Introducción a la Algorítmica y Programación (3300)

Prof. Ariel Ferreira Szpiniak - aferreira@exa.unrc.edu.ar
Departamento de Computación
Facultad de Cs. Exactas, Fco-Qcas y Naturales
Universidad Nacional de Río Cuarto

Teoría 12

Abstracción, Modularización, Bibliotecas

Abstracción

Como hemos visto hasta ahora a los largo del curso, una de las herramienta fundamentales para construir programas es la **Abstracción**, que permite dividir un problema grande en pequeños problemas de más fácil solución. La solución del problema original es la composición de las pequeñas soluciones:

- **Abstraer**: separar las cualidades de un objeto para considerarlas aisladamente o para considerar el mismo objeto en su pura esencia o noción.
- **Abstracto**: Que significa alguna cualidad con exclusión del sujeto.

Las abstracciones que nosotros hemos analizado son: ***Acciones y Funciones. Más adelante analizaremos Tipos Abstractos de Datos (TAD).***

Abstracción

En los lenguajes de programación, el concepto fundamental que se usa para lograr la abstracción es el de **Módulo**, que es una sección de un programa bien construida, con un fin específico, y que puede ser reutilizado.

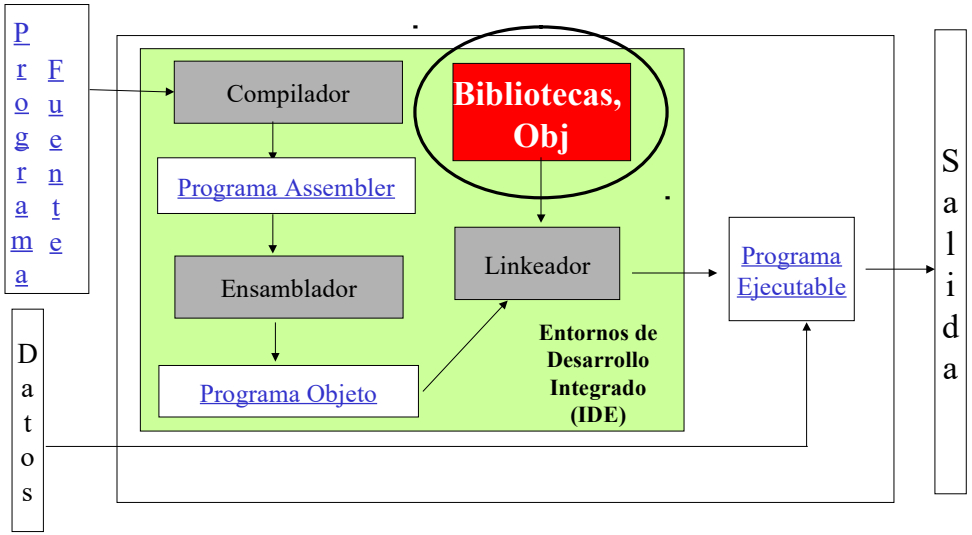
Una forma de profundizar el concepto de abstracción es mediante la construcción de módulos que sean independientes de un algoritmo o programa en particular, es decir, que no formen parte indisoluble del mismo.

Abstracción

Tradicionalmente, este tipo de módulos se han utilizado para lograr la **Compilación separada** de un programa, la que permite recompilar únicamente las secciones del programa que han sufrido cambios. Esta técnica hace posible crear bibliotecas en las que el código fuente no tiene porqué ser accesible por el programador que hace uso de esa biblioteca.

En **C** existe una manera de generar módulos, independientes de cualquier programa, mediante el uso de Bibliotecas o Librerías. Por ejemplo, **stdio.h** que proporciona entrada/salida y **string.h** para manipulación de cadenas de caracteres, las cuales usamos en nuestros programas pero no tenemos acceso a su implementación.

Abstracción



Biblioteca o Librería - Concepto

- Una biblioteca o librería es un conjunto de constantes, tipos de datos variables, acciones y funciones.
- Son independientes.
- Una librería puede utilizar otras librerías.
- Tienen una estructura similar a los algoritmos
- Acentúa el carácter estructurado y modular
- Además, las librerías son reusables, con lo cual se puede escribir una librería ahora y usarla en cuantos algoritmos sea necesario.

Biblioteca o Librería Estructura

Una biblioteca está constituida por las siguientes secciones:

– Cabecera
– Sección **Interface** (o sección pública)
– Sección **Implementación** (o sección privada)
– Sección de inicialización (opcional)

Biblioteca o Librería Estructura

```
Biblioteca <nombre>
Interfaz
Usa <lista de bibliotecas> {opcional}
{declaraciones públicas}
Implementación
{declaraciones privadas}
{definición de procedimientos y funciones}
Inicio
 <secuencia de acciones> {opcional}
Fin
```

Bibliotecas en C - Concepto

La biblioteca o librería (library) nos permite el uso de funciones en un programa sin la necesidad de escribir su código en nuestro programa, únicamente llamando a la biblioteca.
C posee una recopilación estandarizada por la Organización Internacional para la Estandarización (ISO). ANSI C (adoptado por ISO) consta de 24 archivos **cabecera**, denominados **.h**, que pueden ser incluidos en un programa con una simple directiva: **#include**.
Cada **cabecera** contiene la declaración de una o más funciones, tipos de datos y macros.

- Su estructura es similar a un programa, pero sin main.
- Las bibliotecas pueden ser predefinidas (las 24 estándar) o definidas por el programador.

Bibliotecas en C - Estándar

<assert.h>: Contiene la macro assert (aserción), utilizada para detectar errores lógicos y otros tipos de fallos en la depuración de un programa.
<complex.h>: Conjunto de funciones para manipular números complejos.
<ctype.h>: Contiene funciones para clasificar caracteres según sus tipos o para convertir entre mayúsculas y minúsculas independientemente del conjunto de caracteres (típicamente ASCII o alguna de sus extensiones).
<errno.h>: Para analizar los códigos de error devueltos por las funciones de biblioteca.
<fenv.h>: Para controlar entornos en coma flotante.

Bibliotecas en C - Estándar

<float.h>: Contiene la definición de constantes que especifican ciertas propiedades de la biblioteca de coma flotante.
<inttypes.h>: Para operaciones de conversión con precisión entre tipos enteros.
<iso646.h>: Para utilizar los conjuntos de caracteres ISO 646.
<limits.h>: Contiene la definición de constantes que especifican ciertas propiedades de los tipos enteros, como rango de valores que se pueden representar (_MIN, _MAX).
<locale.h>: Para la función setlocale() y las constantes relacionadas. Se utiliza para seleccionar el entorno local apropiado (configuración regional).
<math.h>: Contiene las funciones matemáticas comunes.

Bibliotecas en C - Estándar

<setjmp.h>: Declara las macros setjmp y longjmp para proporcionar saltos de flujo de control de programa no locales.
<signal.h>: Para controlar algunas situaciones excepcionales como la división por cero.
<stdarg.h>: Posibilita el acceso a una cantidad variable de argumentos pasados a una función.
<stdbool.h>: Para el tipo booleano.
<stdint.h>: Para definir varios tipos enteros.
<stddef.h>: Para definir varios tipos de macros de utilidad.
<stdio.h>: Proporciona el núcleo de las capacidades de entrada/salida del lenguaje C.

Bibliotecas en C - Estándar

<stdlib.h>: Para realizar ciertas operaciones como conversión de tipos, generación de números pseudo-aleatorios, gestión de memoria dinámica, control de procesos, funciones de entorno, de ordenamiento y búsqueda.

<string.h>: Para manipulación de cadenas de caracteres.

<tgmath.h>: Contiene funcionalidades matemáticas de tipo genérico (type-generic).

<time.h>: Para tratamiento y conversión entre formatos de fecha y hora.

<wchar.h>: Para manipular flujos de datos anchos y varias clases de cadenas de caracteres anchos (2 o más bytes por carácter), necesario para caracteres de diferentes idiomas.

<wctype.h> Para clasificar caracteres anchos.

Bibliotecas en C - Estructura

¿Cómo crear nuevas bibliotecas?

1) Se escriben solo las declaraciones de las funciones. Se guarda con extensión .h.

Ejemplo: milibreria.h

2) Se escribe un programa con las funciones, tipos de datos y macros. No se coloca main(). Se incluye el .h anterior (#include <milibreria.h>). Se guarda con el mismo nombre que el archivo anterior, pero con extensión .c

Ejemplo: milibreria.c

3) En el programa que utiliza la bilioteca se la debe incluir, junto al resto de las bibliotecas necesarias:

Ejemplo: #include <stdio.h>

#include <milibreria.h>

Bibliotecas en C - Estructura

Ejemplo

1) Se escriben solo las declaraciones de las funciones. Se guarda con extensión .h.

milibreria.h:

```
int multiplica(int a, int a);
int suma(int a, int b);
int resta(int a, int b);
```


Bibliotecas en C - Estructura

Ejemplo

2) Se escribe un programa con las funciones, tipos de datos y macros:

milibreria.c:

```
#include <milibreria.h>
int multiplica(int a, int a){
   return(a*b);
}
int suma(int a, int b){
   return(A+B);
}
int resta(int a, int b){
   return(a-b);
}
```

Bibliotecas en C - Estructura

Ejemplo

3) En el programa que utiliza la bilioteca se la debe incluir, junto al resto de las bibliotecas necesarias:

ejemploUsoMiLibreria.c:

```
#include <stdio.h>
#include <milibreria.h>
int main(void)
{
    int x,y;
    scanf("%d %d",&x,&y);
    printf("x*y=%d \n",multiplica(x,y));
    printf("x+y=%d \n",suma(x,y));
    printf("x-y=%d \n",resta(x,y));
    return 0;
}
```


Bibliotecas en C - Estructura

Ejemplo arithmetic

1) Se escriben solo las declaraciones de las funciones. Se guarda con extensión .h.

arithmetic.h:

```
int is_prime( int n );
int division( int dividend, int divisor, int *
quotient );
int factorial( int n );
```


Bibliotecas en C - Estructura

- **Ejemplo arithmetic**

2) Se escribe un programa con las funciones, tipos de datos y macros:

arithmetic.c:

```
#include <arithmetic.h>
int is_prime( int n )
{
     int result = 1;
     for ( int i = 2; i < n; i++ ){
          if ( !(n % i) ){
               result = 0;
          }
     };
     return result;
}
```


Bibliotecas en C - Estructura

```
int division( int dividend, int divisor, int
* quotient )
{
    int remainder;
    remainder = dividend;
    *quotient  = 0;
    while ( remainder >= divisor ){
         remainder -= divisor;
         (*quotient)++;
    };
    return remainder;
}
```

Bibliotecas en C - Estructura

```
int factorial( int n )
{
    int result  = 1;
    if ( !n )
        return result;
    do
    {
        result *= n--;
    }
    while ( n > 0 );
    return result;
}
```


Bibliotecas en C - Estructura

Ejemplo arithmetic

3) En el programa que utiliza la bilioteca se la debe incluir, junto al resto de las bibliotecas necesarias:

ejemploUsoArithmetic.c:

```
#include <stdio.h>
#include <arithmetic.h>
int main( void )
{
        int q = 0;
        printf( "is %i prime? %i\n", 13, is_prime(13));
        printf( "is %i prime? %i\n", 12, is_prime(12));
        printf( "%i! = %i\n", 0, factorial(0));
        printf( "%i! = %i\n", 7, factorial(7));
        printf( "%i (remainder) + %i (divisor) * %i (quotient) = %i
(dividend)\n", division(17,3,&q),3,q,17);
        printf( "%i (remainder) + %i (divisor) * %i (quotient) = %i
(dividend)\n", division(12,4,&q),4,q,12);
        return 0;
}
```