Ayudantía 2

Carlos Lagos - carlos.lagosc@usm.cl

Funciones

¿Qué es una función en C++?

Una función es un bloque de código reutilizable que realiza una tarea específica dentro de un programa. Las funciones son esenciales para la modularidad y organización del código, permitiendo:

- **División de tareas:** Dividir un programa en partes más manejables.
- **Reutilización:** Invocar una función múltiples veces en diferentes partes del programa.
- **Abstracción:** Ocultar los detalles de implementación, permitiendo al usuario enfocarse en la lógica.

Funciones

```
tipo_retorno nombre_función(tipo_parametro parametro1, tipo_parametro parametro2) {
    // Cuerpo de la función
    return valor;
}
```

- **tipo_retorno:** Tipo de dato devuelto (e.g., `int`, `void`).
- **nombre_función:** Identificador de la función.
- **tipo_parametro:** Tipo de datos de los parámetros (e.g., `int`, `double`).
- **parametro1, parametro2:** Nombres de los parámetros como variables locales.
- **return:** Palabra clave para devolver un valor.

Ejemplos

```
int sumar(int a, int b) {
    return a + b;
}
```

```
// Las funciones de tipo void no requieren return
void hello_world() {
    std::cout << "Hola mundo" << std::endl;
}
```

Ejercicios

1. Crea una función que, dado un arreglo de enteros de tamaño 10 entregado por parámetro, devuelva la multiplicación de todos sus elementos.
2. Dado un arreglo A de tamaño N (donde N es como máximo 100), define el costo como la cantidad mínima de operaciones necesarias para que se cumpla la condición: $\prod_{i=1}^{N} a_i = 0$. Una operación consiste en elegir un número del arreglo y sumarle o restarle uno. Debes crear una función que devuelva este costo.

¿Qué es una función recursiva?

Matemáticamente, una función recursiva es aquella que se refiere a sí misma, es decir, se llama a sí misma dentro de su definición. Si no se implementa correctamente, puede llevar a bucles infinitos, por lo que es crucial tener cuidado al programarla.

Ejemplos de funciones recursivas

Secuencia de Fibonacci

La sucesión de Fibonacci para calcular el término $f(i)$ se define de la siguiente manera:

$$f(i) = \begin{cases} 0 & \text{si } i = 0 \\ 1 & \text{si } i = 1 \\ f(i-1) + f(i-2) & \text{si } i > 1 \end{cases}$$

Ejercicios

1. Implementa una función recursiva que calcule el factorial de un número.
2. Implementa una función recursiva que calcule los términos del triángulo de Pascal.

Memoria dinamica

Memoria Dinámica

En C++, la memoria dinámica es aquella que se reserva en tiempo de ejecución y es gestionada por el programador. Esta memoria se asigna en el **heap**, a diferencia de la memoria estática que se asigna en la **pila** (stack) para variables normales.

Reserva de Memoria Dinámica

La reserva de memoria dinámica se realiza utilizando el operador `new`, seguido del tipo de dato.

```
tipo_de_dato *nombre_puntero = new tipo_de_dato;
```

Por ejemplo:

```
int *ptr_entero = new int;
```

Asignación de Valores

Una vez reservada la memoria, se puede acceder a ella utilizando el puntero y asignarle un valor.

```
*nombre_puntero = valor;
```

Por ejemplo:

```
*ptr_entero = 10;
```

Liberación de Memoria

Es importante liberar la memoria asignada dinámicamente para evitar fugas de memoria. Esto se hace utilizando el operador `delete`.

```
delete nombre_puntero;
```

Por ejemplo:

```
delete ptr_entero;
```

Uso de la Memoria Dinámica

La memoria dinámica es útil cuando se necesita almacenar datos de manera flexible o cuando el tamaño del almacenamiento no se conoce en tiempo de compilación.

```
int main() {
    int *ptr_entero = new int;
    *ptr_entero = 10;
    delete ptr_entero;
    return 0;
}
```

Arreglos usando memoria dinamica

```cpp
#include <iostream>

using namespace std;

int main(){
    int *arregloDinamico;
    int n;
    cin >> n;
    arregloDinamico = new int[n];
    for(int i = 0; i < n; i++){
        cin >> arregloDinamico[i];
    }

    for(int i = 0; i < n; i++){
        if(arregloDinamico[i] % 2 == 0){
            cout << *(arregloDinamico + i) << endl;
        }else{
            cout << arregloDinamico[i]*-1 << endl;
        }
    }

    delete[] arregloDinamico;
    return 0;
}
```

Arreglos 2D usando memoria dinamica

```cpp
#include <iostream>

using namespace std;

int main(){
    int n;
    cin >> n;
    int **arreglodinamico2d;
    arreglodinamico2d = new int*[n];
    for(int i = 0; i < n; i++){
        arreglodinamico2d[i] = new int[i + 1];
        for(int j = 0; j < i + 1; j++){
            arreglodinamico2d[i][j] = j + 1;
        }

    }
    for(int i = 0; i < n; i++){
        for(int j = 0; j < i + 1; j++){
            cout << arreglodinamico2d[i][j] << " ";
        }
        cout << endl;

    }
    cout << endl;

    for(int i = 0; i < n; i++){
        delete[] arreglodinamico2d[i];
    }
    delete[] arreglodinamico2d;
    return 0;
}
```

Ejercicio

Implementa un programa que gestione un arreglo dinámico, el cual comienza vacío y puede modificarse mediante operaciones ingresadas por consola. Las operaciones disponibles son:

1. Imprimir los valores almacenados en el arreglo.
2. Insertar un elemento al final del arreglo.
3. Eliminar un elemento en una posición i del arreglo.

Es importante optimizar el uso de memoria, asegurando que no quede memoria sin utilizar mediante una gestión eficiente con memoria dinámica.

Memoria dinamica y funciones

Imaginen que quieren crear una función que devuelva un puntero a un arreglo creado dentro de la misma función. Esto generará un error, ya que la memoria en la pila (stack) se libera automáticamente al finalizar la función o el entorno en el que fue llamada. En cambio, la memoria asignada en el heap, que es gestionada por el programador, se mantiene hasta que el programador decida liberarla.

Memoria Dinamica

```cpp
#include <iostream>
using namespace std;
int* crearArregloEnHeap() {
    int* arreglo = new int[5]; // Memoria en el heap
    for (int i = 0; i < 5; i++) {
        arreglo[i] = i + 1;
    }
    return arreglo; // La memoria en el heap se mantiene hasta que la liberemos
}

int main() {
    int* arregloHeap = crearArregloEnHeap();
    for (int i = 0; i < 5; i++) cout << arregloHeap[i] << " ";
    cout << endl;
    delete[] arregloHeap;
    return 0;
}
```