Виключення (Exception)

Потреба у виключеннях

```
setlocale(LC_ALL, "ukr");
int num1;
int num2;
int var = 3; // керуюча змінна для while
while (var != 0) { // цикл
    cout << "Введить число num1: ";
    cin >> num1;
    cout << "Введить число num2: ";
    cin >> num2;
    cout << "num1 + num2 = " << num1 + num2 << endl;
    cout << "num1 / num2 = " << num1 / num2 << endl;
    cout << "num1 - num2 = " << num1 - num2 << endl;
    cout << "==========================" << endl << endl;
   var--;
}
```

Приклад Exception:try-catch

```
while (var != 0) {
    cout << "Введить число num1: ";        cin >> num1;
    cout << "Введить число num2: ";        cin >> num2;
    cout << "num1 + num2 = " << num1 + num2 << endl;
    cout << "num1 / num2 = ";
    try { // тут код, який може викликати помилку
        if (num2 == 0) {
            throw 42; // генерувати ціле число 42
        }
        cout << num1 / num2 << endl; // ділення
    }
    catch (int thr) { // сюда передається число, яке згенерував throw
        cout << "Помилка №" << thr << " - ділення на 0!!!" << endl;
    }
    cout << "num1 - num2 = " << num1 - num2 << endl;
    var--;
}
```

Ділення на нуль

```
int delenie(int n1, int n2) {
    if (n2 == 0) {
        throw 42; // виклик виключення
    }
    return n1 / n2;
};

    //У цьому прикладі, якщо num2 дорівнює 0, throw генерує рядок, а не число.
Строка “падає” у try-catch-блок і виводиться на екран.
    //Щоб виняток спрацював правильно, цю функцію треба викликати в блоці
try..catch:

    try {
        cout << delenie(7, 0); // функція що кидає виключення
        cout << endl;
    } catch (int thr) { // відловлення виключень
        cout << "Помилка №" << thr << " Ділення на 0!!!" << endl;
    }
```

C++ не містить виключення на простих типах

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

int main(){

  int x = 45;
  int y =0;
  int z =0;
  try{
       z = x/y;

  }
  catch(...){
     cout<<"div By zero";
  }

}
```

Блок try — catch - throw

```
1) try {
  // Код роботи
  // if ( A)  throw 1; // виключення що кидає 1
  // if ( B)  throw ‘a’; // виключення що кидає символ
  // if(C) throw “some error”; // виключення що кидає рядок

}
catch (int param) { cout << "int exception"; } // відловлює ціле число
catch (char param) { cout << "char exception"; }// відловлює символ
catch (...) { cout << "default exception"; } // відловлює все інше
```

2) ***<TypeFun> <myfunction> (<Type> param) throw (<TypeOut>);***

```
double myfunction (char param) throw (int); /* функція створює
виключення що кидає ціле число */
```

Форма throw

```
#include <iostream>
using namespace std;
// функція може згенерирувати тільки int, char и double
void Xhandler(int test) throw (int, char, double){
if(test==0) throw test; // генерація int
if(test==1) throw 'a'; // генерація char
if(test==2) throw 123.23; // генерація double
}

/*
// дана функція не може згенерувати виключення
void Xhandler2(int test) throw(){
// це непрацює
if(test==0) throw test;
if(test==1) throw 'a';
if(test==2) throw 123.23;
}
*/
```

```
int main(){
cout << "start\n";
  try {
    Xhandler(0); // передаємо 0,1,2 в Xhandler()
    Xhandler(1);
    Xhandler(2);
}
catch (int i) {
    cout << "Caught an integer\n";
}
catch (char c) {
   cout << "Caught char\n";
}
catch(double d) {
   cout << "Caught double\n";
}
cout << "end";
return 0;
}
```

Повторна генерація виключення

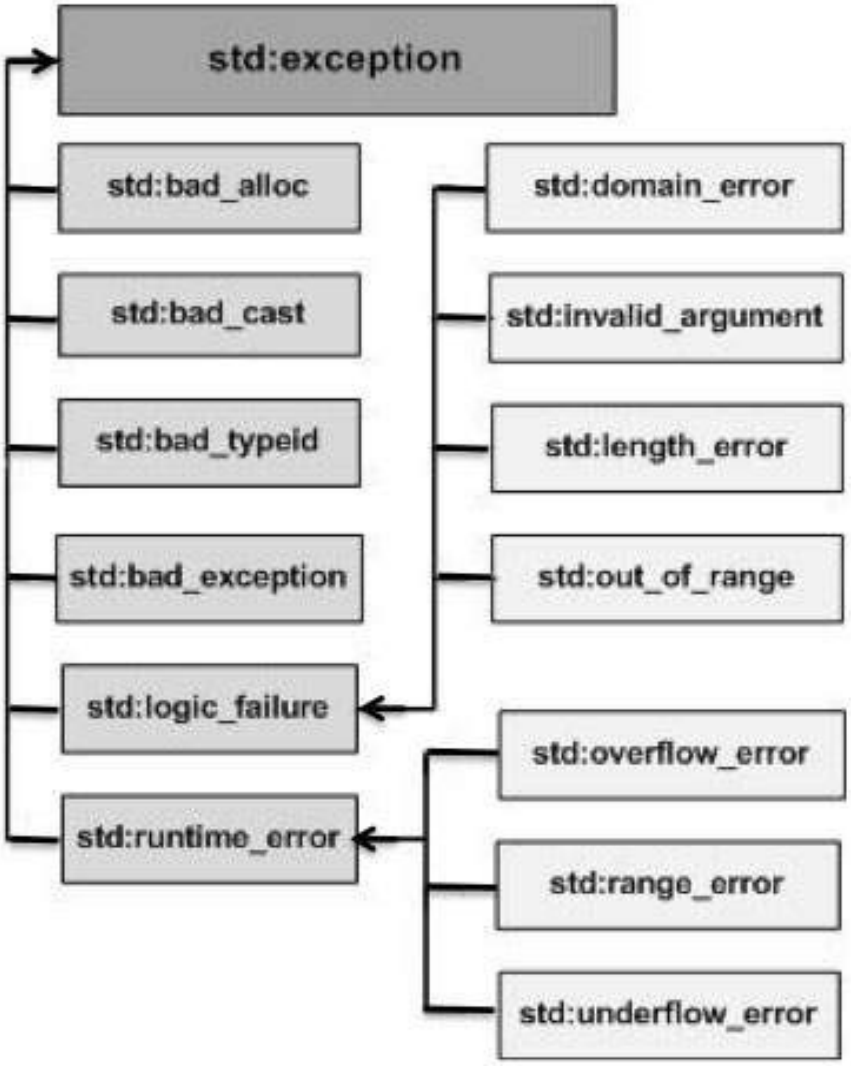
```
#include <iostream>
using namespace std;
void Xhandler(){
  try {
       throw "hello"; // генерація char *
   }
   catch (const char*) { // перехват char *  //error not const
       cout << "Caught char * inside Xhandler\n";
       throw "aa"; // повторна генерація char * зовні функції
   }
}
int main() {
 cout << "Start\n";
  try{
      Xhandler();
   }
  catch(const char *) { // error not const
   cout << "Caught char * inside main\n";
   }
  cout << "End";
}
```

Відловлення виключення

```
#include <iostream>       // std::cerr
#include <typeinfo>       // operator typeid
#include <exception>      // std::exception

class Polymorphic {
      virtual void member(){}
};
int main () {
  try {
    Polymorphic * pb = 0;
    typeid(*pb);  // виключення bad_typeid
  }
  catch (std::exception& e)  { // відловлює виключення
    std::cerr << "exception caught: " << e.what() << '\n';
  }
  return 0;
}
```

Стандартні виключення




```
class exception {
public:
  exception () throw();
  exception (const exception&) throw();

  exception& operator= (const
exception&) throw();

  virtual ~exception() throw();

  virtual const char* what() const throw();
}
```

Приклад: Обробка файлів

```
int func(string full_path){ // функція читання файлів
   ifstream file(full_path.c_str(), ios::binary);  // файлова змінна
   if(file.is_open()) {
        int magic_number = 0; // читаємо ціле число
        file.read((char*)&magic_number, sizeof(magic_number));
        if(magic_number!=42) throw magic_number; // якщо воно немагічне- виключення
        /* інакше робимо якусь магію .... */
    } else { // генеруємо runtime_error виключення
        throw runtime_error("Unable to open file `" + full_path + "`!");
    }
return 0;
}
*****

 // використання в реальному коді
 try                        {    func("1.txt");    }
 catch(runtime_error e){    cout<<e.what()<< endl;   } // виведення помилки
 catch(int a)               {   cout<<"a="<<a<<endl;    }
```

```
#include <iostream>

using namespace std;
int main(int argc, char **argv){

    int A=25, B=5, C=0;
    try {
        std::cout<<"A "<<std::endl;
        std::cin>>A;
        std::cout<<"B "<<std::endl;
        std::cin>>B;
        if (B==0) throw "Error";
        C = A/B;
        cout<<"C=" <<C<<endl;
    }
catch( const char* e) {
        std::cout<<e<<std::endl;    }
catch(...) {
       cout<<"A=" <<A<<"B=" <<B<<endl;    }
```

Виключення out_of_range, bad_alloc

```
//int x[9]; // з масивом не спрацює
vector<int> x(9);
try {
    for(int i =0; i<10; ++i) {
        x[i]=i;
    }
    cout<<"Все гаразд"<<endl;
} catch(out_of_range exception) {
    cout<<exception.what();
} catch(...) {
  cout<<"невідома проблема";
}
```

```
int * arr;

try {
    arr = new int[1000000000];
    cout<<"Все гаразд"<<endl;
     delete[] arr;
} catch(bad_alloc ex) {
    cout<<ex.what();
}
```

Створення виключень

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

struct MyException : public exception { // MyException — нащадок exception
  const char* what() const throw() { // перевантаження методу what
     return "C++ Exception";
   }
};

int main() {
   try {
      throw MyException(); // перевірка роботи
   } catch(MyException& e) { // відловлюємо виключення
      std::cout << "MyException caught" << std::endl;
      std::cout << e.what() << std::endl;
   } catch(std::exception& e) {
      //Обробка інших помилок
   }
}
```

Створення виключень - 2

```
struct MyException : public exception {
   int line;  // будемо зберігати ще номер рядка з помилкою
   MyException(int m){ line = m; }
   const char * what () const throw () {
      return "C++ Exception";
   }
   int getLine(){ return line;  } // можемо створити власний метод
};

int my_func() {
   try {
      throw MyException(2);
   } catch(MyException& e) {
      std::cout << "MyException caught" << std::endl;
      std::cout << e.what() << std::endl;
      std::cout << e.getLine() << std::endl;
   } catch(std::exception& e) {
      //Інші помилки
      std::cout << e.what() << std::endl;
   }
}
```

Створення виключень-3

```
class BasicQueueException : public std::logic_error {
public:
  BasicQueueException(const char* message):std::logic_error(message){}
};
class EmptyQueueException : public BasicQueueException {
public:
  EmptyQueueException():BasicQueueException("Queue empty"){}
};
void Queue::pop() {
    if(isEmpty()) throw EmptyQueueException();
    // ....
}
// ...
try{
    q.pop();
}
catch(BasicQueueException& e) {
    std::cerr << e.what() << std::endl;
}
```

nothrow

```
// Приклад nothrow
#include <iostream>     // std::cout
#include <new>          // std::nothrow

int main () {
  std::cout << "Attempting to allocate 1 MiB... ";
  char* p = new (std::nothrow) char [1048576]; // не створюємо виключення
                                               // а перевіряємо результат функції
  if (!p) {            // null pointer конвертується до 0 , тобто false
    std::cout << "Failed!\n";
  }
  else {
    std::cout << "Succeeded!\n";
    delete[] p;
  }
```

Assert

```
#include <cassert> // підключає assert()
int factorial(int n) {
// виключимо випадок  від'ємних чисел
assert(n >= 0);
// виключимо випадок  великих чисел
assert(n<11);
// можна було: assert(n >= 0 && n<11);
if (n < 2) { return 1; }
return factorial(n - 1) * n;
} //Assertion failed: exp, file , line

int factorial2(int n) {
int result = 1;
while (n > 1) {
        //  обробимо переповнення
        assert(result <= INT_MAX / n);
        result *= n; --n;
        }
return result; //Assertion failed: exp, file , line
}
```

N.B.

0) assert викликає функцію abort() і завершує не підчищаючи програму

1)Включити повідомлення про помилку можна:
assert(<вираз> && “Повідомлення");

Приклад:
assert(found && “Not found in database");

2) **#define NDEBUG**
вимикає всі **assert** до кінця програми