Tratarea exceptiilor in C++

Tratarea exceptiilor permite tratarea problemelor care pot aparea intr-un program intr-un mod mai organizat. Avantajul tratarii exceptiilor il costituie o automatizare mai mare a codului.

Tratarea exceptiilor se realizeaza prin blocuri *try…catch* si pot fi aruncate cu *throw*.

try – delimiteaza sectiunea de cod unde se cauta exceptiile,
catch - este folosit pentru a determina tipul exceptiei si modul in care vor fi tratate exceptiile prinse
throw - se poate apela de oriunde din codul programului, intr-un bloc *try…catch* sau functie.

Daca o eroare este aruncata si nu este prinsa intr-un bloc *try…catch* ea va determina o terminare anormala a programului prin apelul functiilor abort() sau terminate().

Aplicatii: [Jamsa & Klander, Stroustrup]

1. Scrierea unui try-catch simplu

```
int  main(void)
 {
   cout << "Start" << endl;
   try {
      cout << "Inside try block." << endl;
      throw 100;
      cout << "This will not execute.";
    }
   catch(int i) {
      cout << "Caught an exception -- value is: ";
      cout << i << endl;
    }
   cout << "End";
   return 0;
 }
```

In loc sa asteptam ca programul sa comita o eroare utilizam *throw* pentru producerea unei erori.

Dupa ce blocul *try* lanseaza eroarea, blocul *catch* o capteaza si prelucreaza valoarea transmisa de instructiunea *throw*.

2. Exceptiile sunt specifice tipurilor

```
int  main(void)
 {
   cout << "Start" << endl;
   try {
      cout << "Inside try block." << endl;
      throw 100;
      cout << "This will not execute.";
    }
   catch(double d) {
      cout << "Caught a double exception -- value is: ";
      cout << d << endl;
    }
   cout << "End";
   return 0;
 }
```

Exceptia tratata in cadrul blocului *try* trebuie sa aiba acelasi tip cu tipul specificat de blocul *catch*.

Programul de mai sus capteaza o exceptie de tip *double*, iar blocul *try* lanseaza o exceptie de tip *int*. ➔ Anormal program termination

3. Lansare exceptii cu o functie din cadrul blocului try

```
void XHandler(int test)
 {
    cout << "Inside XHandler, test is:" << test << endl;
    if(test) throw test;
 }

int main(void)
 {
   cout << "Start: " << endl;
   try {
      cout << "Inside try block." << endl;
      XHandler(1);
      XHandler(2);
      XHandler(0);
    }
catch(int i) {
   cout << "Caught an exception. Value is: ";
   cout << i << endl;
 }
cout << "End ";
return 0;
 }
```

4. Bloc try intr-o functie

```
void XHandler(int test)
 {
   try {
      if(test) throw test;
    }
   catch(int i)
    {
      cout << "Caught exception #: " << i << endl;
    }
 }

int main(void)
 {
   cout << "Start: " << endl;
   XHandler(1);
   XHandler(2);
   XHandler(0);
   XHandler(3);
   cout << "End";
   return 0;
 }
```

Atunci cand plasam un bloc *try* intr-o functie, limbajul C++ reinitializeaza blocul de fiecare data cand intram in acea functie.

Programul lanseaza doar 3 exceptii desi avem 4 apeluri deoarece apelul XHandler(0) este evaluat ca fals.

5. Cand se executa instructiunea *catch*

```
int main(void)
 {
   cout << "Start" << endl;
   try
```

```
        {
           cout << "Inside try block." << endl;
           cout << "Still inside try block." << endl;
         }
        catch(int i)
        {
           cout << "Caught an exception--value is: " << endl;
           cout << i << endl;
         }
        cout << "End";
        return 0;
        }
```

Instructiunile din blocul *catch* se vor executa doar daca programul lanseaza o exceptie in cadrul blocului *try*.

6. Mai multe *catch* cu un singur *try*

```
void XHandler(int test)
 {
   try
   {
      if(test==0) throw test;
      if(test==1) throw "String";
      if(test==2) throw 123.23;
    }
   catch(int i)
    {
      cout << "Caught exception #: " << i << endl;
    }
   catch(char *str)
    {
      cout << "Caught string exception: " << str << endl;
    }
   catch(double d)
    {
      cout << "Caught exception #: " << d << endl;
    }
 }

int main(void)
 {
   cout << "Start: " << endl;
   XHandler(0);
   XHandler(1);
```

```
        XHandler(2);
        cout << "End";
       return 0;
     }
```

7. (…) cu exceptii

Sintaxa:

```
      try
       {
          // instructiuni
        }
       catch(...)
        {
          //tratare exceptie
        }
```

Exemplu:

```
   void XHandler(int test)
    {
      try
      {
         if(test==0) throw test;
         if(test==1) throw 'a';
         if(test==2) throw 123.23;
       }
      catch(...)
       {
         cout << "Caught one." << endl;
       }
    }

   int main(void)
    {
      cout << "Start: " << endl;
      XHandler(0);
      XHandler(1);
      XHandler(2);
      cout << "End";
     return 0;
    }
```

8. Captare exceptii explicite si exceptii generice

```
void XHandler(int test)
 {
   try
   {
      if(test==0) throw test;
      if(test==1) throw 'a';
      if(test==2) throw 123.23;
    }
   catch(int i)
    {
      cout << "Caught an integer." << endl;
    }
   catch(...)
    {
      cout << "Caught one." << endl;
    }
 }

int main(void)
 {
   cout << "Start: " << endl;
   XHandler(0);
   XHandler(1);
   XHandler(2);
   cout << "End";
  return 0;
 }
```

9. Restrictionarea exceptiilor

```
tip-returnat nume-functie (lista-argumente) throw (lista-tipuri)
      {
        //corpul functiei
      }
```

Pentru a restrictiona exceptiile pe care functiile noastre le pot lansa adaugam o clauza *throw* in definirea functiei.

Atunci cand apelam o functie cu clauza *throw* ea poate lansa doar acele tipuri pe care le are in lista-tipuri. Daca functia lanseaza orice alt tip de exceptie, programul se va termina in mod anormal.

Daca dorim ca functia sa nu lanseze nici o exceptie utilizam lista-tipuri vida.

```
void XHandler(int test) throw()
 {
   if(test==0)
     throw test;
   if(test==1)
     throw 'a';
   if(test==2)
     throw 123.23;
 }

int main(void)
 {
   cout << "Start: " << endl;
   try
    {
      XHandler(0);     // try passing 1 and 2 for different
                       // responses
     }
   catch(int i)
    {
      cout << "Caught an integer." << endl;
    }
   catch(char c)
    {
      cout << "Caught a character." << endl;
    }
   catch(double d)
    {
      cout << "Caught a double." << endl;
    }
   cout << "End ";
   return 0;
 }
```

Alt exemplu:

```
void XHandler(int test) throw(int, char, double)
 {
   if(test==0) throw test;
   if(test==1) throw 'a';
   if(test==2) throw 123.23;
 }
```

```
int main(void)
 {
   cout << "Start: " << endl;
   try {
      XHandler(0);                     // try passing 1 and 2
                                     // for different responses
    }
   catch(int i) {
      cout << "Caught an integer." << endl;
    }
   catch(char c) {
      cout << "Caught a character." << endl;
    }
   catch(double d) {
      cout << "Caught a double." << endl;
    }
   cout << "End ";
    return 0;
 }
```

Care e diferenta intre cele doua exemple?

10.Relansarea unei exceptii

```
void XHandler(void)
 {
   try {
      throw "hello";
    }
   catch(char *) {
      cout << "Caught char * inside XHandler." << endl;
      throw;
    }
 }


int main(void)
 {
   cout << "Start: " << endl;
   try {
      XHandler();
    }
   catch(char *)
   {
      cout << "Caught char * inside main." << endl;
    }
   cout << "End ";
   return 0;
 }
```

Probleme:

P1. Impartire la zero

```
void divide(double a,double b)
{
	try
	{
		if (!b) throw b; //vf daca divide la zero
		cout<<"rezultatul este  "<<a/b<<endl;
	}
	catch(double b)
	{
		cout<<"Nu se poate divide la zero "<<endl;
	}
}
int main()
{
	double i,j;
	do
	{
		cout<<"dati numaratorul"<<endl;
		cin>>i;
		cout<<"dati numitorul (0 pentru stop)"<<endl;
		cin>>j;
		divide(i,j);
	}
	while (i!=0);
	return 0;
}
```

P2. Studiati urmatoarea problema

```
#define MAXX 80
#define MAXY 25
class Point
{
public:
	class xZero {};
	class xOutOfScreenBounds {};
```

```
		Point(unsigned __x, unsigned __y)
		{
		x = __x;
		y = __y;
		}
		unsigned GetX()
		{
		return x;
		}
		unsigned GetY()
		{
		return y;
		}

	void SetX(unsigned __x)
	{
	if(__x > 0)
		if(__x <= MAXX)
			x = __x;
		else
		throw xOutOfScreenBounds();
	else
	throw xZero();
	}

	void SetY(unsigned __y)
	{
	if(__y > 0)
		if(__y <= MAXY)
			y = __y;
		else
		throw xOutOfScreenBounds();
	else
		throw xZero();
	}

	protected:
		int x, y;
	};

	int main()
	{
	Point p(1, 1);
	try {
	p.SetX(5); // CORRECT!
	// p.SetX(0); // throws an xZero exception
```

```
        cout << "p.x successfully set to " << p.GetX() << "."<<endl;
        // throws an xOutOfScreenBounds exception
        p.SetX(100);
        }

        catch(Point::xZero)
        {
        cout << "Zero value!\n";
        }

        catch(Point::xOutOfScreenBounds)
        {
        cout << "Out of screen bounds!\n";
        }

        catch(...)
        {
        cout << "Unknown exception!\n";
        }
        return 0;
        }
```

Deoarece exceptia este instantierea unei clase, prin derivare pot fi realizate adevarate ierarhii de tratare a exceptiilor.

Atentie! Exista posibilitatea de aparitie a unor exceptii chiar in cadrul codului de tratare a unei exceptii! Astfel de situatii trebuie evitate!

Tema:

Pb2, Pb3 din lab9-danielpop
(http://web.info.uvt.ro/~danielpop/oop/Lab9.pdf)