Programowanie 2. Język C++. Wykład 11.

11.1 Obsługa błędów i wyjątków

błąd - przyczyna, która powoduje niewłaściwe działanie programu.

error – różnica między spodziewanym a faktycznym działaniem programu.
fault, bug – defekt, przyczyna, która powoduje niewłaściwe działanie programu.
failure – stan w którym program nie działa właściwe.
mistake – działanie człowieka, które powoduje niewłaściwe działanie programu.

wyjątek (exception) – przewidywalne przez programistę niewłaściwe zachowanie programu wykonywalnego.

wyjątek (*IEEE 610, Standard Computer Dictionary*) - zdarzenie które powoduje niewłaściwe działanie programu.
Rozróżnia się następujące typy wyjątków:

- niewłaściwe adresowanie,
- niewłaściwe dane,
- niewłaściwie wykonana operacja,
- przepełnienie (overflow exception),
- niedopełnienie (underflow exception),
- niewłaściwa ochrona (protection exception).

wyjątek – jest to obiekt, który przekazuje informację o niewłaściwym działaniu programu do innego obszaru programu.

debugging - proces wykrywania i usuwania błędów w programie.

obsługa wyjątków – proces analizy zdarzeń które mogą powodować niewłaściwe zachowanie programu i definiowanie procedur alternatywnych zachowań programu.

11.2 Polecenia try, throw, catch

Składnia polecenia: try, throw, catch.

```
try
{
      <instrukcje>;
       throw <obiekt>;
}
catch(<obiekt>)
{

      <instrukcje>;
}
```

Programowanie 2. Język C++. Wykład 11.

Przykład 1. Polecenie try, throw & catch (w11-01-try-catch.cpp).

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

void main()
{
      try
      {
        throw "tekst z obszaru try"; // wyrzuć tekst
      }

      catch(const char * msg) // złap wyrzucony tekst i przypisz do wskaźnika msg
      {
            cout << msg << endl;
      }
 }
```

W bloku `try{}`, instrukcje po poleceniu `throw` nie są już wykonywane.

Przykład 2. Polecenie try, throw & catch (w11-02-try-throw-catch.cpp)

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

void main()
{
      try
      {
            cout << "Przed throw" << endl;
            throw 007;                  // wyrzuć liczbę 7

            cout << "Po throw" << endl; // instrukcja nie zostanie wykonana
      }

      catch (int i)                     // złap wyrzuconą liczbę
      {
            cout << "Blad #" << i << endl;
      }
}
```

Instrukcja `catch()` stosowana jest do odpowiedniego ‘wyrzuconego’ typu.

Przykład 3. Wyłapanie odpowiedniego typu przez instrukcję `catch()`(w11-03-try-throw-catch.cpp).

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

void main()
{
	try
	{
		// throw 7;
		throw 3.14;
	}
	catch (int i)
	{
		cout << "Blad, #" << i << endl;
	}
	catch (double d)    // złapany typ double
	{
		cout << "Blad, wpisales liczbe: " << d << endl;
	}
}
```

`catch()`wyłapuje wyrzucony typ gdy jest tego samego typu co typ wyrzucany, typu `const` lub referencją typu wyrzucanego.

Przykład 4. Wyłapanie typu const, referencji typu wyrzucanego (w11-04-catch-throw.cpp).

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

void main()
{
	try
	{
		int x = 7;
		throw x;
	}
	// polecenie catch() wyłapuje wyrzuconą zmienną typu int gdy:
	// catch (const int ci), catch (int i), catch (int & r)
/*	catch (int i)
	{
		cout << "Blad, x=  " << i << endl;
	}
*/
/*	catch (const int ci)
	{
		cout << "Blad, x=  " << ci << endl;
	}
*/
	catch (int & r)
	{
		cout << "Blad, x=  " << r << endl;
	}
}
```

Przykład 5. Try&catch obiektowo (w11-05-catch-throw.cpp).

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

class A{
public:
	A(){ i=100; }
	int i;
};

class B: public A {};

void main()
{
	try
	{
		B b;
		throw b;
	}
	catch (A a)
	{
		cout << "Blad, a.i=  " << a.i << endl;
	}

/*	catch (A &a)  // też OK
	{
		cout << "Blad, a.i=  " << a.i << endl;
	}
*/
}
```

Przykład 6. Try&catch obiektowo (w11-06-catch-throw.cpp).

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

class A{
public:
	A(){ i=100; }
	int i;
};

class B: public A {};

void main()
{
	try
	{
		B b;
		B * pb = &b;
		throw pb;
	}
	catch (A * pa)
	{
		cout << "Blad, pa->i=  " << pa->i << endl;
	}
}
```

Przykład 7. Obsługa wyjątku przepełnienia (w11-07-overflow.cpp).

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

class Wyjatek {
public:
      Wyjatek(){ cout << "Wyjatek()" << endl; }
      ~Wyjatek(){ cout << "~Wyjatek()" << endl; }
      void f() {cout << "obsluga wyjatku" << endl; }
};

void main() {
      try
      {
            int i = 4294967296;
            if (i == 0)
                  throw Wyjatek();
      }
      catch (Wyjatek & e){
            e.f();
      }
}
```

Przykład 8. Klasa do obsługi błędów (w11-08-ErrorInfo.cpp).

```
#include <iostream>
#include <string>

class ErrorInfo   {
public:
    std::string gdzie;
    std::string przyczyna;
};

void f(bool b)
{
    ErrorInfo e;

      if (b==0){
        e.gdzie = "f()";
        e.przyczyna = "blad wywolania ";
        throw e;
      }
}

void main()
{
    try
    {
        f(0);
    }
    catch (ErrorInfo const &e)
    {
        std::cout << e.przyczyna << e.gdzie << std::endl;
    }
}
```

Metoda ***fun()*** wyrzuca obiekt utworzony za pomocą konstruktora kopiującego.
W poleceniu `catch (A & e)` tworzona jest referencja typu A.
W poleceniu `catch (A e)` tworzony jest obiekt typu A za pomocą konstruktora kopiującego.

Przykład 9. Konstruktor kopiujący (w11-09-throwKonstrktorKopiujacy.cpp).

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

class A {
public:
      A(string s) : str(s){   cout << "A(string)" << endl;  }
      A(A const &r) : str(r.str + " (kopia)")  { cout << "A(A&r)" << endl;  }
      ~A(){cout << "~A()" << endl; }

      void fun();
      void msg(){ cout << str << endl;   }

private:
        string  str;
};

void A::fun(){
            A e(" *** obiekt lokalny, w fun() *** ");
                                        // wywoływany jest A(A const &r)
            throw e;
      }

void main(){
    A a("obiekt");    // wywoływany jest A(string)

      try
      {
            a.fun();
      }
      catch (A & e)     {
            cout << "wyjatek, " ;
            e.msg();
      }
}
```

```
/* Wynik działania programu:
A(string)
A(string)
A(A&r)
~A()
wyjatek,  *** obiekt lokalny, w fun() ***  (kopia)
~A()
~A()
*/
```

Przykład 10. Klasa wyjątek (w11-10-Wyjatek.cpp).

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

class Wyjatek {
   public:
         Wyjatek(char *);
         char* pokazStr() { return str; }
  private:
        char * str;
 };

Wyjatek::Wyjatek(char * pch) {
      str = new char[80];
      char komunikat[] = "Brak pamieci dla: ";

      strncpy(str,komunikat,60); // funkcja przypisuje 60 znaków
                                 // str = komunikat; zwraca str,
      strncat(str,pch,19);    // funkcja łączy 19 znaków pch z str
 }

void main(){
      try
      {
            int * pi = new int;
            if (pi)
                  throw Wyjatek("int");
      }
      catch (Wyjatek & e)
      {
            cout << e.pokazStr() << endl;
      }
}
```

Przykład 11. Polecenie try&catch i metody wirtualne (w11-11-try-catch-virtual.cpp).

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

class A {
public:
	virtual void f() { cout << "virtual f() z A" << endl; }
};

class B : public A {
public:
	void f()  { cout << "f() z B" << endl;  }
};

void g() { throw B();}

void main()
{
	try
	{
		g();
	}
	catch(A a)
	{
		a.f();
	}
	try
	{
		g();
	}
	catch(A& a)
	{
		a.f();
	}
}
```

```
/* Wynik działania programu:
virtual f() z A
f() z B
*/
```

11.3 CLI C++, klasa `Exception`

Przykład 1. Użycie obiektów klasy `Exception` (w11-12-catch-throw-cli.cpp).

```
#include "stdafx.h"
using namespace System;

void  main() {

    int x = 0;

      try
    {
        int y = 100 / x;
    }
      catch ( ArithmeticException^ e )
    {
        Console::WriteLine( "ArithmeticException Handler: {0}", e );
    }

//    catch ( Exception^ e )
//    {
//        Console::WriteLine( "Generic Exception Handler: {0}", e );
//    }

}

/* wynik działania program:
ArithmeticException Handler: System.DivideByZeroException:
Attempted to divide by zero.
at main() in c:\cl rcon\clrcon.cpp:line 10
*/
```

Elementy klasy `Exception`
Konstruktor

Exception	Przeładowany. Initializes a new instance of the Exception class.

Metody

Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
Finalize	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection. (Inherited from Object.)
GetBaseException	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
GetObjectData	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType	Gets the runtime type of the current instance.
MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)
ToString	Creates and returns a string representation of the current exception. (Overrides Object..::.ToString()()().)

Properties

Data	Gets a collection of key/value pairs that provide additional user-defined information about the exception.
HelpLink	Gets or sets a link to the help file associated with this exception.
HResult	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.
InnerException	Gets the Exception instance that caused the current exception.
Message	Gets a message that describes the current exception.
Source	Gets or sets the name of the application or the object that causes the error.
StackTrace	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite	Gets the method that throws the current exception.