Programación I

Funciones

Angel Simón

Funciones

Una función es un conjunto de instrucciones que realizan una tarea específica de manera independiente del resto del programa. Se caracterizan por tener un nombre, un valor devuelto y parámetros.

- **Nombre:** El nombre de una función debe obedecer las reglas para nombrar variables.
- **Valor devuelto:** En su tarea a resolver, una función puede devolver un valor (entero, float, char, etc) o no devolver nada (void).
- **Parámetros:** Para poder resolver su tarea, una función puede recibir ninguno o varios parámetros. Estos se guardarán en variables que serán locales a la función.

Llamado a una función

Ejemplo de uso de una función llamada es_primo desde la función main

```
#include <iostream>
using namespace std;
int main(){
   int nro, contPrimos = 0;
   bool r;
   cin >> nro;
   while (nro != 0){
       r = es_primo(nro);
       if (r == true){
           contPrimos++;
       }
      cin >> nro;
   }
   cout << endl << contPrimos;
  return 0;
}
```

Llamado a la función es_primo

- El objetivo de la función **es_primo** es determinar si un número es o no primo.
- Recibe como parámetro un número **int** (que la función evaluará)
- Devuelve como valor de retorno un valor **bool** siendo **true** si es primo y **false** si no lo es.

Declaración

```
tipo nombreFuncion (parámetros);
```

La **declaración** de una función debe determina el tipo de dato que devuelve primero.

Luego cómo se llamará la función.

Por último, los tipos de datos de los parámetros que recibirá separados por coma. Si no recibe parámetros no se pone nada entre los paréntesis.

Declaración de una función

Un ejemplo de una declaración de función puede ser la siguiente:

```
bool es_primo(int);
```

Declaración de función que recibe un entero y devuelve un valor bool.

```
void mostrar_primos_entre(int, int);
```

Declaración de función que recibe dos enteros y no devuelve nada.

Definición

```
tipo nombreFuncion (parámetros){
    // Código de la función
}
```

La **definición** de una función es similar a la **declaración** pero se nombran las variables en los parámetros y se establece lo que hace la función entre las llaves de la misma.

Definición de una función

```
bool es_primo(int numero){
  int i, c=0;
  for(i=1; i<=numero; i++){
    if (numero % i == 0){
      c++;
    }
  }
  if (c == 2){
    return true;
  }
  else{
    return false;
   }
}
```

Definición de la función es_primo

- La función recibe un parámetro entero llamado numero. Esta variable es **local a la función**.
- La función declara las variables locales i y c para resolver su tarea.
- La función **devuelve** verdadero si encuentra dos divisores o falso si encuentra otra cantidad.

Definición de una función

```
void mostrar_primos_entre (int inicio, int fin){
  int i;
  bool r;
  for(i=inicio; i<=fin; i++){
    r = es_primo(i);
    if (r == true){
       cout << i << endl;
    }
  }
}
```

Definición de la función mostrar_primos_entre

- La función recibe dos parámetros entero llamados inicio y fin. Estas variables son **locales a la función**.
- La función declara las variables locales i y r para resolver su tarea.
- La función **no dvuelve** un valor de retorno.

Ámbito de variables

Una variable es local al **ámbito** en el que fue declarada. El ámbito está dado por las llaves. Fuera de ellas la variable no existe.

```
int main(){
  int i=0, numero= 9;
  float val;
  val = mi_funcion(numero);
  return 0;
}
```

```
float mi_funcion(int dato){
  float i;
  i = dato / 2;
  return i;
}
```

Variable	Valor

Variable	Valor

Ámbito de variables

Una variable es local al **ámbito** en el que fue declarada. El ámbito está dado por las llaves. Fuera de ellas la variable no existe.

```
int main(){
  int i=0, numero= 9;
  float val;
  val = mi_funcion(numero);
  return 0;
}
```

```
float mi_funcion(int dato){
  float i;
  i = dato / 2;
  return i;
}
```

Variable	Valor
i	0
numero	9
val	4.5

Variable	Valor
dato	9
i	4.5

Retornar valores

Una función puede retornar un valor o ninguno. Esto se logra mediante la palabra reservada **return**.

Si se encuentra un return en una función, la misma finaliza y devuelve el valor indicado a quien la llamó.
Ejemplo: *return valor*;

En los casos de funciones que no devuelven nada. El funcionamiento y la lógica son iguales sólo que no hay que indicar ningún valor al llamadar a return.
Ejemplo: *return;*

Funciones y vectores

Una función puede recibir uno o varios vectores como parámetros. Lo puede hacer indicando el tipo de dato, el nombre y par de corchetes.

Ejemplo: *void mi_funcion(int mi_vector[], fioat mi_otro_vector[])*

Una función puede recibir también una matriz. Es similar a los vectores pero indicando el tamaño de todas las dimensiones menos la primera.

void mi_funcion(int mi_matriz[][5], fioat mi_otra_matriz[][10])

Siempre que se reciba un vector como parámetro ser recibe **por dirección**. Esto determina que si modifico el vector en la función también lo modificaré en la función que la llama.

Funciones y vectores

```
void cargar_vector(int vec[], int tam){
  int i;
  int (i=0; i<tam; i++){
    cout << "Ingresar valor: ";
    cin >> vec[i];
  }
  return;
}
```

```
void que_como(int hora, char comida[]){
  if (hora >= 6 && hora <=11){
  strcpy(comida, "Desayuno");
  }
  else if (hora >= 12 && hora <=16){
   strcpy(comida, "Almuerzo");
  }
  else if (hora >= 17 && hora <=23){
   strcpy(comida, "Cena");
  }
  else{
   strcpy(comida, "Cualquier cosa");
  }
}
```

Funciones

Ventajas de utilizar funciones:

- Dividir un problema grande en pequeños problemas más chicos. Abstrayendo esa tarea del problema general a resolver.
- Dar legibilidad al código, haciendo algoritmos más fáciles de entender y mantener.
- Dar capacidad de reutilización al código. Haciendo funciones que resuelven tareas únicas aumenta la posibilidad de poder utilizarlas nuevamente.
- Evita la repetición de código. Cada vez que necesito realizar la misma tarea puedo llamar nuevamente a la función.
- Modificar el funcionamiento de un programa o encontrar errores es más fácil si tenemos la lógica separada en funciones.

Ejemplo

- Hacer un programa que determine si un número ingresado es primo o no.

- Hacer un programa que muestre los números primos entre el 1 y el 1000.