Funciones

Principal material bibliográfico utilizado

- www.jorgesanchez.net
- Fundamentos de programación C/C++ 4ta Edición – Ernesto Peñaloza Romero - Alfaomega
- Lenguaje C – Adolfo Beltramo, Nélida Matas.

Concepto de función

- Hemos mencionado que C es un lenguaje de funciones.
- Una función es una parte del programa compuesta de sentencias o instrucciones.
- Una función se identifica con un nombre, a través del cual se la invoca.
- Una función se delimita mediante { }, estas indican el inicio y fin de un bloque en C.
- Todo programa en C se base en la función “main”.

Programación modular

- Los programas resuelven problemas de forma computacional.
- Cuando la complejidad de los problemas aumenta, aumenta la complejidad de los programas.
- Para resolver un problema complejo, se puede adoptar el concepto “divide y vencerás”.
- Un programa complejo se puede dividir en partes pequeñas de menor complejidad.
- Cada una de estas partes se puede programar de forma independiente (módulos - en C se llaman funciones).
- La función “main”, es la función que se ejecuta apenas el programa inicia la ejecución, y contiene el código que se ejecuta en primer lugar.
- Dentro del “main” habrá llamadas a funciones ya creadas por el propio programador ó de bibliotecas de C u otras (una biblioteca o librería es una colección de funciones).

Conceptos generales de funciones

especificador de tipo nombre de la función(**declaraciones de parámetros**)
{
cuerpo de la función
}

- El especificador de tipo determina el tipo que devuelve la función.
- Para retornar un valor la función lo hace mediante la instrucción return.
- Si la función no retorna ningún valor, en el especificador de tipo se utiliza void, y en el cuerpo de la función no se utiliza el return.
- El nombre de la función es cualquier identificador válido en C.
- La declaración de parámetros especifica los tipos y variables separados por coma, para incluir los argumentos a recibir por la función. El uso es opcional puede no recibir parámetros.

Conceptos generales de funciones

```
double potencia(int base, int exponente){
  int contador;
  int negativo; /* Indica si el exponente es negativo*/
  double resultado=1.0;

  if (exponente<0) {
       exponente=-exponente;
       negativo=1;
  }
for(contador=1;contador<=exponente;contador++)
       resultado*=base;

  if (negativo) resultado=1/resultado;
  return resultado;
}
```

Conceptos generales de funciones

Prototipos

- Aunque no es obligatorio, se pueden utilizar los prototipos que hacen que el compilador desde el primer momento conozca sobre la existencia de la función.

- Estos se declaran luego de los include.

- La declaración de una función se conoce también como prototipo de la función. En el prototipo de una función se tienen que especificar los parámetros de la función, así como el tipo de dato que devuelve.

```
tipo_de_retorno nombre_de_la_función(lista_de_parámetros);
```

- En el prototipo de una función no se especifican las sentencias que forman parte de la misma, sino sus características.

Conceptos generales de funciones

```c
#include <stdio.h>
#include <conio.h>

/*Prototipo de la función*/
double potencia(int base, int exponente);

int main(){
  int b, e;
  do{
    printf("Escriba la base de la potencia (o cero para salir");
    scanf("%d",&b);
    if (b!=0){
         printf("Escriba el exponente");
          scanf("%d",&e);
         printf("Resultado: %lf", potencia(b,e));
    }
  }while(b!=0);
  getch();
}
/*Cuerpo de la función*/
double potencia(int base, int exponente){
  int contador;
  int negativo=0;
  double resultado=1.0;
  if (exponente<0) {
       exponente=-exponente;
       negativo=1;
  }
  for(contador=1;contador<=exponente;contador++)
       resultado*=base;
  if (negativo) resultado=1/resultado;
  return resultado;
}
```

Conceptos generales de funciones

- Una función no se puede declarar dentro de otra función, por lo que todas las funciones son globales o externas, lo que hace que puedan llamarse desde cualquier parte de un programa.
- Una vez que se ejecutan las instrucciones de la función, se devuelve el control del programa a la siguiente instrucción (si existe) inmediatamente después de la que provocó la llamada a la función
- Cuando se accede a una función desde un determinado punto del programa, se le puede pasar información mediante unos identificadores especiales conocidos como argumentos (son los parámetros).
- Una vez que la función procesa esta información devuelve un valor mediante la instrucción *return.*

  return <expresión>;

  Donde <*expresión> puede ser cualquier tipo de dato salvo un array o una función. Además,* el valor de la expresión debe coincidir con el tipo de dato declarado en el prototipo de la función.
- *Una función puede devolver un solo valor ó nada (en tal caso se lo llama procedimiento).*
- Si no existen *return, la* ejecución de la función continúa hasta la llave del final del cuerpo de la función *(}).*

Conceptos generales de funciones

```
void imprime_cabecera();
{
printf("esta función sólo imprime esta línea");
return;
}
```

equivale a:

```
void imprime_cabecera();
{
printf("esta función sólo imprime esta línea");
}
```

Conceptos generales de funciones

- programa para calcular el precio de un producto

```
#include <stdio.h>
float precio(float base, float impuesto); /* declaración */
main()
{
    float importe = 2.5;
    float tasa = 0.07;
    printf("El precio a pagar es: %.2f\n", precio(importe, tasa));
    return 0;
}
float precio(float base, float impuesto) /* definición */
{
    float calculo;
    calculo = base + (base * impuesto);
    return calculo;
}
```

Conceptos generales de funciones

- El siguiente programa calcula el cubo de los números del 1 al 5 utilizando una función definida por el usuario.

```
#include <stdio.h>
int cubo(int base);
main()
{
	int numero;
	for(numero=1; numero<=5; numero++)
	{
		printf("El cubo del número %d es %d\n", numero, cubo(numero));
	}
	return 0;
}
int cubo(int base)
{
int potencia;
potencia = base * base * base;
return potencia;
}
```

La salida es:

- El cubo del número 1 es 1
- El cubo del número 2 es 8
- El cubo del número 3 es 27
- El cubo del número 4 es 64
- El cubo del número 5 es 125

Acceso a una función

- Cuando se llama a una función dentro de una expresión, el control del programa se pasa a ésta y sólo regresa a la siguiente expresión de la que ha realizado la llamada.

Por ejemplo:

```
a = cubo(2);
calculo = b + c / cubo(3);
```

Acceso a una función

- Vamos a acceder a las funciones *primera y segunda desde la función main.*

```
#include <stdio.h>
void primera(void);
void segunda(void);
main()
{
      printf("La primera función llamada, main\n");
      primera();
      segunda();
      printf("Final de la función main\n");
      return 0;
      }
void primera(void)
{
      printf("Llamada a la función primera\n");
      return;
}
void segunda(void)
{
      printf("Llamada a la función segunda\n");
      return;
}
```

La salida es:
La primera función llamada, main
Llamada a la función primera
Llamada a la función segunda
Final de la función main

Variables locales

- Cuando declaramos variables dentro de la función principal del programa, es decir, dentro de la función *main, están únicamente asociadas a esta función, en otras palabras,* son variables locales de la función *main y no se puede acceder a ellas a través de ninguna* otra función.

- Todas las variables que hemos utilizado en los ejemplos vistos hasta ahora son **variables automáticas. La utilización de la palabra reservada *auto* es *opcional, aunque normalmente*** no se utiliza, por ejemplo:

  auto int contador;

  equivale a

  int contador;

Variables locales

- Utilización del mismo identificador de variable en diferentes funciones mostrando su localidad.

```
#include <stdio.h>
void imprimeValor();
main()
{
	int contador = 0;
	contador++;
	printf("El valor de contador es: %d\n", contador);
	imprimeValor();
	printf("Ahora el valor de contador es: %d\n", contador);
	return 0;
}
void imprimeValor()
{
	int contador = 5;
	printf("El valor de contador es: %d\n", contador);
}
```

- La salida es:

```
El valor de contador es: 1
El valor de contador es: 5
Ahora el valor de contador es: 1
```

Variables globales

- las variables globales se extienden desde el punto en el que se definen hasta el final del programa. En otras palabras, si definimos una variable al principio del programa, cualquier función que forme parte de éste podrá utilizarla simplemente haciendo uso de su nombre.

- La utilización de variables globales proporciona un mecanismo de intercambio de información entre funciones sin necesidad de utilizar argumentos , pero su utilización podría llevarnos a programas de difícil interpretación y complejos de depurar.

Variables globales

- Utilización de variables globales como mecanismo de intercambio de información entre funciones.

```
#include <stdio.h>
void unaFuncion();
void otraFuncion();
int variable;
main()
{       variable = 9;
        printf("El valor de variable es: %d\n", variable);
        unaFuncion();
        otraFuncion();
        printf("Ahora el valor de variable es: %d\n", variable);
        return 0;
}
void unaFuncion()
{       printf("En la función unaFuncion, variable es: %d\n", variable);
}
void otraFuncion()
{       variable++;
        printf("En la función otraFuncion, variable es: %d\n",variable);
}
```

La salida es:

```
El valor de variable es: 9
En la función unaFuncion, variable es: 9
En la función otraFuncion, variable es: 10
Ahora el valor de variable es: 10
```

Variables globales

- Cuando definimos una variable global, lo hacemos de la misma forma en que se declara una variable ordinaria. La definición de una variable global se realiza fuera de cualquier función.

- Si una función desea utilizar una variable global previamente definida, basta con utilizar su nombre sin realizar ningún tipo de declaración especial dentro de la función.

- Sin embargo, si la definición de la función aparece antes de la definición de la variable global, se requiere incluir una declaración de la variable global dentro de la función.

- Para declarar una variable global se utiliza la palabra reservada *extern.*

- Al utilizar *extern,* le estamos diciendo al compilador que el espacio de memoria de esa variable está definido en otro lugar. Es más, en la declaración de una variable externa *(extern) no se* puede incluir la asignación de un valor a dicha variable.

Variables globales

- Utilización del modificador de tipo *extern.*

```
#include <stdio.h>
void unaFuncion();
void otraFuncion();
main()
{
	extern variable;
	variable = 9;
	printf("El valor de variable es: %d\n", variable);
	unaFuncion();
	printf("Ahora el valor de variable es: %d\n", variable);
	return 0;
}
void unaFuncion()
{
	extern variable;
	printf("En la función unaFunción, variable es: %d\n", variable);
}
int variable;
```

Su salida es:
El valor de variable es: 9
En la función unaFuncion, variable es: 9
Ahora el valor de variable es: 9

Variables estáticas

- Las variables estáticas pueden ser tanto locales como globales. Una variable estática local, al igual que una variable automática, está únicamente asociada a la función en la que se declara con la salvedad de que su existencia es permanente. En otras palabras, su contenido no se borra al finalizar la función, sino que mantiene su valor hasta el final del programa.

Variables estáticas

- Por ejemplo, en el siguiente programa declaramos la variable *contador como estática dentro de la función imprimeValor y desde la función main llamamos a esta función varias veces:*

```
#include <stdio.h>
void imprimeValor();
main()
{
	imprimeValor();
	imprimeValor();
	imprimeValor();
	imprimeValor();
	return 0;
}
void imprimeValor()
{
	static int contador = 0;
	printf("El valor de contador es: %d\n", contador);
	contador++;
}
```

Su salida es:

```
	El valor de contador es: 0
	El valor de contador es: 1
	El valor de contador es: 2
	El valor de contador es: 3
```

Paso de argumentos y punteros

- En C todos los argumentos que se pasan a una función se pasan por valor. En otras palabras, se pasa una copia del valor del argumento y no el argumento en sí (por ello, este procedimiento se conoce en algunas ocasiones como **paso por copia). Al pasar** una copia del argumento original a la función, cualquier modificación que se realice sobre esta copia no tendrá efecto sobre el argumento original utilizado en la llamada de la función.

Paso de argumentos y punteros

- Veamos un ejemplo del paso por valor de argumentos a una función:

```
#include <stdio.h>
void modificar(int variable);
main()
{
	int i = 1;
	printf("\ni=%d antes de llamar a la función modificar", i);
	modificar(i);
	printf("\ni=%d después de llamar a la función modificar", i);
}
void modificar(int variable)
{
	printf("\nvariable = %d dentro de modificar", variable);
	variable = 9;
	printf("\nvariable = %d dentro de modificar", variable);
}
```

La salida es la siguiente:

- i=1 antes de llamar a la función modificar
- variable = 1 dentro de modificar
- variable = 9 dentro de modificar
- i=1 después de llamar a la función modificar

Paso de argumentos y punteros

- Sin embargo, en muchas ocasiones lo que queremos es que una función cambie los valores de los argumentos que le pasamos. Para lograrlo se utiliza lo que se conoce como **paso de argumentos por referencia. En estos casos, no se pasa una copia del argumento,** sino el argumento mismo.
- Cuando realizamos un paso de argumentos por referencia en C, realmente lo que estamos pasando son direcciones de memoria.

Paso de argumentos y punteros

- Veamos el ejemplo anterior utilizando el paso de argumentos por referencia:

```
#include <stdio.h>
void modificar(int *variable);
main()
{
	int i = 1;
	printf("\ni=%d antes de llamar a la función modificar", i);
	modificar(&i);
	printf("\ni=%d después de llamar a la función modificar", i);
return 0;
}
void modificar(int *variable)
{
printf("\nvariable = %d dentro de modificar", *variable);
*variable = 9;
printf("\nvariable = %d dentro de modificar", *variable);
}
```

La salida de este ejemplo sería:

- i=1 antes de llamar a la función modificar
- variable = 1 dentro de modificar
- variable = 9 dentro de modificar
- i=9 después de llamar a la función modificar

Paso de argumentos y punteros

- Para pasar la dirección de memoria de una variable se utiliza el operador &. Al finalizar la función, el valor de dicha dirección permanece igual y lo que se ha modificado es el contenido de esa dirección de memoria.

- Dentro de la función se utilizan los punteros para trabajar con las direcciones de memoria *(*variable).*

Otro ejemplo

```
#include <stdio.h>
void total (int);
void imprima ();
int acum;
int main()
{
		int cont;
		acum=0;
		for(cont=0;cont<10;cont++)
		{
				total(cont);
				imprima();
		}

}

void total(int x)
{
		acum=acum+x;
}

void imprima ()
{
		int cont;
		for(cont=0;cont<5;cont++)
				printf('*');
		printf("La suma es: %d \n",acum);
}
```

- La variable acum es global.
- La función total no puede acceder a la variable cont local del main.
- La variable cont de la función imprima no tiene nada que ver con la variable cont del main

Bibliotecas y librerías

- Muchas veces las funciones se almacenan en archivos externos, por lo cual se hace necesario incluir en nuestro programa sentencias que indiquen donde están esas funciones. Esto se realiza mediante las sentencias “include”.

- La sentencia include es una directiva de Preprocesador (una instrucción para el compilador que le indica donde estarán las funciones que utilizamos)

FIN