Estructuras de Datos

Abstracciones, Interfaces y Uso de Librerías en C++

Prof. Luis Garreta

Ingeniería de Sistemas y Computación
Pontificia Universidad Javeriana – Cali

7 de agosto de 2017

Estructura del código fuente en C++

- Un programa escrito en C++ se podría hacer en un único fichero de texto
- Sin embargo, un proyecto serio requiere que el código fuente de un programa se divida en varios ficheros para que sea manejable.
- Esto es lo que se conoce en C++ como:
  - **Interfaces o archivos de cabecera (Headers) y**
  - **Implementaciones o Archivos fuentes (Sources)**

Archivos de cabecera y archivos fuentes

- En C++ existen principalmente dos tipos de ficheros:
  - Los archivos de descripción de interface, tambien llamados archivos de encabezado (header) o archivos "include" (archivos .h, .hpp).
  - Los ficheros fuentes o archivos de implementación (archivos .c, .cpp, .cc (normalmente con extensión .cpp).

Archivos de Interface (Encabezado, header)

- Los archivos de encabezado (archivos .h, .hpp) contienen las declaraciones de constantes, variables y funciones de las que consta el módulo, así como llamadas a otros archivos de encabezado necesarios.

Archivos de Implementación (Fuentes, sources)

- En los archivos de implementación (archivos .c, .cpp, .cc) se implementa el código para las funciones del módulo declaradas en el archivo de encabezado.
- Durante el proceso de compilación se preprocesan los archivos de implementación y se añaden los archivos de encabezado, obteniendose un archivo objeto que será combinado con otros archivos objeto y al que se le agregarán las librerías para obtener un archivo ejecutable.

¿Por qué dividir el código fuente (I)?

- **Compilación más eficiente**. Si tu tienes un fichero fuente con 10.000 líneas de código y haces una pequeña modificación tendrás que recompilar las 10.000 líneas de código. Sin embargo, si tienes 10 ficheros de 1.000 líneas de código si haces una pequeña modificación en cualquiera de ellos solo deberás recompilar ese fichero ahorrando de recompilar 9.000 líneas de código fuente.
- **Organización**. Imagina que estas creando un videojuego y tienes un solo archivo llamado juego.cpp con todas las clases, estructuras y variables, deberás buscar a lo largo de todo el archivo si decides que quieres modifica una parte. Ahora piensa que has dividido tu juego en los archivos main.cpp graficos.cpp audio.cpp y entrada.cpp si quieres modificar alo relacionado con los gráficos sabes que debes buscar en el fichero graficos.cpp reduciendo la búsqueda considerablemente.

¿Por qué dividir el código fuente (II)?

- **Facilitar la reutilización**. Imagina que estás trabajando con fracciones en tu programa y decides crear una clase Fracción para el manejo de fracciones, la pruebas la testeas y compruebas que funciona. Si en el futuro desarrollas una aplicación que necesite el uso de fracciones solo deberás recuperar tu fichero .h y .cpp asociado a la clase fracción para implementar la funcionalidad en tu nuevo proyecto.
- **Compartir código en varios proyectos.** Este punto es similar al anterior, pero hay veces en las que más de un proyecto utilizan el mismo código, sería genial incluir el mismo el ambos proyectos y si se modifica este código automáticamente se modifique en ambos proyectos (es la esencia de las bibliotecas).
- **Dividir tareas entre programadores.** Si tenemos un equipo de programadores y queremos que trabajen a la vez en un mismo proyecto solo debemos asignar un fichero a cada uno y luego juntarlos, sin que se pisen unos a otros (Este es el principio de la Programación Orientada a Objetos y la división en clases independientes unas de otras).

Cómo dividir el código fuente de un programa C++

- La mayoría de las reglas a seguir son lógicas y un tanto arbitrarias, pero se presupone buen sentido al programador.
- Lo ideal es dividir por módulos según que hace que.
- Si tienes un juego que tiene una clase encargada de la entrada de datos, otra dfe los gráficos y otra del audio lo lógico es hacer la división en estos módulos.
- Una vez hecha la división lógica por funcionalidad queda ver que va en el archivo de cabecera y que en el archivo de código fuente.

Que va en el archivo de cabecera?

Los archivos de cabecera suelen incluir todos o algunos de los siguientes elementos:

- Definición de estructuras y clases.
- Definición de tipos (typedef).
- Prototipos de funciones.
- Variables Globales (ver más adelante).
- Constantes
- Macros #define.
- Directivas #pragma.

Ejemplo 1 - Standard Library: stdlib.h

```
#ifndef _STDIO_H
#if !defined __need_FILE && !defined __need___FILE
# define _STDIO_H 1 # include <features.h>
__BEGIN_DECLS
# define __need_size_t
# define __need_NULL # include <stddef.h>
# include <bits/types.h>
# define __need_FILE # define __need___FILE
...
...
/* Write formatted output to stdout.
   This function is a possible cancellation point and therefore not    marked
        with __THROW.  */
extern int printf (const char *__restrict __format, ...); /* Write formatted
     output to S.  */
extern int sprintf (char *__restrict __s, const char *__restrict __format, ...)
     __THROWNL;
...
...
 __END_DECLS
#endif /* <stdio.h> included.  */
#endif /* !_STDIO_H */
```

Ejemplo 2: Header MiCalculadora.h

```
// archivo: MiCalculadora.h

#ifndef __MICALCULADORA_H
#define __MICALCULADORA_H

#include <math.h>
#define PI 3.1415926535897932

float calcCoseno(float Angulo) ;

#endif /*__MICALCULADORA_H*/
```

Ejemplo 2: Implementación MiCalculadora.cpp

```
// archivo: MyHeader.cpp

#include "MiCalculadora.h"

float calcCoseno(float Angulo)
{

    float result;

    result=(float)cos(Angulo*PI/180.0f);

    return result;

}
```

Ejemplo2: Llamado main.cpp

```
// archivo main.cpp:

# include <stdio.h>
# include "MyHeader.h"

void main( void ) {
    float Ang = 45.0f;
    printf( "El coseno de %f grados es %.3f\n", Ang, calcCoseno(Ang));
}
```

Tarea1: Completar la Calculadora

- Para el ejemplo de la calculadora, completarlo con 10 funciones más (ejemplo: calcSeno, calcExponencial, calcLogaritmo, etc.)
- Para la prueba realizar un menú en el main (usando switch) que ejecute cada función.

Ejemplo1: Uso de Librerías iostream y math

El siguiente programa usa cuatro funciones de tres librerías distintas: *iostream y math*

```
// Ejemplo de uso de la librerias iostream y math
#include <iostream>  // cin, cout
#include <math.h>    // pow

using namespace std; // strings

int main () {
  int valorBase;
  int valorExponente;

  // Lee desde el teclado
  cout << "Ingrese valor para la base: "  ;
  cin >>  valorBase;

  cout << "Ingrese valor para el exponente: " ;
  cin >>  valorExponente;

  int valorPotencia = pow (valorBase, valorExponente);
  cout << "La potencia es: " << valorPotencia << endl;

  return 0;
}
```

Ejemplo2: Uso de Librerías iostream y string

El siguiente programa usa cuatro funciones de tres librerías distintas: *iostream y string*

```
// Ejemplo de uso de la librerias iostream y string (c++)
#include <iostream>  // cin, cout, cinline

using namespace std; // string (c++)

int main () {
  string usuario;
  string clave;
  string claveInterna="hola1233";

  cout << "Ingrese usuario y clave: ";
  cin >> usuario >> clave;

  // Longitud y Comparación (==, !=)
  if (clave.length () < 5 && clave != claveInterna) {
    cout << "Clave Incorrecta";
    return -1;
  }

  // Concatenacion
  string userKey = usuario + clave;
  cout << "Usuario " << userKey << " logeado" << endl;

  return 0;
}
```