

Funciones y funciones de librería

OMI YUC - Alonso Huerta

“A function is a block of code with a name.”

– C++ Primer

¿Cómo crear funciones?

```
int main(){  
    return 0;  
}
```

¿Cómo crear funciones?

- Return type `int`
- Name `main`
- Parameters `()`
- Body { `return 0;` }

```
int main(){  
    return 0;  
}
```

```
int suma(int a, int b){  
    int c = a + b;  
    return c;  
}
```

```
string saluda(string name){  
    return "Hola, " + name + "!";  
}
```

```
string es_mayor(int a, int b){  
    if(a > b){  
        return "primero es mayor que segundo";  
    }  
    return "segundo es mayor o igual que primero";  
}
```

¿Cómo llamar una función?

```
string saluda(string name){  
    return "Hola, " + name + "!";  
}  
  
string saludo = saluda("Raymundo");  
  
cout << saludo; // "Hola, Raymundo!"
```

¿Cómo llamar una función?

```
int suma(int a, int b){  
    int c = a + b;  
    return c;  
}  
  
int sumado = suma(1, 6);  
  
cout << sumado; // 7
```

void

// Observa como la función no regresa nada

```
void line_break(){  
    cout << "-----";  
}
```


void

```
// Observa que aunque la función tiene un `return`,  
// no regresa nada.
```

```
void es_mayor(int a, int b){  
    if(a > b){  
        cout << "primero es mayor que segundo";  
        return;  
    }  
    cout << "segundo es mayor o igual que primero";  
}
```

Parámetros (arreglos)

```
double promedio(int arr[], size);
```

```
double promedio(int *arr, size);
```

```
double promedio(int arr[6], size);
```

```
int mi_arreglo[] = {1, 2, 3, 4, 5, 6};
```

```
double prom = promedio(mi_arreglo, 6);
```


Parámetros (por referencia)

```
int aumenta(int a){  
    a += 1;  
    return a;  
}
```

```
int num = 6;  
int aum = aumenta(num);
```

```
cout << num; // 6  
cout << aum; // 7
```

Parámetros (por referencia)



```
int aumenta(int &a) {  
    a += 1;  
    return a;  
}  
  
int num = 6;  
int aum = aumenta(num);  
  
cout << num; // 7  
cout << aum; // 7
```

Parámetros (por referencia)

```
string to_upper(string nombre){  
    for(int i = 0; i < nombre.size(); i++){  
        nombre[i] = toupper(nombre[i]);  
    }  
    return nombre;  
}
```

```
string mi_nombre = "Raymundo";  
string nombre_upper = to_upper(mi_nombre);
```

```
cout << mi_nombre << endl; // Raymundo  
cout << nombre_upper << endl; // RAYMUNDO
```

Parámetros (por referencia)

↓

```
string to_upper(string &nombre){  
    for(int i = 0; i < nombre.size(); i++){  
        nombre[i] = toupper(nombre[i]);  
    }  
    return nombre;  
}  
  
string mi_nombre = "Raymundo";  
string nombre_upper = to_upper(mi_nombre);  
  
cout << mi_nombre << endl; // RAYMUNDO  
cout << nombre_upper << endl; // RAYMUNDO
```

Parámetros (valores predeterminados)

```
void ecuacion(int a, int b = 0, int c = 0){  
    cout << a << "x^2 + ";  
    cout << b << "x + ";  
    cout << c;  
}
```

```
ecuacion(5);           // 5x^2 + 0x + 0  
ecuacion(6, 2);        // 6x^2 + 2x + 0  
ecuacion(9, 0, 1);     // 9x^2 + 0x + 1
```

¿Para qué me sirve?

Compactar y reutilizar código

```
double promedio(int arreglo[], int size){
```

```
    int suma = 0;  
    for(int i = 0; i < size; i++){  
        suma += arreglo[i];  
    }
```

```
    double promedio = (double)suma/size;
```

```
}
```



```
promedio(arreglo1, 5);  
promedio(arreglo2, 8);
```

Funciones de librería

`swap(var1, var2);` // intercambia el valor de var1 con var2

`min(var1, var2);` // regresa el valor entre el mínimo de var1 y var2

`max(var1, var2);` // regresa el valor entre el máximo de var1 y var2

`pow(base, pot);` // regresa base potenciado a pot

`sort(arr, arr + size);` // ordena el arreglo de menor a mayor

`reverse(arr, arr + size);` // invierte el arreglo

`reverse(str.begin(), str.end());` // invierte el string

```
int var1 = 23, var2 = 43;

swap(var1, var2);
cout << var1 << ' ' << var2; // 43 23

int m = min(var1, var2);
cout << m; // 23

int M = max(var1, var2);
cout << M; // 43

double num = pow(2, 1/2);
cout << num; // 1.414
```

```
int arr[] = {3, 7, 1, 9};
```

```
sort(arr, arr + 4); // [1, 3, 7, 9]
```

```
reverse(arr, arr + size); // [9, 7, 3, 1]
```

```
int arr2[] = {3, 4, 8, 10, 7, 4, 5};
```

```
sort(arr + 2, arr + 5); // [3, 4, 7, 8, 10, 4, 5]
```

```
reverse(arr + 3, arr + 6); // [3, 4, 7, 5, 4, 10, 8]
```

pair<T, S>
#include <utility>

```
pair<int, int> pareja_int_int;  
pareja_int_int = make_pair(3, 5);
```

```
cout << pareja_int_int.first << endl; // 3  
cout << pareja_int_int.second << endl; // 5
```

```
pair<int, string> pareja_int_string;  
pareja_int_string = make_pair(7, "palabra");
```

```
cout << pareja_int_string.first << endl; // 7  
cout << pareja_int_string.second << endl; // palabra
```

```
pair<char, string> pareja_char_string;  
pareja_char_string = make_pair('@', "kaboom");
```

```
cout << pareja_char_string.first << endl; // @  
cout << pareja_char_string.second << endl; // kaboom
```

```
pair< pair < int, int>, int > p;  
p = make_pair(make_pair(1, 2), 3);  
  
cout << p.first.first << endl; // 1  
cout << p.first.second << endl; // 2  
cout << p.second << endl; // 3
```

```
pair<int, int> p1 = make_pair(1, 2);  
pair<int, int> p2 = make_pair(1, 2);  
  
cout << (p1 == p2) << endl; // 1 (true)
```



```
pair<int, int> p1 = make_pair(1, 2);  
pair<int, int> p2 = make_pair(3, 4);  
  
cout << (p1 <= p2) << endl; // 1 (true)
```

```
pair<int, int> p1 = make_pair(1, 2);  
pair<int, int> p2 = make_pair(1, 3);  
  
cout << (p1 <= p2) << endl; // 1 (true)
```

vector<T>

#include <vector>

```
int arr[INFI]; // INFI un valor lo suficientemente grande
int size = 0;

// Agregar un valor al final del arreglo

// Saber cuantos valores tengo en el arreglo

// Saber si el arreglo está vacío

// Obtener referencia a el valor en el indice i

// Obtener el valor al inicio del arreglo

// Obtener el valor al final del arreglo

// Borrar el elemento al final del arreglo

// Vaciar el arreglo

// Borrar un valor en el indice i
```

```
// Agregar un valor al final del arreglo
void push_back(int arr[], int &size, int valor){

}
```

```
// Agregar un valor al final del arreglo  
void push_back(int arr[], int &size, int valor){  
  
    arr[size] = valor;  
    size = size + 1;  
  
}
```

```
// Saber cuantos valores tengo en el arreglo
int size(int arr[], int &size){

}
```

```
// Saber cuantos valores tengo en el arreglo  
int size(int arr[], int &size){  
  
    return size;  
  
}
```



```
// Saber si el arreglo está vacío
bool empty(int arr[], int &size){

}
```

```
// Saber si el arreglo está vacío
bool empty(int arr[], int &size){

    return size(arr, size) == 0;

}
```

```
// Obtener referencia a el valor en el indice i
int& at(int arr[], int &size, int i){

}
```

```
// Obtener referencia a el valor en el indice i
int& at(int arr[], int &size, int i){

    if(i < 0 || i >= size){} // error
    return arr[i];

}
```

```
// Obtener el valor al inicio del arreglo
int front(int arr[], int &size){

}
```

```
// Obtener el valor al inicio del arreglo  
int front(int arr[], int &size){  
  
    if(empty(arr, size)){ // error  
        return arr[0];  
    }  
}
```

```
// Obtener el valor al final del arreglo
int front(int arr[], int &size){

}
```

```
// Obtener el valor al final del arreglo  
int front(int arr[], int &size){  
  
    if(empty(arr, size)){ // error  
        return arr[size - 1];  
    }  
}
```



```
// Borrar el elemento al final del arreglo
void pop_back(int arr[], int &size){

}
```

```
// Borrar el elemento al final del arreglo  
void pop_back(int arr[], int &size){  
  
    if(empty(arr, size)){ // error  
        size = size - 1;  
  
    }  
}
```

```
// Vaciar el arreglo  
void clear(int arr[], int &size){  
  
}
```

```
// Vaciar el arreglo  
void clear(int arr[], int &size){  
    size = 0;  
}
```

```
// Borrar un valor en el índice i
void erase(int arr[], int &size, int i){

}
}
```

```
// Borrar un valor en el indice i
void erase(int arr[], int &size, int i){

    for(int idx = i; idx < size-1; idx++){
        arr[idx] = arr[idx+1];
    }

    pop_back(arr, size);
}
```

```
int arr[INFI]; // INFI un valor lo suficientemente grande
int size = 0;

// Agregar un valor al final del arreglo
push_back(arr, size, valor);
// Saber cuantos valores tengo en el arreglo
size(arr, size);
// Saber si el arreglo está vacío
empty(arr, size);
// Obtener referencia a el valor en el indice i
at(arr, size, i);
// Obtener el valor al inicio del arreglo
front(arr, size);
// Obtener el valor al final del arreglo
back(arr, size);
// Borrar el elemento al final del arreglo
pop_back(arr, size);
// Vaciar el arreglo
clear(arr, size);
// Borrar un valor en el indice i
erase(arr, size, i);
```

```
vector<int> vec = {1, 2, 3};  
  
vec.push_back(9); // [1, 2, 3, 9]  
  
vec.push_back(10); // [1, 2, 3, 9, 10]  
  
vec.size(); // 5  
  
vec.pop_back(); // [1, 2, 3, 9]  
  
cout << vec.size(); // 4  
  
cout << vec[3]; // 9  
  
cout << vec.front(); // 1  
  
cout << vec.back(); // 9  
  
vec.clear(); // [ ]  
  
cout << vec.empty(); // 1 (true)
```



```
vector<char> vec = {'i', 'e', 'b', 'd'};  
sort(vec.begin(), vec.end()); // ['b', 'd', 'e', 'i']  
reverse(vec.begin(), vec.end()); // ['i', 'e', 'd', 'b'];
```

```
bool criterio(int a, int b){  
    return a < b;  
}  
  
sort(vec.begin(), vec.end(), critetio);
```

```
bool criterio(int a, int b){  
    return a.first > b.first;  
}  
  
int arr[] = {7, 5, 9, 8};  
  
sort(arr, arr+4, criterio); // [9, 8, 7, 5]
```

```
bool criterio(pair<int, int> a, pair<int, int> b){  
    if(a.first == b.first){  
        return a.second > b.second;  
    }  
    return a.first < b.first;  
}  
  
vector< pair<int, int> > vec_p = {{2, 3}, {4, 3}, {2, 6}};  
  
sort(vec_p.begin(), vec_p.end(), criterio);  
// [{2, 6}, {2, 3}, {4, 6}]
```