



Carátula para entrega de prácticas

Facultad de Ingeniería

Laboratorio de docencia

Laboratorios de computación
salas A y B

Profesor: Alejandro Esteban Pimentel Alarcon

Asignatura: Fundamentos de programación.

Grupo: 3

No de Práctica(s): 12

Integrante(s): Crail Ávila Regina
Ortiz Garcia Cesar Alan

*No. de Equipo de cómputo
empleado:* 11

No. de Lista o Brigada: 8973
9070

Semestre: 20-21

Fecha de entrega: 3/Noviembre/19

Observaciones:

CALIFICACIÓN: _____

Objetivo

Elaborar programas en C donde la solución del problema se divida en funciones. Distinguir lo que es el prototipo o firma de una función y la implementación de ella, así como manipular parámetros tanto en la función principal como en otras.

LOS PROTOTIPOS

Examinemos el archivo **23PROTIPO1.CPP**

```
1
2  #include <iostream.h>
3
4  void relleno(int alas, float pies, char ojos);
5
6  main()
7  {
8
9      int brazo = 2;
10
11     float pie = 1000.0;
12
13     char ojo = 2;
14
15     relleno(3, 12.0, 4);
16
17     relleno(brazo, pie, ojo);
18 }
19
20 void relleno(int alas, float pies, char ojos)
21 {
22
23
24     cout << "Hay " << alas << " alas." << "\n";
25
26     cout << "Hay " << pies << " pies." << "\n";
27
28     cout << "Hay " << (int)ojos << " ojos." << "\n\n";
29
```

Un prototipo es un modelo limitado de una entidad más completa que vendrá luego. En este caso, la función "relleno" es la entidad completa que vendrá luego y el prototipo se ilustra en la línea 3. El **prototipo proporciona información sobre el tipo devuelto o producido por la función, así como sobre sus parámetros**. Se utiliza para verificar los llamados a la función, ya que controla el número y el tipo de los parámetros, comprobando si son los apropiados. En nuestro ejemplo, cada llamado a la función llamada "relleno()" debe tener exactamente tres parámetros o el compilador dará un mensaje de error.

Además del número correcto de parámetros, los tipos deben ser compatibles o el compilador emitirá un mensaje de error. El aviso sobre el que trabaja el compilador se ve en las líneas 11 y 12, la comprobación de tipo puede hacerse en base al prototipo de la línea 3 aunque la función misma no está definida aún.

Si el prototipo no es dado, el número de parámetros no se verificará, ni los tipos de los parámetros. Aún cuando se tiene el número equivocado de parámetros, se conseguirá una compilación y vinculación aparentemente buenas, pero el programa puede hacer algunas cosas extrañas cuando se ejecute. *jaja*

Para escribir el prototipo, simplemente se copia la cabecera de la función al comenzar el programa y se añade un punto y coma al final como una señal al compilador de que ésta no es una función sino un prototipo. Los nombres de las variables dadas en el prototipo son optativos y actúan meramente como comentarios al lector del programa, ya que ellos son completamente ignorados por el recopilador. Si se reemplaza la variable "alas" en la línea 3 con cualquier otro nombre no habría ninguna diferencia en la compilación.

Aunque deseamos usar el tipo char para "ojos" en la función, queremos usarlo más bien como un número y no como un caracter.

El entero en la línea 19 requiere que se exija la impresión del valor numérico como un caracter ASCII. El próximo ejemplo de programa es similar pero sin el int.

LOS TIPOS COMPATIBLES

Un tipo compatible es cualquier tipo simple que puede convertirse a otro en una manera significativa. Por ejemplo, si se usó un entero como el parámetro real y la función esperaba un tipo float como parámetro formal, el sistema hará la conversión automáticamente. También se puede cambiar un float a char, o un char a int. Aunque hay reglas definitivas de conversión que se deben seguir.

Si una función espera un parámetro entero, y se le proporciona uno real, la conversión no se realiza porque los valores son completamente diferentes.

La compatibilidad de tipos discutida anteriormente en este manual se aplica igualmente a la compatibilidad de tipos cuando se hacen llamados a una función. Además, el tipo especificado como el tipo de regreso debe ser compatible con el regreso esperado en la declaración de llamada, o el recopilador emitirá una advertencia.

¿COMO TRABAJA UN PROTOTIPO?

Esta es una oportunidad para tratar prototipos y ver cómo trabajan y qué tipos de mensajes de error se producen cuando se hacen cosas equivocadas. Cambiemos los parámetros reales en la línea 11 para leer (12.2, 13, 12345) y ver qué dice el compilador sobre este cambio. Probablemente no dirá nada porque son todos compatibles.

Si se cambian los valores por (12.0, 13), el compilador emitirá una advertencia o error porque no hay argumentos suficientes.

También debería producirse un mensaje de error si se cambia uno de los parámetros en la línea 12 escribiendo un signo ampersand frente a uno de los nombres de las variables. Finalmente, cambiando la primera palabra en la línea 3 entre void e int se puede ver otro mensaje de error. En la línea 15 se requerirá que la función concuerde con un prototipo, pero el compilador no lo encontrará.

UN POCO MAS SOBRE PROTOTIPOS

Veamos el ejemplo de programa llamado **24PROTIPO2.CPP**

```
1
2   #include <iostream.h>
3
4   void relleno(int, float, char);
5
6   main()
7   {
8
9
10    int brazo = 2;
11
12    float pie = 1000.0;
13
14    char ojo = 65;
15
16    relleno(3, 12.0, 67);
17
18    relleno(brazo, pie, ojo);
19  }
20
21  void relleno (int alas,      // Número de alas
22               float pies,    // Número de pies
23               char ojos)     // Número de ojos
24  {
25
26    cout << "Hay " << alas << " alas." << "\n";
27
28    cout << "Hay " << pies << " pies." << "\n";
29
30    cout << "Hay " << ojos << " ojos." << "\n\n";
31
32  }
33
34
35
```

Este programa es idéntico al primero, con excepción de unos pequeños cambios.

Los nombres de las variables se han omitido en el prototipo en la línea 3 solamente como una ilustración de que ellos se interpretan como comentarios por el compilador de C++.

La cabecera de la función tiene un formato diferente para permitir un comentario al final de los parámetros reales. Esto debería hacer la cabecera de función un poco más explicativa. Sin embargo, debemos recordar que los comentarios no deberían usarse, reemplazándolos con una selección cuidadosa de los nombres de variables.

¿QUE COSTO TIENE UN PROTOTIPO?

El prototipo no cuesta absolutamente nada en lo que concierne a la velocidad o tiempo de ejecución, ya que demora una cantidad insignificante de tiempo en la comprobación que el compilador debe hacer.

El único precio que se debe pagar para usar un prototipo es el tamaño extra de los archivos de fuente de los mismos, y el tiempo extra que el compilador tarda en leer los prototipos durante el proceso de compilación, pero ambos costos son insignificantes.

PASE POR REFERENCIA

Examinemos el archivo **25PASREF.CPP** para un ejemplo de un pase por referencia:

```
1
2  #include <iostream.h>
3
4  #include <stdio.h>
5
6  void violin(int in1, int &in2);
7
8  main()
9  {
10
11     int cont = 7, ind = 12;
12
13     cout << "Los valores son ";
14
15     printf("%3d %3d\n", cont, ind);
16
17     violin(cont, ind);
18
19     printf("%3d %3d\n", cont, ind);    cout << "Los valores son ";
20
21 }
22
23 void violin(int in1, int &in2)
24 {
25     in1 = in1 + 100;
26
27     in2 = in2 + 100;
28
29     cout << "Los valores son ";
30
31     printf("%3d %3d\n", in1, in2);
32 }
33
```

Por predefinición, cuando los programas introducen un parámetro a una función, C++ hace una copia del valor de este parámetro y lo pone dentro de un lugar de memoria temporal llamado *pila*. Entonces la función utiliza una copia del valor. Cuando se termina de ejecutar la función, C++ descarta el contenido de la pila y cualquier cambio que la función haya hecho a la copia.

Para cambiar el valor de un parámetro, la función debe saber la dirección de memoria del parámetro. El operador de dirección & se utiliza en los programas para indicarle a la función cuál es la dirección del parámetro para que pueda realizar los cambios deseados.

Este procedimiento es un **pase por referencia**, permite el pasaje de una variable a una función y retornar los cambios hechos en la función al programa principal.

Observemos el prototipo en la línea 4, donde la segunda variable tiene un signo & delante del nombre de la variable. Este signo indica al compilador que esa variable actuará como un puntero (indicador) a la variable real del programa principal que se usará en la función.

En la función misma, en las líneas 21 a 24, la variable "in2" se usa como cualquier otra, pero pasándola desde el programa principal a la función, no usa una copia de ella. En efecto, el nombre "in2" es un sinónimo para la variable

"ind" en el programa principal, hace referencia a ella. En cambio, la otra variable llamada "in1" se trata como cualquier otra variable normal en C.

Cuando el programa se compila y ejecuta, se observa que la variable "in1" cambia en la función pero vuelve a su valor original cuando se retorna al programa principal (sigue valiendo 7). Sin embargo, la segunda variable "in2", cambia en la función y el nuevo valor se refleja en la variable del programa principal (su valor cambia de 12 a 112), lo que se puede ver cuando los valores se imprimen por pantalla (línea 16).

Si se prefiere omitir los nombres de variables en los prototipos, se escribiría el prototipo como se indica a continuación: void violín (int, int&);

PARÁMETROS POR DEFECTO

Veamos el programa **26PARAMET.CPP** como un ejemplo del uso de parámetros por defecto:

```
1
2  #include <iostream.h>
3
4  #include <stdio.h>
5
6  int volumen(int largo, int ancho = 2, int alto = 3);
7
8  main()
9  {
10
11  int x = 10, y = 12, z = 15;
12
13  cout << " Algunos datos de la caja son " << volumen(x, y, z) << "\n";
14
15  cout << " Algunos datos de la caja son " << volumen(x, y) << "\n";
16
17  cout << " Algunos datos de la caja son " << volumen(x) << "\n";
18
19  cout << volumen(x, 7) << "\n";   cout << " Algunos datos de la caja son ";
20
21  cout << " Algunos datos de la caja son ";
22
23  cout << volumen(5, 5, 5) << "\n";
24  }
25
26  int volumen(int largo, int ancho, int alto)
27  {
28
29  printf("%4d %4d %4d   ", largo, ancho, alto);
30
31  return largo * ancho * alto;
32
33  }
34
35
```

Este programa realmente se ve extraño ya que contiene algunos valores por defecto para los parámetros en el prototipo, pero faltan valores muy útiles como veremos luego.

Este prototipo dice que el primer parámetro llamado "largo" debe darse para cada llamado de esta función porque no fue proporcionado ningún valor por defecto.

El segundo parámetro llamado "ancho", sin embargo, no requiere que se especifique para cada llamado, ya que si no es especificado, se usará el valor 2 para la variable "ancho" dentro de la función. Asimismo, el tercer parámetro es optativo, y si no es especificado, el valor 3 se usará para la "altura" dentro de la función.

En la línea 10 de este programa se especifican los tres parámetros, así que no hay nada de particular en este llamado a la función.

Sin embargo, en la línea 11, se especifican solamente dos valores, por lo tanto se usará el valor por defecto para el tercer parámetro y el sistema trabajará como si se hubiera llamado a la función con volumen(x, y, 3), ya que el valor por defecto para el tercer parámetro es 3. En la línea 12, se especifica únicamente un parámetro, que se usará para el primer parámetro formal, y faltarán los otros dos. El sistema considerará el llamado como volumen(x, 2, 3).

Notaremos que la salida por pantalla de estas tres de líneas se revierte. Esto se explicará luego.

Hay unas reglas que deben ser obvias pero serán revisadas de cualquier manera. Una vez que a un parámetro se le da un valor por defecto o de autoselección en la lista de parámetros formales, todos los restantes deben tener también valores de autoselección. No es posible dejar hoyos en medio de la lista, únicamente los valores anteriores pueden faltar. Por supuesto, los valores especificados deben ser de los tipos correctos o se emitirá un error de compilación. Los valores por defecto pueden darse en el prototipo o en la cabecera de función, pero no en ambos.

Si ellos se dan en ambos lugares, el compilador debe usar solamente el valor por defecto, pero debe verificar cuidadosamente que ambos valores sean idénticos. Esto podría complicar un programa ya muy complicado, por lo que se recomienda que los valores por defecto se declaren en el prototipo y no en la función.

¿POR QUÉ SE INVIERTE LA SALIDA?

Cuando el compilador encuentra una declaración cout, la línea completa de código se repasa inicialmente desde la derecha. Pero cuando se evalúa cualquier función, los datos se analizan de izquierda a derecha.

Por lo tanto en la línea 10, volumen() se evalúa y se muestra primero. Luego las declaraciones del cout se muestran de izquierda a derecha, apareciendo a continuación «Algunos datos de la caja son». Finalmente, el resultado que es devuelto por volumen() aparece al final de la impresión.

Esta impresión no es la que intuitivamente se esperaría, pero así funciona C++.

Las líneas 14 a 17 son parecidas a cualquiera de las líneas de 10 a 12, pero están separadas en dos declaraciones que hacen que la impresión aparezca en el orden esperado.

NUMERO VARIABLE DE ARGUMENTOS

El programa **27 VARARGS.CPP** es una ilustración del uso de un número variable de argumentos en un llamado a función. El compilador para ayuda comprobando cuidadosamente cuántos parámetros se usan en los llamados de función y comprobando también los tipos de estos parámetros.

```
1    #include <iostream.h>
2
3    #include <stdarg.h>
4
5    void mostrar_var(int numero, ...);           // Declaración de una función con un parámetro
6
7    main()
8    {
9
10   int ind = 5;
```

```

11
12  int uno = 1, dos = 2;
13
14  mostrar_var(uno, ind);
15
16  mostrar_var(3, ind, ind + dos, ind + uno);
17
18  mostrar_var(dos, 7, 3);
19  }
20
21  void mostrar_var(int numero, ...)
22  {
23
24  va_list param_pt;
25
26  va_start(param_pt, numero);           // Llamada a la macro
27
28  cout << "Los parámetros son ";
29
30  for (int ind = 0 ; ind < numero ; ind++)
31
32  cout << va_arg(param_pt, int) << " "; // Extracción de un parámetro
33
34  cout << "\n";
35
36  va_end(param_pt);                     // Se cierra la macro
37  }
38
39
40
41

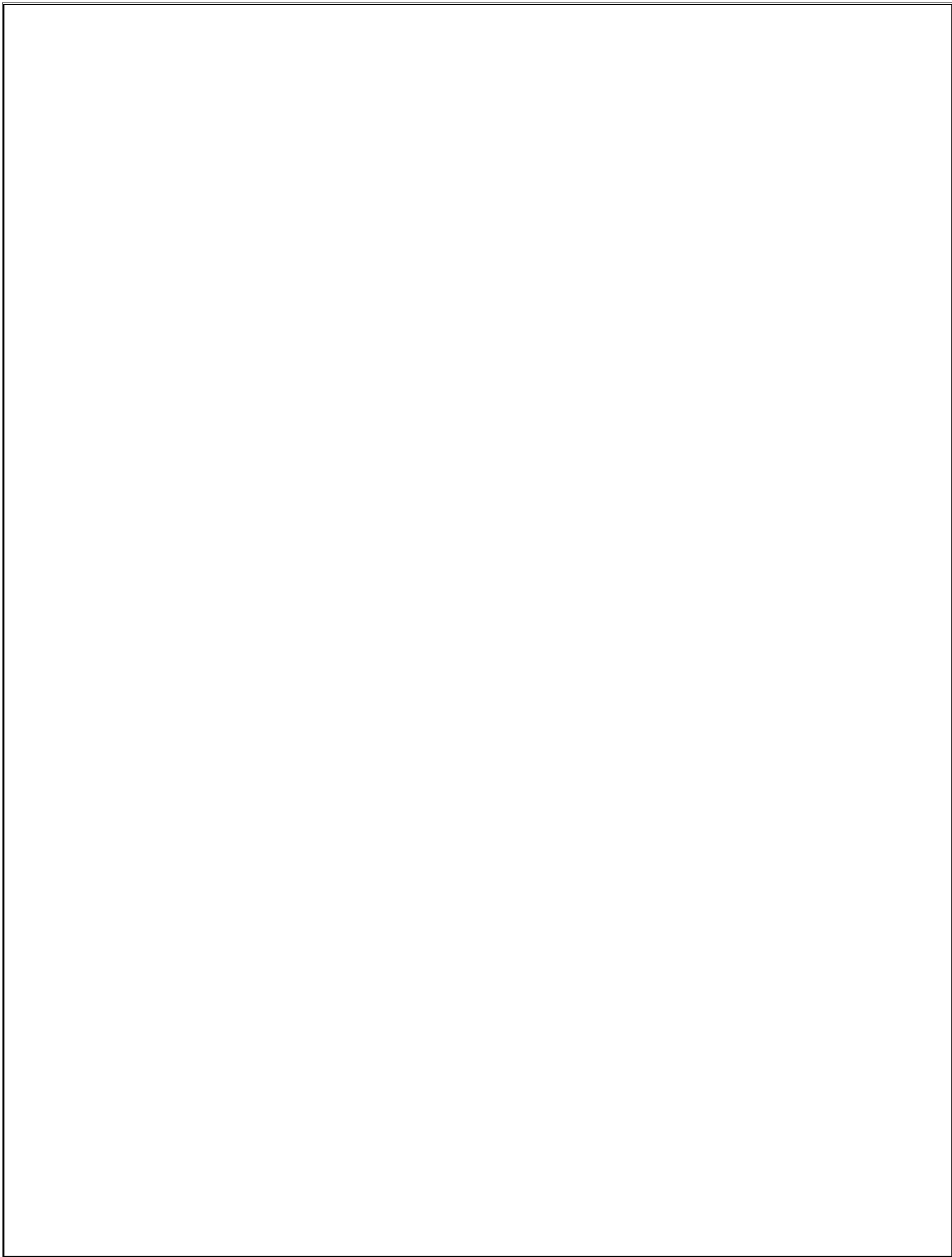
```

En ciertas ocasiones, podemos desear escribir una función que usa un número variable de parámetros. La función `printf()` es un ejemplo de esto. EL ANSI – C tiene una serie de tres macros disponibles en el archivo «`stdarg.h`» para permitir el uso de un número variable de argumentos. Estas macros están disponibles también para el uso con C++, pero necesitamos eliminar de alguna manera la comprobación de tipos que hace C++ con todas las funciones. Los tres puntos ilustrados en la línea 5 harán esto para nosotros. Este prototipo dice que se requiere un único argumento de tipo `int` como el primer parámetro, entonces el compilador no hará ninguna comprobación adicional de tipos.

Es evidente que el programa principal consiste en tres de llamados a la función, cada uno con un número diferente de parámetros, y el sistema no hace diferencias en los tipos utilizados. Mientras el primer parámetro sea de tipo `int`, el sistema compilará y ejecutará el programa sin problemas. Por supuesto el compilador no comprueba los tipos que estén más allá del primer parámetro. El programador debe asegurarse de que usa los tipos de parámetros adecuados.

En este programa, simplemente se muestran los números por pantalla para ilustrar que son manejados adecuadamente.

Por supuesto, debemos tener en cuenta que el usar un número variable de argumentos en un llamado de función puede conducir a oscurecer el código y debe usarse muy poco en un programa, pero esta posibilidad existe y puede usarse si es necesario.

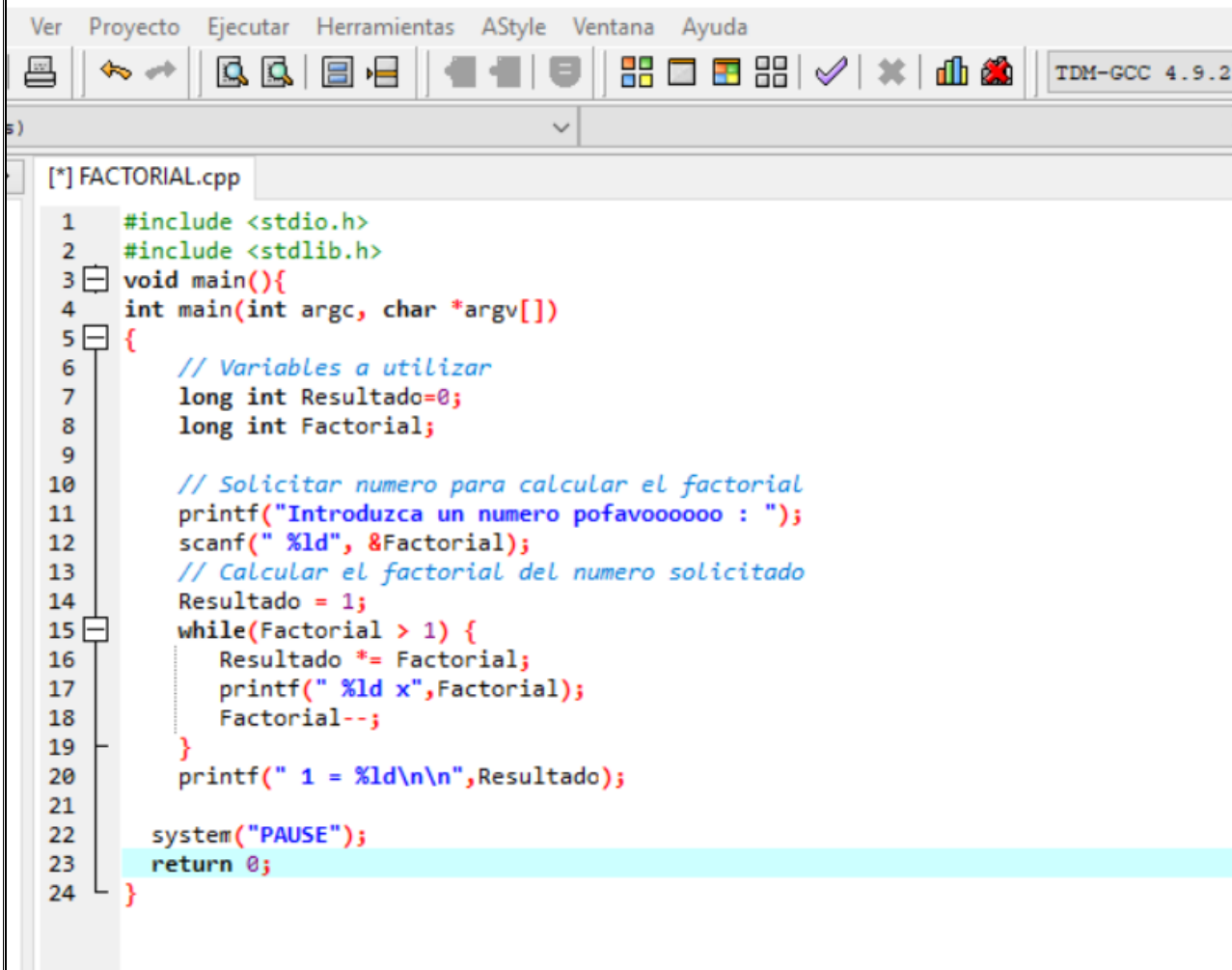


Las actividades deben tener los prototipos de sus funciones, y sus funciones implementadas después del *main*.

- Crear un programa que tenga una función que regrese el factorial de un número de entrada.

recibe un valor n , evaluamos la variable, si la variable es menor que cero la función nos retorna el valor 0, si n es mayor que 1 llamamos otra vez a nuestra función factorial pero con $n-1$ y a esto lo multiplicamos por n , esto se repetirá hasta que n llegue a tomar el valor de 1.

ACTORIAL.cpp - Dev-C++ 5.11



```
1  #include <stdio.h>
2  #include <stdlib.h>
3  void main(){
4  int main(int argc, char *argv[])
5  {
6      // Variables a utilizar
7      long int Resultado=0;
8      long int Factorial;
9
10     // Solicitar numero para calcular el factorial
11     printf("Introduzca un numero pofavoooooooo : ");
12     scanf(" %ld", &Factorial);
13     // Calcular el factorial del numero solicitado
14     Resultado = 1;
15     while(Factorial > 1) {
16         Resultado *= Factorial;
17         printf(" %ld x", Factorial);
18         Factorial--;
19     }
20     printf(" 1 = %ld\n\n", Resultado);
21
22     system("PAUSE");
23     return 0;
24 }
```

ios

Registro de Compilación

Depuración

Resultados

Cerrar

Compilation results...

- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\Asus\Desktop\FACTORIAL.exe
- Output Size: 129.2724609375 KiB
- Compilation Time: 0.50s

Sel: 0 Lines: 24 Length: 584 Insertar Done parsing in 0.015 seconds

í para buscar

lar el factorial
ofavoooooooo : ");
numero solicitado

C:\Users\Asus\Desktop\FACTORIAL.exe

Introduzca un numero pofavoooooooo : 10
10 x 9 x 8 x 7 x 6 x 5 x 4 x 3 x 2 x 1 = 3628800
Presione una tecla para continuar . . .

na calcular
numero pofav
al);
il del numer

C:\Users\Asus\Desktop\FACTORIAL.exe

Introduzca un numero pofavoooooooo : 5
5 x 4 x 3 x 2 x 1 = 120
Presione una tecla para continuar . . .

ial;
torial);

- Crear un programa que tenga una función que regrese el resultado de la serie:

el resultado debe llegar a uno. Programando con la posibilidad de pedir diferentes valores de n. Se guarda, los resultados se ajustan a lo que ya queríamos .

FACTORIAL.cpp - Dev-C++ 5.11

```
Ver Proyecto Ejecutar Herramientas AStyle Ventana Ayuda
[+] FACTORIAL.cpp
1 #include <stdio.h>
2 #include <stdlib.h>
3 void main(){
4 int main(int argc, char *argv[])
5 {
6     // Variables a utilizar
7     long int Resultado=0;
8     long int Factorial;
9
10    // Solicitar numero para calcular el factorial
11    printf("Introduzca un numero pofavoooooooo : ");
12    scanf("%ld", &Factorial);
13    // Calcular el factorial del numero solicitado
14    Resultado = 1;
15    while(Factorial > 1) {
16        Resultado *= Factorial;
17        printf("%ld x", Factorial);
18        Factorial--;
19    }
20    printf(" 1 = %ld\n", Resultado);
21
22    system("PAUSE");
23 }
```

Registro de Compilación Depuración Resultados Cerrar

Compilation results...

```
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\Asus\Desktop\FACTORIAL.exe
- Output Size: 129.2724609375 KiB
- Compilation Time: 0.50s
```

Sel: 0 Lines: 24 Length: 584 Insertar Done parsing in 0.015 seconds

í para buscar

Conclusión:

Un tipo compatible es cualquier tipo simple que puede convertirse a otro en una manera significativa. Por ejemplo, si se usó un entero como el parámetro real y la función esperaba un tipo float como parámetro formal, el sistema hará la conversión automáticamente. También se puede cambiar un float a char, o un char a int. Aunque hay reglas definitivas de conversión que se deben seguir.

Si una función espera un parámetro entero, y se le proporciona uno real, la conversión no se realiza porque los valores son completamente diferentes.

La compatibilidad de tipos discutida anteriormente en este manual se aplica igualmente a la compatibilidad de tipos cuando se hacen llamados a una función. Además, el tipo especificado como el tipo de regreso debe ser compatible con el regreso esperado en la declaración de llamada, o el recopilador emitirá una advertencia.

Además del número correcto de parámetros, los tipos deben ser compatibles o el compilador emitirá un mensaje de error. El aviso sobre el que trabaja el compilador se ve en las líneas 11 y 12, la comprobación de tipo puede hacerse en base al prototipo de la línea 3 aunque la función misma no está definida aún.

Si el prototipo no es dado, el número de parámetros no se verificará, ni los tipos de los parámetros. Aún cuando se tiene el número equivocado de parámetros, se conseguirá una compilación y vinculación aparentemente buenas, pero el programa puede hacer algunas cosas extrañas cuando se ejecute. *jaja*