AYUDANTÍA C

Raúl Andrés Álvarez Esteban
Ricardo Esteban Schilling Broussaingaray

IIC2333 [2020-1] - Sistemas Operativos y Redes

INTRODUCCIÓN

INTRODUCCIÓN

Los objetivos de esta ayudantía son:

- Tener un primer acercamiento al lenguaje C.
- Aprender los elementos importantes para programar en este lenguaje.
- Aprender uso de memoria.
- Aprender a descargar y subir archivos desde el servidor del curso.

EL LENGUAJE C

¿Por qué utilizar C y no Python?

¿Por qué utilizar C y no Python?

Nos permite tener **mayor control** sobre nuestro programa. También **varios sistemas operativos estan programados en C**.

SINTÁXIS EN C

La sintáxis en este lenguaje es distinta a lo que ustedes usualmente conocen. Específicamente:

1. Se debe explicitar el tipo de cada variable.
2. Funciona en torno a la función `main`.
3. Se debe explicitar los bloques del programa con corchetes y puntos y coma.

1. Se debe explicitar el tipo de cada variable.

Cada vez que se defina una variable se debe especificar su tipo de la forma `tipo nombre = valor`. Los tipos disponibles son `int`, `char`, `float`, `double` y `void`, entre otros.

Por ejemplo:

```
float pi = 3.14;
int promedio_final = 6;
char letra = 'c';
```

2. Funciona en torno a la función `main`.

Cuando se ejecuta un programa en C lo que se ejecuta es la función `main`, por ejemplo:

```
int main(int argc, char *argv[])
{
    printf("Hola Mundo!");
    return 0;
}
```

Donde `argc` es un número que indica la cantidad de argumentos que fueron entregados via consola y `argv` es un array con dichos argumentos.

3. Se debe explicitar los bloques del programa con corchetes y puntos y coma.

Toda línea de código (excepto inicio de funciones) debe ser terminada con punto y coma. Mientras que todo bloque de código debe estar rodeado por corchetes. Se considera bloque de código lo siguiente:

- Cuerpo de una función.
- Cuerpo de un `if`
- Cuerpo de un `while`
- etc.

Es decir, todo conjunto de lineas de código, es considerado un bloque.

SINTÁXIS EN C

Por ejemplo, así se ven los `while` y los `if`:

```
while (condicion)
{
    // Código del while
}

if (condicion)
{
    // Código del if
}
```

Ciclos For

La sintáxis de estos ciclos es la siguiente:

```
for ( init; condition; increment ) {
    // Código
}
```

En `init` se define el contador, en `condition` se define la condición de término y en `increment` se define como se va a modificar el contador en cada ciclo. Un for que cuenta de 0 a 10 es el siguiente:

```
for ( int i = 0; i < 11; i++ ) {
    // Código
}
```

Funciones

La sintáxis de funciones es la siguiente:

```
tipo_retorno nombre_funcion(tipo_argumento nombre_argumento)
{
    // Código
}
```

Un ejemplo de una función sería el siguiente:

```
int multiplicacion(int a, int b)
{
    return a * b;
}
```

MANEJO DE MEMORIA Y PUNTEROS

MANEJO DE MEMORIA

Todo programa utiliza memoria para funcionar, pero este es un **recurso limitado** y tiene que ser manejado por el programa. En lenguajes como `Python` este manejo se hace automáticamente, pero en C se debe hacer **Manualmente**.

¿Por qué hay que manejar memoria?

¿Por qué hay que manejar memoria?

Cada función en C tiene su propio espacio de memoria, llamado `Stack`, el cual es eliminado al haber terminado la ejecucción de dicha función. Para poder generar cambios en el estado de nuestro programa, es necesario tener un espacio de memoria separado, donde existirán objetos mas `permanentes`. Esta memoria debe ser pedida en cantidades exactas, por lo que es necesario manejar tanto el uso de esta.

MANEJO DE MEMORIA

Las funciones típicas para manejar memoria son las siguientes:

- `malloc(size)`: Reserva la cantidad de espacio dada por `size` y retorna el puntero a esta dirección.
- `calloc(n, item_size)`: Reserva `n` espacios de tamaño `item_size`, además de inicializar el contenido de cada posición
- `free(item)`: Libera la memoria actualmente ocupada por `item` (Debe ser un puntero obtenido mediante `malloc` o `calloc`)

Estas funciones retornan un tipo de variable llamada **puntero**.

Un puntero es un número que representa una dirección de memoria.

Ejemplo

```
int* numeros = calloc(3, sizeof(int));
numeros[0] = 50;
numeros[1] = 51;
numeros[2] = 52;
for (int i = 0; i < 3; i++) {
    printf("%i\n", numeros[i]);
};
```

Al utilizar este código pedimos memoria para tres `int` mientras obtenemos el tamaño de un `int` mediante la función `sizeof`.

Es importante notar del ejemplo anterior que, efectivamente, en C los arreglos no son más que punteros. Por ejemplo:

```
int ejemplo[10];
```

Aquí se quiere definir un arreglo de enteros de 10 elementos. No obstante, lo que realmente se declara es un puntero al primer elemento del arreglo. Por lo tanto:

```
if (ejemplo == &ejemplo[0]){
  printf("Son iguales.\n");
}
```

Imprimirá el mensaje.

Operadores de Punteros

Existen dos operadores de punteros:

- *: Obtiene el contenido de un puntero.
- &: Obtiene la dirección de memoria (puntero) de una variable.

Por ejemplo:

```
int *num = malloc(sizeof(int));
*num = 10;
int dir = &num;
```

FUNCIONES TÍPICAS Y MANEJO DE STRINGS

STRINGS

En C conocemos como string a los arreglos de caracteres individuales, es decir que un string no es mas que un `char *`

La función mas básica que tenemos para mostrar un string en pantalla se llama `printf`, cuyo primer argumento es un string que será formateado y enviado a la pantalla, mientras que el resto de los argumentos son valores que llenarán el formato de string.

Ejemplo de uso:

```
int main(int argc, char* argv[]){
    int age = 21;
    char *nb = "Ricardo";
    printf("Soy %s y tengo %d años\n", age, nb);
}
```

Pueden encontrar todos los especificadores de formato en esta página.

Similarmente, la función para leer strings del input es scanf.

```
int main(int argc, char* argv[]){
    int edad;
    scanf("Tengo %d años", &edad);
    printf("Tienes %d años\n", edad);
}
```

Esta función lee un string de input e intenta formatear de manera correspondiente. Luego de poner el string de formato, debemos entregarle punteros a las variables donde guardaremos los valores formateados.

Existen funciones similares que nos permiten leer y escribir en archivos. Estas funciones son `fprintf` y `fscanf`, las cuales reciben los mismos parámetros que las funciones originales, pero además reciben un puntero a un archivo.

Notar que:

```
fprintf(stdout, x, y) == printf(x, y)

fscanf(stdin, x, y) == scanf(x, y)
```

ARCHIVOS

Para abrir un archivo en C, disponemos de la función `fopen`, que recibe, similar a python, el nombre del archivo y el modo en el que queremos abrirlo.

Esta función nos entrega entonces un puntero al archivo, el cual podemos usar con `fprintf` y `fscanf`.

MANEJO DE STRINGS

Debido a la naturaleza tanto de la memoria donde son contenidos los strings como de su condición de arreglo, en C poseemos diversas funciones para manejo de strings, entre ellas `strcpy` que nos permite copiar un string a otro (siempre que exista el espacio suficiente), `strlen` que nos da el largo del string y `strcmp` que nos permite comparar a ver si dos strings son iguales o no.

COMPILACIÓN

COMPILACIÓN Y GCC

A diferencia de lenguajes interpretados como python, C es un lenguaje compilado, lo cual significa que debemos convertir nuestro código a un ejecutable antes de poder correrlo.

Usualmente, se utiliza el comando `gcc` para compilar un archivo. Por ejemplo, el comando `gcc -o exe ej.c` compila el programa `ej.c` en el ejecutable `exe`.

No obstante lo anterior, se puede automatizar este procedimiento con un archivo `Makefile`, el que posee en su interior todas las configuraciones necesarias para la compilación. Este último se ejecuta con el comando `make`.

FIN