Introducción al Cómputo

Excepciones

Niveles de acceso a miembros

Exception handling

- C++ soporta “Exception handling” como una forma alternativa de manejar errores.
- Las excepciones son condiciones anormales (errores?) que un programa encuentra durante su ejecución.
- El mecanismo involucra 3 palabras claves:
  - **`try`**: define el bloque de código que directamente o indirectamente puede disparar una excepción.
  - **`catch`**: define el bloque de código que es ejecutado cuando una excepción particular es “atrapada”.
  - **`throw`**: utilizada para disparar una excepción.

Exception handling: ventajas

- **Separación de la manipulacion de errores del código normal**. Cuando se utiliza una manipulación de errores tradicional siempre hay bloques **`if`**…**`else`**… para manipular errores mezclados con el código normal. Esto hace el código menos legible y mantenible. Con bloques **`try`**…**`catch`**… el código de manipulación de errores está separado del código normal.

- **Selectividad de que excepciones manipular**. Las funciones pueden encontrar muchos tipos de excepciones pero elegir manipular solo algunos de ellos. Las excepciones disparadas y no manipuladas pueden ser atrapadas y manipuladas por la función llamadora directa o indirectamente.

- **Agrupamiento de tipos de error**. En C++ tanto tipos nativos como UDT pueden ser disparados como excepciones. Se podrían crear una jerarquía de tipos de excepciones.

Exception handling: ejemplos

ejem7_7.cpp

```
#include "icom_helpers.h"

int main(void)
{
   int x = -1;

   cout << "Antes del try\n";
   try {
      cout << "Dentro del try \n";
      if (x < 0) {
         throw x;
         cout << "Despues throw (nunca ejecutado)\n";
      }
   }
   catch (int x ) {
      cout << "Excepcion atrapada\n";
   }
   cout << "despues del catch \n";
   return 0;
}
```

```
$ ./ejem7_7
Antes del try
Dentro del try
Excepcion atrapada
despues del catch
```

Exception handling: ejemplos

```
int fun(int a) {
    if (a < 0) {
        throw a;
        cout << "Despues throw (nunca ejecutado)\n";
    }
    return -a;
}
int main(void) {
   int x = -1;
   cout << "Antes del try\n";
   try {
       cout << "Dentro del try \n";
       fun(x);
       cout << "Despues throw (nunca ejecutado)\n";
   } catch (int x ) {
      cout << "Excepcion atrapada\n";
   }
   cout << "despues del catch \n";
   return 0;
}
```

Exception handling: ejemplos

ejem7_8.cpp

```
#include "icom_helpers.h"

int main()
{
    try  {
       throw 10;
        //throw string("error!");
    }
    catch (string excep)  {
        cout << "atrape " << excep;
    }
    catch (...)  {
        cout << "Default Exception\n";
    }
    return 0;
}
```

```
$ ./ejem7_8
Default Exception
```

Exception handling: ejemplos

ejem7_9.cpp

```
#include "icom_helpers.h"

int main()
{
    try {
        try  {
            throw 20;
        }
        catch (int n) {
             cout << "primer manipulacion\n";
             throw;   //redisparando la misma exception
        }
    }
    catch (int n) {
        cout << "segunda manipulacion\n";
    }
    return 0;
}
```

```
$ ./ejem7_9
primer manipulacion
segunda manipulacion
```

Data Hiding

- Motivación:
  - Aseguramiento de un estado consistente
  - Exponer solo la funcionalidad deseada
  - Ocultar detalles internos
  - Flexibilidad para cambiar la representación interna sin necesidad de modificar las interfaces externas. Lo que hacía uso del tipo a partir de su interface no debería verse afectado.

Niveles de acceso

```
struct A {
      private:
            int X, Y;
      public:
            A(int x_, int y_) { X = x_; Y = y_; }
            void setX(int x_) { X = x_; }
            void setY(int y_) { Y = y_; }
            int getX()        { return X; }
            int getY()        { return Y; }
};
```

- Puede haber múltiples secciones públicas y privadas.

- Por defecto, el nivel de acceso para los UDT definidos a través de estructuras es “público”.

struct vs class

- Existe otra manera de hacer UDT: **`class`**

- Hacerlo a través de **`class`** es equivalente a hacerlo con **`struct`** con la salvedad que el nivel de acceso por defecto es privado.

```
class A {
      public:
            A(int x_, int y_)  { X = x_, Y = y_; }
            void setX(int x_) { X = x_; }
            void setY(int x_) { X = x_; }
            int getX()        { return X; }
            int getX()        { return Y; }
      private:
            int X, Y;
};
```