**Лабораторне заняття №6 (2-й семестр)**

**Перевантаження операторів.**

***Перевантаження операції [ ]***

**Операція індексування []** перевантажується для того, щоб використовувати стандартний запис С++ для доступу до елементів членів класу. Операція "[]" перевантажується як бінарна операція.

Її можна перевантажувати тільки для класу і тільки за допомогою функцій-членів класу.

Однак оскільки ця операція зазвичай використовується ліворуч знака “=”, перевантажена функція має повертати власне значення за посиланням.

В наступному прикладі для класу Vector за допомогою перевантаження операції індексування повертається i-тий елемент масиву цілих чисел beg

int Vector::operator [](int i)

{

if(i<0) cout<<"index <0";

if(i>=size) cout<<"index>size";

return beg[i];

}

Параметр операторної функції **operator[]()** може мати будь який тип даних: ***символ, дійсне число, строка.***

***Перевантаження оператора "( )"***

С++ дозволяє перевантажувати **оператор виклику функції ()**. При його перевантаженні створюється не новий засіб виклику функції, а операторна функція, якій можна передати довільне число параметрів.

У загальному випадку при перевантаженні оператора ( ) визначаються параметри, які необхідно передати функції operator()(). А аргументи, які задаються при використанні оператора ( ) в програмі, копіюються у ці параметри. Об’єкт, який генерує виклик операторної функції, адресується покажчиком this. Наприклад, для класу Vector:

void Vector:: operator()(int n)

{

for(int i=0; i<size; i++)

beg[i]=n\*beg[i];

}

***Перевантаження операторів вводу >> та виведення <<***

У мові С++ передбачено засіб **вводу** і **виведення** стандартних типів даних, з використанням операторів помістити в потік >> і взяти з потоку <<. Ці оператори вже перевантажені в бібліотеці <iosream> для роботи з різними стандартними типами даних. Включаючи строки та адреси. Але ці оператори можна також перевантажувати для вводу та виведення типів даних, які визначені користувачем.

Функції перевантаження операторів помістити в потік>> та взяти з потоку << не можуть бути членами класу. Для того, щоб вони мали доступ до елементів класу ії перевантажують як дружні функції.

ostream &operator<<(ostream &output, const Vector &v)

{

if(v.size==0) out<<"Empty\n";

else

{

for (int i=0; i<v.size; i++)

output<<v.beg[i]<<" ";

out<<endl;

}

return output;

}

istream &operator >>(istream &input, Vector &v)

{

for(int i=0; i<v.size; i++)

{

cout<<">";

input>>v.beg[i];

}

return input;

}

Зверніть увагу, що згідно об’явленню ці функції повертають посилання на об’єкт типу ostream чи istream. Це дозволяє об’єднати в одному складовому виражені декілька операторів виведення.

Операторні функції у цьому випадку мають два параметри. Перший являє собою посилання на потік, якій використовується у лівій частині оператору. Другий являє об’єкт, який стоїть у правій частині оператору. За необхідністю другий параметр також може мати посилання на об’єкт. Саме тіло функції, для розглянутого прикладу, складається з інструкцій виведення чи вводу масиву класу Vector.

***Перевантаження операторів new і delete***

Оператори **new** і **delete** також можуть бути перевантажені у мові С++. Для різного роду задач може виникнути потреба створення власної версії цих операторів.

Для перевантаження використовується наступний формат:

// Виділення пам’яті для об’єкту.

void \*operator new(size\_t size)

{

//Конструктор викликається автоматично.

return pointer\_to\_memory;

}

// Видалення об’єкта.

void operator delete(void \*p)

{//Звільнення па’мяті. Деструктор викликається автоматично

}

Тип **size\_t** спеціально визначено, щоб забезпечити збереження розміру максимально потрібної області пам’яті, яка може бути виділена для об’єкту. Тип **size\_t,** це цілочисельний тип без знаку. Параметр **size\_t** визначає кількість байтів пам’яті, які потрібні для збереження об’єкту.

// Виділення пам’яті для масиву об’єктів.

void \*operator new[](size\_t size)

{// Кожний конструктор викликається автоматично

return pointer\_to\_memory;

}

// Видалення масиву об’єктів.

void operator delete[](void \*p)

{//Деструктор для кожного об’єкту масиву викликається автоматично

}

Необхідно також організувати перевірку виділення динамічної пам’яті.

Наведемо представлення розроблюваного класу в UML:

|  |
| --- |
| Vector |
| - size: int  - \*beg: int |
| + Vector()  +Vector(s:int)  +Vector(&v:const Vector)  +~Vector();  +&operator=(&v:const Vector)  :const Vector  +operator[](i:int):int  +operator()(n:int)  +getSize():int  +\*operator new(size:size\_t)  +operator delete(\*p:void)  +&operator<<(&out:ostream,  &v:const Vector):ostream  +&operator>>(&in:istream,  &v:Vector):istream; |

Заголовковий файл «Vector.h»

#ifndef THREE\_D\_H\_INCLUDED

#define THREE\_D\_H\_INCLUDED

#include <iostream>

using namespace std;

const int MAX\_SIZE=20;

class Vector

{

int size;

int \*beg;

public:

Vector()

{

size=0;

beg=0;

}

Vector(int s);

Vector(const Vector &v);

~Vector();

//перевантаження оператора =

const Vector& operator=(const Vector&v);

//перевантаження операції індексування

int operator[](int i);

//перевантаження оператора виклику функції

void operator()(int n);

int getSize();

//перевантаження операторів виділення та видалення динамічної //пам’яті

void \*operator new(std::size\_t size);

void operator delete(void\* p);

//перевантаження операторів вводу та виведення

friend ostream& operator<<(ostream&out,const Vector&v);

friend istream& operator>>(istream& in, Vector&v);

};

#endif // THREE\_D\_H\_INCLUDED

файл з реалізацією класу «Vector.cpp»

#include "Vector.h"

#include <iostream>

using namespace std;

Vector::Vector(int s)

{

if(s>MAX\_SIZE)

cout<<"Vector length more than MAXSIZE\n";

size=s;

beg=new int [s];

for(int i=0; i<size; i++)

beg[i]=i;

}

Vector::Vector(const Vector &v)

{

size=v.size;

beg=new int [size];

for(int i=0; i<size; i++)

beg[i]=v.beg[i];

}

Vector::~Vector()

{

if (beg!=0)

delete[]beg;

}

const Vector& Vector::operator =(const Vector &v)

{

if(this==&v)

return \*this;

if(beg!=0)

delete []beg;

size=v.size;

beg=new int [size];

for(int i=0; i<size; i++)

beg[i]=v.beg[i];

return\*this;

}

ostream& operator<<(ostream&output, const Vector&v)

{

if(v.size==0) output<<"Empty\n";

else

{

for (int i=0; i<v.size; i++)

output<<v.beg[i]<<" ";

output<<endl;

}

return output;

}

istream& operator >>(istream &input, Vector &v)

{

for(int i=0; i<v.size; i++)

{

cout<<">";

input>>v.beg[i];

}

return input;

}

int Vector::operator [](int i)

{

if(i<0) cout<<"index <0";

if(i>=size) cout<<"index>size";

return beg[i];

}

int Vector::getSize ()

{

return size;

}

void Vector:: operator()(int n)

{

for(int i=0; i<size; i++)

beg[i]=n\*beg[i];

}

void \*Vector::operator new(std::size\_t size)

{

cout<<"\n New memory \n";

Vector \*p=new Vector[size];

if(!p)

cout<<"Not memory \n";

return p;

}

void Vector:: operator delete(void\* p)

{

cout << "Delete memory\n";

return::operator delete(p);

}

файл «main.cpp» з прикладом використання перевантажених операторів

#include "Vector.h"

#include <iostream>

using namespace std;

int main()

{

Vector x(15);

Vector y(x);

cout<<"Massiv x \n";

cout<<x;

cout<<"Nomer?\n";

int i;

cin>>i;

cout<<x[i]<<endl;

cout<<"Massiv y \n";

cout<<y<<endl;

x(3);

cout<<"Massiv x\*3 \n";

cout<<x;

Vector \*p1=new Vector(5);

cout<<"Size="<<p1->getSize();

cout<<"\nel\_3="<<(\*p1)[3];

cout<<"\n"<<\*p1<<endl;

(\*p1)(5);

cout<<"\n"<<\*p1<<endl;

\*p1=y;

cout<<"Massiv p1=y \n";

cout<<"\n"<<\*p1<<endl;

delete p1;

return 0;

}

**Лабораторна робота 6.**

**Хід виконання завдання:**

1. Для заданого поняття (згідно варіанту) змоделювати клас.

Клас повинен включати:

* 1. конструктори;
  2. деструктор;
  3. перевантаження операторів:
* індексування [ ];
* виклику функції ( );
* вводу >> та виведення <<;
* виділення та видалення пам’яті new, delete. Виділення пам’яті організувати для масиву об’єктів.

1. Описати клас в UML-нотації.
2. Написати програму, в якій користувач матиме можливість проводити маніпуляції зі створеним класом. Організувати роботу з масивом екземплярів класу.

**Варіанти завдань.**

1. Дата
2. Багатокутник.
3. Циліндр.
4. Трикутник.
5. Матриця.
6. Час.
7. Трапеція.
8. Відрізок.
9. Конус.
10. Студент.
11. Комплексне число.
12. Ламана.
13. Правильна дріб.
14. Пряма
15. Працівник.
16. Прямокутник.
17. Вектор в просторі.