Лабораторне заняття №10 (2-й семестр)

Виключні ситуації.

1.Типи помилок, які можуть виникати в програмах

У програмах на C++ можуть виникати помилки. Розрізняють три типи помилок, які можуть виникати у програмах:

- *синтаксичні*. Це помилки в синтаксисі мови C++. Виявляються компілятором.
- *логічні*. Це помилки програміста, які важко виявити на етапі розробки програми. Ці помилки виявляються на етапі виконання під час тестування роботи програми.
- *помилки часу виконання*. Такі помилки виникають під час роботи програми. Помилки часу виконання можуть бути логічними помилками програміста, помилками зовнішніх подій (наприклад, нехватка оперативної пам'яті), невірним введенням даних користувачем тощо. У результаті виникнення помилки часу виконання, програма призупиняє свою роботу. Тому, важливо перехопити цю помилку і правильно обробити її для того, щоб програма продовжила свою роботу без зупинки.

Для обробки помилки часу виконання застосовується механізм обробки виключних ситуацій.

2.Виключна ситуація та виключення

Виключна ситуація – це подія, що призвела до збою в роботі програми. У результаті виникнення виключної ситуації програма не може коректно продовжити своє виконання.

Приклади дій у програмі, що можуть призвести до виникнення виключних ситуацій:

- ділення на нуль;
- нехватка оперативної пам'яті при застосуванні оператора new для її виділення (або іншої функції);
- доступ до елементу масиву за його межами (помилковий індекс);
- переповнення значення для деякого типу;
- взяття кореня з від'ємного числа;
- інші ситуації.

Мова програмування C++ дає можливість перехоплювати виключні ситуації та відповідним чином їх обробляти.

В C++ механізм обробки виключних ситуацій ґрунтується на трьох ключових словах: ***try, catch і throw***. Вони утворюють взаємопов'язану підсистему, в якій вкористаня одного з них передбачає застосування іншого.

Програмні інструкції, які на думку програміста повинні бути проконтрольовані на предмет виникнення помилок (виключень), розміщуються в *try-блоці.* Якщо виключення виникає в цьому блоці, воно генерує певну інформацію за допомогою ключового слова *throw* (викид виключення). Цей виняток може бути перехоплено програмним шляхом за допомогою *catch-блоку* і оброблено відповідним чином.

```
try {
     // try-блок (блок коду, що підлягає перевірці на наявність
  //помилок)
     // генерування виключення оператором throw
}
```

```
catch(type1 argument1)
{
     // catch-блок (обробник виключення типа type1
}
```

Щоб у блоці try згенерувати виключну ситуацію, потрібно використати оператор throw. Оператор throw може бути викликаний всередині блоку try або всередині функції, яка викликається з блоку try.

Загальна форма оператора throw наступна

```
throw виключення;
```

У результаті виконання оператора throw генерується виключення деякого типу. Це виключення повинно бути оброблене в блоці catch.

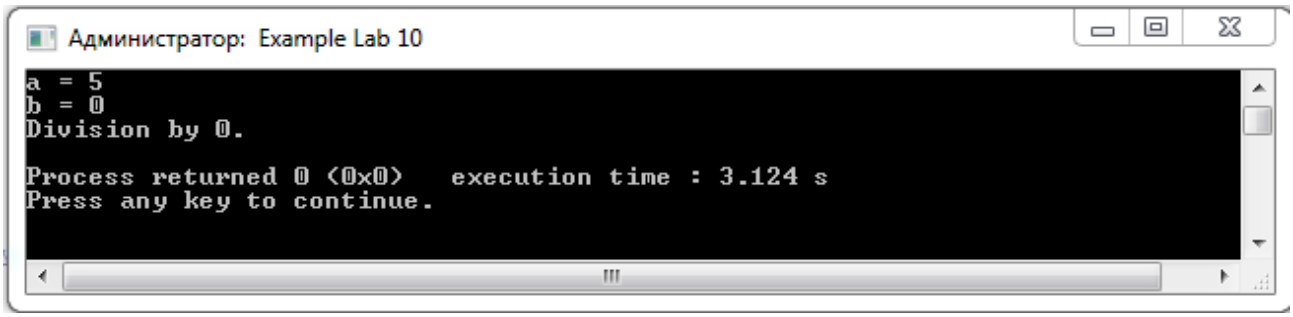
Розглянемо принцип оброблення виключної ситуації на прикладі виникнення виключних ситуацій при діленні на нуль, та визначенні кореню від'ємного числа.

```
#include <iostream>
using namespace std;

void main()
{
  // обробка виразу sqrt(a)/sqrt(b)
  double a, b;
  cout << "a = ";
  cin >> a;
  cout << "b = ";
  cin >> b;
  double c;

  try {
    // початок блока try
    if (b == 0)
      throw 1; // генерування виключення 1
    if (b < 0)
      throw 2; // генерування виключення 2
    if (a < 0)
      throw 2;
    c = sqrt(a) / sqrt(b);
    cout << "c = " << c << endl;
  }
  catch (int e) // перехоплення помилки типу int
```

```
  {
    if (e == 1)
      cout << "Division by 0." << endl;
    if (e == 2)
      cout << "Negative root." << endl;
  }
}
```



Якщо у блоці try виникне виключна ситуація, яка не передбачена блоком catch, то викликається стандартна функція terminate(), яка за замовчуванням викликає функцію abort(). Ця стандартна функція припиняє виконання програми.

Бувають випадки, коли потрібно перехватити усі виключні ситуації підряд. Для цього, в С++ використовується блок catch(…), який має таку загальну форму.

```
catch(...)
{
   // Обробка усіх виключних ситуацій
 // ...
}
```

Практичний приклад.

Розглянемо наступний приклад, що реалізує роботу з класом Vector. Розмір вектора обмежений значенням MAX_SIZE = 30.

Перевантажити для нього операції:

- доступ за індексом ([int i]),
- додавання елемента (+ int),
- видалення елемента (–).

Передбачити генерацію виняткових ситуацій.

Виняткові ситуації генеруються:

1) в конструкторі з параметром при спробі створити вектор більше максимального розміру;
2) в операції [ ], при спробі звернутися до елемента з номером менше 0 або більше поточного розміру вектора;
3) в операції + –, при спробі додати елемент з номером більше максимального розміру;
4) в операції, при спробі видалити елемент з пустого вектора.

Інформація про виняткові ситуації передається за допомогою призначеного для користувача класу.

Vector
- size: int - *beg: int
+ Vector() +Vector(s:int) +Vector(&v:const Vector) +Vector(s: int, mas: int*); +~Vector(); +&operator=(&v:const Vector) :const Vector +operator[](i:int):int +operator()(n:int) +getSize():int +operator +(a:int): Vector +operator --(): Vector

- Заголовковий файл «Error.h»

```
#ifndef ERROR_H_INCLUDED
#define ERROR_H_INCLUDED
#include <string>
#include <iostream>

using namespace std;

class error //клас помилка.
{
    string str;
public:
//конструктор, ініціює атрибут str повідомленням про помилку.
    error(string s)
    {
        str=s;
    }
    void what()
    {
        cout<<str<<endl;  //виводить значення атрибута str.
    }
};

#endif
```

- 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(int s,int* mas);
    Vector(const Vector&v);
    ~Vector();
    const Vector& operator=(const Vector&v);
    int operator[](int i);
    Vector operator+(int a);
    Vector operator--();
    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»

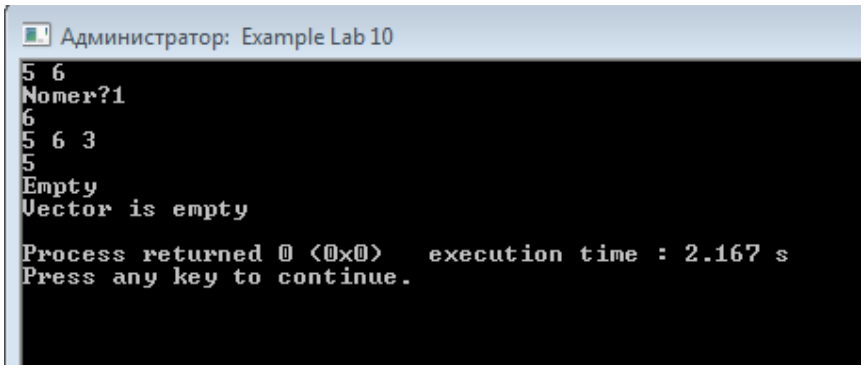
```
#include "Vector.h"
#include "Error.h"
#include <iostream>
using namespace std;
Vector::Vector(int s)
{
    if(s>MAX_SIZE)    throw    error("Vector    length    more    than
MAXSIZE\n");
    size=s;
    beg=new int [s];
    for(int i=0; i<size; i++)
        beg[i]=i+5;
}
Vector::Vector(const Vector &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;
}
Vector::Vector(int s,int *mas)
{
    if(s>MAX_SIZE)    throw    error("Vector    length    more    than
MAXSIZE\n");
    size=s;
    beg=new int[size];
    for(int i=0; i<size; i++)
        beg[i]=mas[i];
}
```

```
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&out, const Vector&v)
{
    if(v.size==0) out<<"Empty\n";
    else
    {
        for (int i=0; i<v.size; i++)
            out<<v.beg[i]<<" ";
        out<<endl;
    }
    return out;
}
istream& operator >>(istream&in, Vector&v)
{
    for(int i=0; i<v.size; i++)
    {
        cout<<">";
        in>>v.beg[i];
    }
    return in;
}
int Vector::operator [](int i)
{
    if(i<0) throw error("index <0");
    if(i>=size) throw error("index>size");
    return beg[i];
}
Vector Vector::operator +(int a)
{
    if(size+1==MAX_SIZE) throw error("Vector is full");
    Vector temp(size+1,beg);
    temp.beg[size]=a;
    return temp;
}
Vector Vector::operator --()
{
    if(size==0) throw error("Vector is empty");
    if (size==1)
    {
        size=0;
        delete[]beg;
        beg=0;
        return *this;
    };
    Vector temp(size,beg);
    delete[]beg;
```

```
    size--;
    beg=new int[size];
    for(int i=0; i<size; i++)
        beg[i]=temp.beg[i];
    return*this;
}
```

- Файл «main.cpp»

```
#include "Vector.h"
#include "Error.h"
#include <iostream>
using namespace std;
int main()
{
    try
    {
        Vector x(2);
        Vector y;
        cout<<x;
        cout<<"Nomer?";
        int i;
        cin>>i;
        cout<<x[i]<<endl;
        y=x+3;
        cout<<y;
        --x;
        cout<<x;
        --x;
        cout<<x;
        --x;
    }
    catch(error &e)
    {
        e.what();
    }
    return 0;
}
```



Завдання для лабораторної роботи:

Хід виконання завдання:

1) Змоделювати клас (згідно варіанту).

2) Перевантажити вказані у варіанті операції.

3) Передбачити генерацію виняткових ситуацій. Інформація про виняткові ситуації передається за допомогою призначеного для користувача класу або ієрархії класів помилок.

4) Написати програму, що створює об`єкти класу і дозволяє генерувати виняткові ситуації

Варіанти завдань

1) Клас-контейнер *Vector* з елементами типу *int*. Реалізувати операції: [] – доступу за індексом; ( ) – визначення розміру вектора; + число – додає константу до всіх елементів вектора, – *n* – видаляє *n* елементів з кінця вектора.

2) Клас-контейнер *Vector* з елементами типу *int.* Реалізувати операції: [] – доступу за індексом; *int* ()– визначення розміру вектора; – *n* – видаляє *n* елементів з кінця вектора; + *N* – додає n елементів у кінець вектора.

3) Клас-контейнер *Vector* з елементами типу *int*. Реалізувати операції: [] – доступу за індексом; + + додає елемент у вектор (постфіксна операція додає елемент у кінець, префіксна – в початок).

4) Клас-контейнер *Vector* з елементами типу *int*. Реалізувати операції: [] – доступу за індексом; ()– визначення розміру вектора; – – видаляє елемент із вектора (постфіксна операція видаляє елемент у кінці вектора, префіксна – на початку).

5) Клас-контейнер *Vector* з елементами типу *int*. Реалізувати операції: [] – доступу за індексом; *int* ()– визначення розміру вектора; * вектор – множення елементів векторів *a* [i] * *b* [i]; + *N* – перехід управо до елемента з номером *n* .

6) Клас-контейнер *Vector* з елементами типу *int*. Реалізувати операції: [] – доступу за індексом; ( ) – визначення розміру вектора, – *n* – видаляє *n* елементів з кінця вектора; + *N* – додає n елементів у кінець вектора.

7) Клас-контейнер *Vector* з елементами типу *int.* Реалізувати операції: [] – доступу за індексом; *int* ()– визначення розміру вектора; / число – заходить результат цілочисельного ділення для всіх елементів вектора – *n* – видаляє *n* елементів з кінця вектора.

8) Клас-контейнер *Vector* з елементами типу *int*. Реалізувати операції: [] – доступу за індексом; + + додає елемент у вектор (постфіксна операція додає елемент у кінець, префіксна – в початок); ()– визначення розміру вектора.

9) Клас-контейнер *Vector* з елементами типу *int*. Реалізувати операції: [] – доступу за індексом; ()– визначення розміру вектора; – – видаляє елемент із вектора (постфіксна операція видаляє елемент у кінці вектора, префіксна – на початку).

10) Клас-контейнер *Vector* з елементами типу *int*. Реалізувати операції: [] – доступу за індексом; *int* ()– визначення розміру вектора; * вектор – множення елементів векторів *a* [i] * *b* [i]; , – *n* – видаляє *n* елементів з кінця вектора.

11) Клас-контейнер *Vector* з елементами типу *int*. Реалізувати операції: [] – доступу за індексом; ( ) – визначення розміру вектора; + число – додає константу до всіх елементів вектора; + *N* – перехід управо до елемента з номером *n* .

12) Клас-контейнер *Vector* з елементами типу *int.* Реалізувати операції: [] – доступу за індексом; *int* ()– визначення розміру вектора; – *n* – видаляє *n* елементів з кінця вектора; + *N* – додає n елементів у кінець вектора.

13) Клас-контейнер *Vector* з елементами типу *int*. Реалізувати операції: [] – доступу за індексом; + + додає елемент у вектор (постфіксна операція додає елемент у кінець,

префіксна – в початок); *int* ()– визначення розміру вектора.

14) Клас-контейнер *Vector* з елементами типу *int*. Реалізувати операції: [] – доступу за індексом; ()– визначення розміру вектора; – – видаляє елемент із вектора (постфіксна операція видаляє елемент у кінці вектора, префіксна – на початку).

15) Клас-контейнер *Vector* з елементами типу *int*. Реалізувати операції: [] – доступу за індексом; *int* ()– визначення розміру вектора; * вектор – множення елементів векторів *a* [i] * *b* [i]; + *N* – перехід управо до елемента з номером *n* .

16) Клас-контейнер *Vector* з елементами типу *int*. Реалізувати операції: [] – доступу за індексом; ( ) – визначення розміру вектора, – *n* – видаляє *n* елементів з кінця вектора; + *N* – додає n елементів у кінець вектора.

17) Клас-контейнер *Vector* з елементами типу *int.* Реалізувати операції: [] – доступу за індексом; *int* ()– визначення розміру вектора; ; + число – додає константу до всіх елементів вектора – *n* – видаляє *n* елементів з кінця вектора.

18) Клас-контейнер *Vector* з елементами типу *int*. Реалізувати операції: [] – доступу за індексом; + + додає елемент у вектор (постфіксна операція додає елемент у кінець, префіксна – в початок); ()– визначення розміру вектора.

19) Клас-контейнер *Vector* з елементами типу *int*. Реалізувати операції: [] – доступу за індексом; ()– визначення розміру вектора; – – видаляє елемент із вектора (постфіксна операція видаляє елемент у кінці вектора, префіксна – на початку).

20) Клас-контейнер *Vector* з елементами типу *int*. Реалізувати операції: [] – доступу за індексом; *int* ()– визначення розміру вектора; * вектор – множення елементів векторів *a* [i] * *b* [i]; , – *n* – видаляє *n* елементів з кінця вектора.

21) Клас-контейнер *Vector* з елементами типу *int*. Реалізувати операції: [] – доступу за індексом; ( ) – визначення розміру вектора; % знаходить останок від ділення на число для всіх елементів вектора; + *N* – перехід управо до елемента з номером *n* .

22) Клас-контейнер *Vector* з елементами типу *int*. Реалізувати операції: [] – доступу за індексом; ( ) – визначення розміру вектора; - число – віднімає константу від всіх елементів вектора, – *n* – видаляє *n* елементів з кінця вектора.

23) Клас-контейнер *Vector* з елементами типу *int.* Реалізувати операції: [] – доступу за індексом; *int* ()– визначення розміру вектора; – *n* – видаляє *n* елементів з кінця вектора; + *N* – додає n елементів у кінець вектора.

Контрольні запитання

1) Які типи помилок можуть виникати в програмах на С++?
2) Коли застосовують механізм обробки виключних ситуацій в програмах?
3) У чому полягає механізм обробки виключних ситуацій?