(i=0; i<SIZE; i++)
	intob[i] = i;
for(i=0; i<SIZE; i++)
	cout << intob[i] << " ";
cout <<endl;
cout << "Масив double-значень: ";
for(i=0; i<SIZE; i++)
	doubleob[i] = (double) i/3;
for(i=0; i<SIZE; i++)
	cout << doubleob[i] << " ";
cout <<endl;
intob[12] = 100; // Помилка часу виконання!
_getch();
return 0;
}
```

Приклад перевантаження оператора <<

```
#include "stdafx.h"
#include <iostream>
#include <conio.h>
using namespace std;

const int SIZE = 8;
struct Complex
{
   double r,e;
};
ostream & operator<<(ostream &t, Complex &C)
{
	t<<C.r<<" + i"<<C.e<<";";
	return t;
}
// Приклад перевантаження оператора <<

template <class AType>
class atype {
public:
AType a[SIZE];
atype() {
//for(int i=0; i<SIZE; i++)
//	a[i] = (double)(rand()%90+ 10)/5;
}
AType &operator[](int i);
};

//Забезпечення контролю границь для класу a type.
template <class AType>
AType &atype<AType>::operator[](int i)
{
if(i<0 || i> SIZE-1) {
cout << "Значення індексу ";
cout << i << " за межами границь масиву.";
}
return a[i];
}
```

```
template <class AType>
ostream & operator<<(ostream &t, atype<AType> &A)
{
	for(int i=0; i<SIZE; i++)
		t<<A.a[i]<<" ";
	t<<endl;
	return t;
}
int main()
{
setlocale(0,"Ukr");
atype<int> intob; // масив int-значень
atype<double> doubleob; // масив double-значень
atype<Complex> complexob; // масив Complex-значень
int i;
cout << "Масив int-значень: ";
 for(i=0; i<SIZE; i++)
	intob[i] = rand()%90 +10;
 cout<<intob; // вивід масиву (використовується перевантажений оператор <<
 cout << "Масив double-значень: ";
 for(i=0; i<SIZE; i++)
	doubleob[i] = (double)(rand()%90 +10)/5;
 cout<<doubleob; // вивід масиву (використовується перевантажений оператор <<
cout <<endl;
cout << "Масив complex-значень: "<<endl;
  for(i=0; i<SIZE; i++)
	{
		complexob[i].r = rand()%90 +10;
		complexob[i].e = rand()%90 +10;
    }
cout << complexob; // вивід масиву (використовується перевантажений оператор <<
cout <<endl;
//intob[12] = 100; // Помилка часу виконання!
_getch();
return 0;
}
```