Додаткова, не обов'язкова лабораторна робота

Перевантаження операторів.

Перевантаження операції [ ]

Операція індексування [] перевантажується для того, щоб використовувати стандартний запис C++ 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 "( )"

C++ 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 C++ 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.

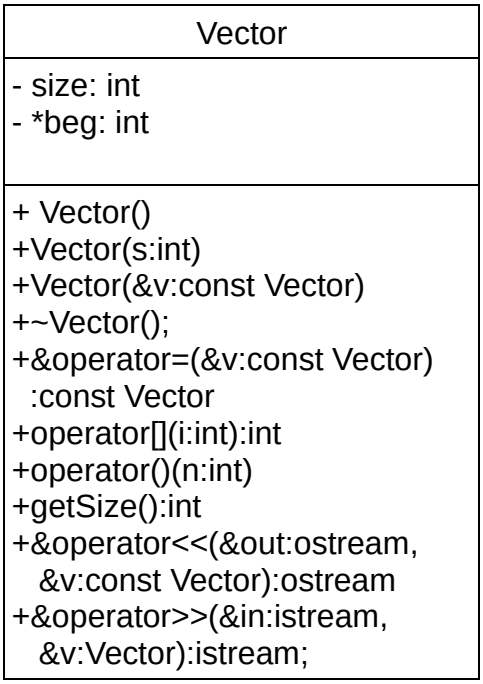
Функції перевантаження операторів помістити в потік << та взяти з потоку >> не можуть бути членами класу. Для того, щоб вони мали доступ до елементів класу її перевантажують як дружні функції.

```
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.

Наведемо представлення розроблюваного класу в UML:



Заголовковий файл «Vector.h»

```
#ifndef VECTOR_H_INCLUDED
#define VECTOR _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 вводу та виведення
    friend ostream& operator<<(ostream&out,const Vector&v);
    friend istream& operator>>(istream& in, Vector&v);
};
#endif // VECTOR_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)
{ cout<<"Copy constructor"<<endl;
    size=v.size;
    beg=new int [size];
    for(int i=0; i<size; i++)
        beg[i]=v.beg[i];
}
```

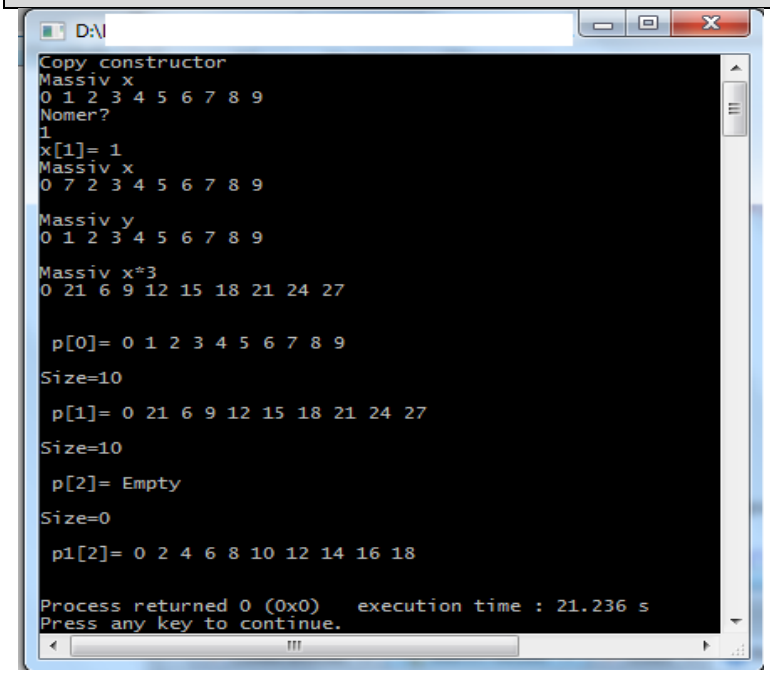
```
Vector::~Vector()
{
    if (beg!=0)
        delete[]beg;
}

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];
}
```

файл «main.cpp» з прикладом використання перевантажених операторів

```
#include "Vector.h"
#include <iostream>
using namespace std;
int main()
{    Vector x(10),y(x);//створення двох об’єктів
     cout<<"Massiv x \n";
     cout<<x;//виклик перевантаженого оператору <<
     cout<<"Nomer?\n";
     int i;
     cin>>i;
     cout<<"x["<<i<<"]= "<<x[i]<<endl;  //виклик перевантаженого оператору []
     x[i]=7;  //виклик перевантаженого оператору []
     cout<<"Massiv x \n";
     cout<<x<<endl;  //виклик перевантаженого оператору <<
     cout<<"Massiv y \n";
     cout<<y<<endl;  //виклик перевантаженого оператору <<
     x(3);  //виклик перевантаженого оператору ( )
     cout<<"Massiv x*3 \n";
     cout<<x<<endl;
     Vector *p1=new Vector[3];  //виділяємо  динамічну пам'ять під масив об’єктів
     *p1=y;  //ініціалізація елементів масиву, виклик перевантаженого оператору =
     *(p1+1)=x;
     for (int i=0; i<3; i++)
      {
        cout<<"\n p["<<i<<"]= "<<p1[i]<<endl;  //перевантаж. оператор <<
        cout<<"Size="<<(p1+i)->getSize()<<endl;
      }

       *(p1+2)=y;
       p1[2](2);//(*(p1+2))(2);  //виклик перевантаженого оператору( )
       cout<<"\n p1[2]= "<<p1[2]<<endl;
       delete [] p1;
    return 0;
}
```



Лабораторна робота 6.

Хід виконання завдання:

1. Для заданого поняття (згідно варіанту) змоделювати клас.
   Клас повинен включати:
   a. конструктори;
   b. деструктор;
   c. перевантаження операторів:
   - індексування [ ];
   - виклику функції ( );
   - вводу >> та виведення <<;

2. Описати клас в UML-нотації.
3. Написати програму, в якій користувач матиме можливість проводити маніпуляції зі створеним класом. Організувати роботу з динамічним об’єктом чи масивом об’єктів 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.
18. Призма
19. Книга
20. Натуральне число
21. Прямокутний трикутник
22. Квадратна матриця
23. Рівнобедрений трикутник

Запитання для самоконтролю:

1. Для чого застосовується перевантаження операторів і операцій?
2. Яким чином можна перевантажити оператори?
3. Як перевантажується оператор індексування [ ]?
4. Для чого і як можна перевантажити оператори вводу та виведення?
5. Особливості перевантаження оператору виклику функції.