Manejo de Excepciones

En lenguajes C++ y Python

Esp. Ing. César Omar Aranda v2.0

Objetivos y contenidos

- Implementar código capaz de manejar excepciones
  - En lenguaje Python
  - En lenguaje C++
- Discutir aspectos básicos de diseño

Tipos de Errores

- Errores en tiempo de compilación
  - Errores de sintaxis
  - Errores de tipo
- Errores en tiempo de enlazado
- Errores en tiempo de ejecución
  - Detectados por computador (se cae!)
  - Detectados por biblioteca (excepciones)
  - Detectados por programa del usuario
- Errores de lógica del programa
  - Detectados por el programador

Ejemplo

```
int area(int largo,  int ancho)
{
return largo*ancho;
}
```

```
int s1 = area(7);
int s2 = area("siete", 2);
int s3 = area(7.5, 10);
int s4 = area(7, -10);
```

Detectados por el compilador

Alternativas para resolver area(7, -10)

- No hacer nada!
- La función que invoca a area( ) verifica los argumentos
  - No siempre se hace
- La función area( ) verifica los argumentos
  - Retorna un código de error
    - Solución no general
  - Modifica un indicador de estado de error
    - Solución no general
- Generar una excepción

Retornar un código de error

- Debe ser verificado al invocar función

```
int area(int largo, int ancho)
{
    if (largo <= 0 || ancho <= 0)
      return -1;        // Valor no válido
     return largo*ancho;
}

int x = area(a, b);
if (x < 0)
   cerr << "Error en área" << endl;
```

Modificar indicador de estado de error

- Debe ser verificado al invocar la función

```
int errno = 0; // Indicador de Estado

int area(int largo, int ancho)
{
    if (largo <= 0 || ancho <= 0)
      errno = -7; // Valor no válido
     return largo*ancho;
}

int x = area(a, b);
if (errno != 0)
   cerr << "Error en área" << endl;
```


Problemas de los enfoques anteriores

- Retorno de código de error
  - Qué pasa si programa no verifica el valor retornado por la función?
  - Qué pasa si función no tiene valores inválidos para usar como indicador de error?
- Indicador de error errno
  - Qué pasa si programa no verifica errno?
  - Cómo elegir un número apropiado que almacenar en errno?
  - Dónde se generó el error indicado por errno?

Excepciones en C++

- Método general para manejo de errores
- Pueden ser usadas para informar de cualquier tipo de error
- Separan detección de error (función llamada) con atención del error (función llamadora)
- Toda excepción generada vía **throw** debe ser atrapada en alguna parte del código usando
- **try . . . catch**

Ejemplo

```
10  int main() {
11      int *x;
12      int y = 1000000000;
13
14      try {
15          x = new int[y];
16          x[0] = 10;
17          cout << "Puntero: " << (void *) x << endl
18          delete[] x;
19      }
20      catch(std::bad_alloc&) {
21          cout << "Memoria insuficiente" << endl;
22      }
23
24      return 0;
25  }
```

```
Puntero: 0x49230008
```

```
Memoria insuficiente
```

Análisis de la cláusula throw

```
6   #include <iostream>
7   using namespace std;
8
9   int main() {
10      try {
11          throw 'x';
12      }
13      catch(char c) {
14          cout << "El valor de c es: " << c << endl;
15      }
16      catch(int n) {
17          cout << "El valor de n es: " << n << endl;
18      }
19
20      return 0;
21  }
```

```
El valor de c es: x
```


Programa del Area con Excepción

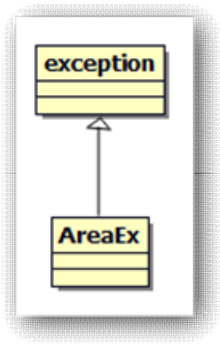
```
class ErrorEnArea{ };     // Objeto excepción

int area(int largo,  int ancho)
{
    if (largo <= 0 ||  ancho <= 0)
        throw ErrorEnArea();     // Debe ser atrapada en alguna parte del programa

    return largo*ancho;
}

// Programa principal
try {
    int x = area(a, b);
}catch(ErrorEnArea) {
    cerr << "Error en área" << endl;
}
```

Clase Exception

```
class exception {
 public:
  exception() throw() { }
  virtual ~exception() throw();
  virtual const char* what() const throw();
 };
```

- Sólo contiene 3 funciones
- La función "what" debe devolver una cadena que indique el motivo de la excepción
- Sirve de base para crear clases personalizadas de manejo de error.

Ejemplo: Definición de Excepción




```
9  class AreaEx : public exception {
10 public:
11     AreaEx(int mot) : exception(), motivo(mot) {
12     }
       const char* what() const throw ();
14 private:
15     int motivo;
16 };
17
   const char* AreaEx::what() const throw () {
19     switch (motivo) {
20         case 1:
21             return "Uno o más lados son negativos";
22         case 2:
23             return "Uno o más lados son cero";
24         case 3:
25             return "Uno o más valores no son números";
26     }
27 }
```

Ejemplo de Uso de la Clase Excepción

```
37  int main() {
38      float b;
39      float h;
40      float S;
41      cout << "Base?: " << endl;
42      cin >> b;
43      cout << "Altura?: " << endl;
44      cin >> h;
45      try {
46          S = superficie(b, h);
47          cout << "Superficie: " << S << endl;
48      } catch (AreaEx &e) {
49          cout << "Ocurrio una excepcion: " << e.what() << endl;
50      }
51
52      return 0;
53  }
```

```
29  float superficie(float b, float h) {
30      if (b == 0 || h == 0)
31          throw AreaEx(2);
32      if (b < 0 || h < 0)
33          throw AreaEx(1);
34      return b*h;
35  }
```

```
Base?:
-23
Altura?:
10
Ocurrio una excepcion: Uno o más lados son negativos
```


Python: Detección y Tratamiento

```
try:
    # aquí ponemos el código que puede lanzar excepciones.
    # es donde se realiza el procesamiento normal.
except IOError, TypeError, NameError:
    # entra aquí en caso de producirse una excepción de las listadas.
    # Se generan mensajes y/o algún procesamiento adicional.
except :
    # entra aquí al producirse una excepción de un tipo
    # no especificado en los except previos
else:
    # entra aquí en caso que no se produzca ninguna excepción.
finally:
   # en este bloque se coloca toda instrucción que haya que realizar, se haya producido una excepción o no
```

Python: Notificación y Propagación

```
try:
        # código de nivel exterior que puede levantar una excepción

        try:
            # código de nivel interior que levanta una excepción
            sentencia_X
    except EsteError:
    # si hay excepción se hace algo y/o se propaga
    raise
            except:
                # entra aquí al producirse una excepción de otro tipo

except EsteError:
        # realiza el tratamiento de la excepción
```


Python: Lanzamiento de una Excepción

La sintaxis de lanzamiento es:

```
raise Clase, instancia
raise instancia
```

Tiene que ser o una instancia de excepción, o una clase de excepción (o sea hereda de Exception).

Al pasar una clase de excepción, la misma resulta instanciada implícitamente mediante su constructor sin argumentos.

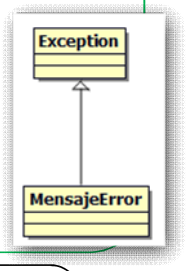
```
try:
    raise NameError('Fui_yo')
except NameError:
    print('Se lanzó una excepción !')
    raise
```

Salida en la consola

```
Se lanzó una excepción !

Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
NameError: Fui_Yo
```

Excepción Python definida por el usuario

```
class MensajeError(Exception):
  """Excepción para errores en el manejo de mensajes del servidor.
     Atributos:
        expresion:   detalle del evento que produce el error
        descripcion: explicación del error"""“

  def __init__(self, expresion, descripcion):
     self.expresion = expresion
     self. descripcion = descripción
```




```
try:
    raise MensajeError(“connect…”, “ip host”)
except MensajeError, e:
    print 'Ha saltado una excepción cuando ', e.expresion
```

Ha saltado una excepción cuando connect…