Programación I
Guía 1- Primeros ejercicios C++

2011

Introducción a C++

Objetivo de la materia

Entender la notación de la Orientación a Objetos. Introducir el uso de un ambiente **UNIX**. Brindar nociones generales de Algoritmia y estructura de Datos. Introducir los conceptos fundamentales de Orientación a Objetos.

Objetivo de la Guía

Conceptos Fundamentales de C++.

Bibliografía

http://cppreference.com
http://cplusplus.com/

Tipos de datos

C define cinco tipos de datos:

```
void
int
float
double
char
```

C++ agrega dos tipos adicionales:

```
boolean
wchar_t
```

Además algunos admiten los modificadores:

```
signed,
unsigned,
short,
long
```

Y si se desea que no se pueda modificar la variable se usa *const*

El tamaño (en bytes) puede explorarse con el operador `sizeof`. No todas las combinaciones son válidas. Un programa para explorar algunos tamaños es:

```
1. #include <iostream>
2. using namespace std;
3. // imprime la longitud de algunos tipos
4.
5. int main(){
```

```
6.  long int li;
7.  cout<<"long integer "<<sizeof li<<endl;
8.  cout <<"unsigned "<<sizeof (unsigned)<<endl;
9.  cout <<"char "<<sizeof (char)<<endl;
10. cout <<"wchar_t "<<sizeof (wchar_t)<<endl;
11.}
12.
```

Conteste usando la Internet:

1. Escriba un ejemplo de un literal de cada tipo.
2. ¿Porqué en la línea 7 se usa `sizeof li` y en la 8 `sizeof (unsigned)` (con los paréntesis)?
3. ¿Cuáles son las combinaciones válidas de modificadores y tipo? ¿Cuáles son los tamaños de cada uno de los posibles tipos?
4. ¿Para qué se usa el `wchar_t`?

Entrada /Salida

Si bien habrá una guía completa dedicada al tema, como introducción la entrada y salida se hace usando los operadores << (extracción) e >> (inyección) que operan sobre un *stream* (corriente) de caracteres y alguna expresión. Como resultado, los operadores << e >> devuelven un *stream*.

Un *stream* de caracteres es una secuencia de caracteres que puede estar en la consola (terminal), en un archivo o en algún dispositivo de comunicación.

Los operadores << e >> se encargan ellos de todas las operaciones de edición necesarias.

Hay definidos en el *namespace* (Espacio de nombres) *std* tres *streams:* una de entrada, la *standard input* o *stdin, llamada* ***cin***; y dos de salida: la *standard output* o *stdout* llamada ***cout*** y la *standard error* o *stderr* llamada ***cerr.***

Estos *streams* coinciden con lo usado en Linux/Unix y Xp/Vista. En estos sistemas operativos estos *streams* se pueden redirigir a archivos o dispositivos usando los caracteres < , > y 2>.

Un ejemplo rápido del uso de los *streams:*

```
1. include <iostream>
2. using namespace std;
3. // Entrada salida simple con streams
4.
5. int main(){
6.  char  letras [10];
7.  cout<<"Esta es la salida standard"<<endl;
8.  cerr<<"Esta es la std error"<<endl;
9.  cin>>letras;
10. cout<<"leido "<<letras<<endl;
11.}
12.
```

1. Pruebe las redirecciones en el programa anterior.
2. ¿Como se detecta el fin de archivo de entrada?

Para leer de un archivo:

```
1. #include <iostream>
2. #include <fstream>
3. using namespace std;
3. // Entrada salida simple con archivos
4.
5. int main(int argc, char * argv []){
6.  char  letras [10];
7.  if (argc < 3) {
8.      cerr <<"uso: "<<argv[0]<<" <entrada>
  <salida>"<<endl;
9.      exit (1);
10.     }
11. ifstream entra (argv[1]);
12. if (!entra) {
13.     cerr<<"Error en el archivo de entrada
  "<<argv[1]<<endl;
14.     exit (1);
15.     }
16. ofstream sale (argv[2]);
17. while (entra >>letras){
18.     sale<<letras;
19.     cout<<"Leido y copiado <"<<letras<<">"<<endl;
20.     }
21.}
22.
```

1. Corra el programa y observe las salidas y los errores si el archivo de entrada no existe.

Mas sobre el tema en la guía de archivos

Pasaje de parámetros

Se llama **parámetro formal** al identificador declarado en la cabeza de la subrutina. Se llama **argumento efectivo** al identificador con que se llama a la subrutina. El pasaje de parámetros es el mecanismo para reemplazar parámetro formal por argumento efectivo.

En C++ hay dos formas de pasar parámetros a una función:

a) **Por Valor**. De esta forma es imposible modificar el valor del argumento efectivo de forma tal que esa modificación tenga efecto en el programa llamador. Es lo standard en C

b) **Por Referencia**. Se lo indica con un & después del tipo del parámetro. En este caso, la función trabaja sobre el mismo área de memoria que ocupa el argumento efectivo. Esto implica que la función puede cambiar el valor del argumento efectivo y esa modificación se mantiene al retornar al programa llamador.

Por compatibilidad con C los *arrays* se pasan siempre por referencia.

Notar que el pasaje por referencia NO ES LO MISMO que pasar un puntero. Al pasar un puntero (estilo C), todas las operaciones de dereferencia están a cargo del programador. Al pasar por referencia, el argumento formal se usa como cualquier variable.

A continuación se desarrolla un ejemplo:

```
1. #include <iostream>
```

```
2. using namespace std;
3. // pasaje de argumentos
4.
5. typedef struct {
6.      int nro1;
7.      int nro2; } duo;
8. /* es la forma moderna de escribir
9.      struct duo {
10.          int nro1;
11.          int nro2; };
12.*/
13.void asigna ( duo& uno,  duo dos){ // pasa por
   referencia
14.   uno.nro1=dos.nro1;
15.   uno.nro2=dos.nro2;
16.}
17.void imp( duo x){
18.  cout<<"("<<x.nro1<<", "<<x.nro2<<")";
19.}
20.int main(){
21.  duo a, b;
22.  a.nro1=10; a.nro2=20;
23.  b.nro1=0;  b.nro2=0;
24.  cout<<"a=";
25.  imp (a); cout<<endl;
26.  asigna (b,a);
27.  cout<<"b=";
28.  imp (b); cout<<endl;
29.}
30.
```

1. Cambiar la línea 13 para pasar la estructura por valor y observar la diferencia.

En este ejemplo se usó la directiva *typedef* que lo que hace es introducir un nuevo tipo, definido por el usuario. En este caso *duo* es una *struct* con dos campos *int*.

Para evitar caer en el manejo tipo C de los arreglos (y de paso para evitar los pproblemas de mantener una variable que indique hasta donde está lleno), los arreglos se manejarán con:

```
typedef struct t_arre{
        int libre;
        int a[20];
    };
```

Donde *libre* indica la primer posición libre del arreglo. Debe inicializarse en cero como:

```
t_arre aa;
aa.libre=0;
```

Ejercicios de Programación

Para no olvidar que es una materia de programación acá van algunos ejercicios. Use la Internet para encontrar las definiciones que no conoce. Documente en su programa el URL donde encontró lo pedido.

1) Una función que encuentre un número en un arreglo. Escriba el programa que cargue el arreglo y que lo pruebe *(driver y tester)*.
2) Escriba un programa que copie un arreglo en otro, pero invirtiendo el orden de sus elementos.
3) Escriba un programa que lea caracteres de la entrada *standard* y haga una tabla de aparición y frecuencia de las distintas letras. Investigue su uso. Recomiendo visitar *http://en.wikipedia.org/wiki/Letter_frequency*
4) Busque en Internet y programe la función de *Hailstone*. ¿Cuál es la conjetura de *Hailstone*? Haga un programa que pida un numero entero al operador y compruebe si se cumple o no.
5) ¿Que es un número perfecto? Escriba una función que indique si un número es perfecto.
6) Escriba una función que indique si un número es capicúa.
7) Escriba una función que reciba dos números, el segundo de ellos de un solo dígito y devuelva cuantas veces se encuentra ese dígito en el primer número.

Problemas Interesantes

1) ¿Que es un dígito verificador de *Luhn*?. Escriba una función para calcularlo. Escriba otra que acepte un numero e indique si su último número es o no un *Luhn* . (para que esto sea útil, deberá trabajar con números largos, 16 cifras al menos; así que le aconsejo usar un arreglo con una cifra en cada posición.
2) Hacer un programa que lea un archivo de texto e indique en que idioma está escrito (Alemán, Francés, Inglés, Español o Italiano)
3) Hacer un programa que reciba dos números que determinan un rango (p. ej 23 y 194) e imprima, para cada número de ese intervalo, en cuantos pasos se cumple la conjetura de *Hailstone.* **Este programa se tomó de un concurso de *performance*, por lo tanto su solución deberá optimizar de alguna forma el cálculo.**