UG

# Lenguaje de Programación

## C++

## Repaso de Material C++

José Luis Alonzo Velázquez

Universidad de Guanajuato

Sesión 34

## Variable

Para poder leer algo, necesitamos un lugar donde poner lo leído, i.e. necesitamos un lugar en la memoria de la maquina donde podamos guardar esta información. A este "lugar" lo llamaremos **objeto**. Un objeto es una región de memoria que tendrá un **tipo** que especifica que clase de información esta siendo colocada en el. Este objeto es llamado **variable**. Será en estas variables donde guardaremos información en nuestros programas.

## Variables locales y globales

Cuando declaramos una variable dentro de main o alguna función, solo actuara de manera **local**, es decir, solo existirá dentro de main o dicha función. En cambio si declaramos la variable fuera del main o cualquier función esta sera una variable **global**, la cual podrá ser modificada desde el main o cualquier función.

## Tipos de Variables

| | |
|---|---|
| bool x | x es un Booleano (valor true o false). |
| char x | x es un carácter (usualmente 8 bits). |
| short x | x es un short int (usualmente 16 bits). |
| int x | x es el entero por defecto. |
| float x | x es un número con punto flotante. |
| double x | x es un flotante con doble precisión. |
| const T x | x es constante (immutable) version de T. |
| long T x | x es un largo T. |
| unsigned T x | x es un sin signo T. |
| signed T x | x con signo T. |

## Especificadores

| Especificador | Salida | Ejemplo |
|---|---|---|
| c | Carácter | a |
| s | Cadena de caracteres | sample |
| d o i | Entero con signo | 392 |
| e | Notación científica e | 3.9265e+2 |
| E | Notación científica E | 3.9265E+2 |
| f | Punto flotante decimal | 392.65 |
| lf | Punto flotante doble | 392.65 |
| g | Uso del mas corto entre %e or %f | 392.65 |
| G | Uso del mas corto entre %E or %f | 392.65 |
| o | Octal con signo | 610 |
| u | Entero sin signo | 7235 |
| x | Entero hexadecimal sin signo | 7fa |
| X | Entero hexadecimal sin signo | 7FA |
| p | Dirección de apuntador | B800:0000 |

## Ejemplos

```
#include <stdio.h>
int main(){
   printf ("Caracteres: %c %c \n", 'a', 65);
   printf ("Decimales: %d %ld\n", 1977, 650000L);
   printf ("Con espacios: %10d \n", 1977);
   printf ("Con ceros: %010d \n", 1977);
   printf ("Con bases distintas: %d \n", 100);
   printf ("Con bases distintas: %x \n", 100);
   printf ("Con bases distintas: %o \n", 100);
   printf ("Con bases distintas: %#x \n", 100);
   printf ("Con bases distintas: %#o \n", 100);
   printf ("flotantes:%4.2f %+.0e %E\n",3.1416,3.1416,3.1416);
   printf ("%s \n", "Una cadena");
   return 0;
}
```

## Como hacer comentarios

```
// esto comenta una linea.
/*
    esto comenta
    un bloque de instrucciones

*/
```

## Operadores aritméticos

| Nombre del operador | Sintaxis |
|---|---|
| Más unitario | $+a$ |
| Suma | $a + b$ |
| Preincremento | $++a$ |
| Postincremento | $a++$ |
| Asignación con suma | $a += b$ |
| Menos unitario (negación) | $-a$ |
| Resta | $a - b$ |
| Predecremento | $--a$ |
| Postdecremento | $a--$ |
| Asignación con resta | $a -= b$ |
| Multiplicación | $a * b$ |
| Asignación con multiplicación | $a *= b$ |
| División | $a / b$ |
| Asignación con división | $a /= b$ |
| Módulo (Resto) | $a \% b$ |
| Asignación con módulo | $a \%= b$ |

## Operadores de comparación

| Nombre del operador | Sintaxis |
|---|---|
| Menor que | a < b |
| Menor o igual que | a <= b |
| Mayor que | a > b |
| Mayor o igual que | a >= b |
| No igual que | a != b |
| Igual que | a == b |
| Negación lógica | !a |
| AND lógico | a && b |
| OR lógico | a \|\| b |

## Estructuras de selección

C++ tiene dos estructuras de control para la selección, **if** (selección simple y binaria) y **switch** (selección múltiple).

## Sintaxis de la estructura de control **if**

```
if(<condicion>){
        <Intruccion>
        <Intruccion>
                ⋮
        <Intruccion>
}else{
        <Intruccion>
        <Intruccion>
                ⋮
        <Intruccion>
}
```

## Sintaxis de la estructura de control **switch**

```
switch (selector){
        case <opcion 1>:
                <bloque de instrucciones>
                break;
        case <opcion 2>:
                <bloque de instrucciones>
                break;
                        ⋮
        case <opcion n>:
                <bloque de instrucciones>
                break;
        default:
                <bloque de instrucciones>
}
```

## Ejemplo

```
switch (selector){
    case 1:
    case 2:
        printf("Salida para los casos 1 y 2\n");
        break;
    case 3:
        printf("Salida para el caso 3\n");
        break;
    default:
        printf("Salida para los restantes casos\n");
}
```

## Estructuras de iteración

La estructura de control o estructura de iteración **while**, nos permite repetir un bloque de instrucciones siempre y cuando se cumpla una condición, es decir, , en esta estructura de control el cuerpo de instrucciones se ejecuta mientras una condición permanezca como verdadera en el momento en que la condición se convierte en falsa el ciclo termina.

## Sintaxis de la estructura de control **while**

```
while (<condicion>){
        <instruccion>
        <instruccion>
                :
        <instruccion>
}
```

## Ejemplo

```
#include <stdio.h>
using namespace std;
int main(){
    int n=0;
    while (( n > 0)&&( n < 10 )){
        printf("El valor de n es: %d\n",n);
        n++;
    }
    return 0;
}
```

## DO WHILE

La estructura de control o la estructura de iteración **do while**, nos permite repetir un bloque de instrucciones siempre y cuando se cumpla una condición, es decir, , en esta estructura de control el cuerpo de instrucciones se ejecuta mientras una condición permanezca como verdadera en el momento en que la condición se convierte en falsa el ciclo termina. La diferencia es que primero realizara el bloque de instrucciones y al final verificara si la condición sigue siendo verdadera, es decir, almenos hará una ves el bloque de instrucciones.

## Sintaxis de la estructura de control **do while**

```
do{
        <instruccion>
        <instruccion>
              ⋮
        <instruccion>
} while (<condicion>);
```

## Importante

Notar que lleva ; al final, lo cual no ocurre en las otras estructuras de control.

## Ejemplo

```
#include <stdio.h>
using namespace std;
int main(){
    int n=0;
    do{
        printf("El valor de n es: %d\n",n);
        n++;
    }while (( n > 0)&&( n < 10 ));
    return 0;
}
```

## ¿Que es una función?

Una función es un conjunto de líneas de código que realizan una tarea específica y puede retornar un valor. Las funciones pueden tomar parámetros que modifiquen su funcionamiento. Las funciones son utilizadas para descomponer grandes problemas en tareas simples y para implementar operaciones que son comúnmente utilizadas durante un programa y de esta manera reducir la cantidad de código. Cuando una función es invocada se le pasa el control a la misma, una vez que esta finalizó con su tarea el control es devuelto al punto desde el cual la función fue llamada.

## Sintaxis

```
<tipo> [clase::] <nombre> ( [Parámetros] )
{
    cuerpo;
}
```

## Sintaxis

```
<tipo> [clase::] <nombre> ( [Parámetros] )
{
    cuerpo;
}
```

## Ejemplo

```
// regresar el cuadrado de un número
double cuadrado(double n)
{
    return n*n;
}
```

## Un arreglo en C++

Un **arreglo** es una colección ordenada de variables del mismo tipo. Las variables que pertenecen a un arreglo se conocen por el nombre de **elementos**.

El **término ordenado** significa que en la colección hay un **primer elemento**, un **segundo elemento**, un **tercer elemento**, y así sucesivamente.

Además, los elementos pueden a su vez organizarse en subgrupos llamadas **dimensiones**.

## Dimensiones

El subgrupo más pequeño posible se conoce como un arreglo de una dimensión. Un arreglo de dos dimensiones se subdivide en arreglos de una dimensión. Un arreglo de tres dimensiones se subdivide en arreglos de dos dimensiones los cuales a su vez se dividen en arreglos de una dimensión. Un arreglo de cuatro dimensiones se subdivide en arreglos de tres dimensiones los cuales a su vez se dividen en arreglos de dos dimensiones los cuales a su vez se dividen en arreglos de una dimensión. La misma idea se aplica en arreglos de más dimensiones.

## Sintaxis

```
<tipo> nombre_variable[longitud];
```

Con esto diremos que nombre_variable es un arreglo de longitud elementos del tipo <tipo>. Cabe destacar que longitud debe ser cualquier expresión entera constante mayor que cero.

## Sintaxis

```
<tipo> nombre_variable[longitud];
```

Con esto diremos que nombre_variable es un arreglo de longitud elementos del tipo <tipo>. Cabe destacar que longitud debe ser cualquier expresión entera constante mayor que cero.

## Asignación de un arreglo

```
nombre_variable[índice] = expresión del tipo <tipo>
```

Esta instrucción asigna el valor asociado de la expresión a la posición índice del arreglo nombre_variable.El índice debe ser una expresión del tipo entero en el rango [0, longitud-1]. Cabe destacar que C++ no chequea que el valor de la expresión sea menor a longitud, simplemente asigna el valor a esa posición de memoria como si formara parte del arreglo, pisando, de esta manera, otros datos que no forman parte del mismo, con lo que finalmente el programa no funciona correctamente.

## En resumen, un arreglo:

- No es una variable; es un grupo de variables conocidas como elementos

## En resumen, un arreglo:

- No es una variable; es un grupo de variables conocidas como elementos
- Cada elemento ocupa una posición dentro del grupo

## En resumen, un arreglo:

- No es una variable; es un grupo de variables conocidas como elementos
- Cada elemento ocupa una posición dentro del grupo
- Todos los elementos son del mismo tipo

## En resumen, un arreglo:

- No es una variable; es un grupo de variables conocidas como elementos
- Cada elemento ocupa una posición dentro del grupo
- Todos los elementos son del mismo tipo
- El nombre del arreglo indica donde se localiza el grupo en la memoria de la computadora

## En resumen, un arreglo:

- No es una variable; es un grupo de variables conocidas como elementos
- Cada elemento ocupa una posición dentro del grupo
- Todos los elementos son del mismo tipo
- El nombre del arreglo indica donde se localiza el grupo en la memoria de la computadora
- Los arreglos se clasifican de acuerdo a las dimensiones que tengan

## En resumen, un arreglo:

- No es una variable; es un grupo de variables conocidas como elementos
- Cada elemento ocupa una posición dentro del grupo
- Todos los elementos son del mismo tipo
- El nombre del arreglo indica donde se localiza el grupo en la memoria de la computadora
- Los arreglos se clasifican de acuerdo a las dimensiones que tengan
- Las dimensiones no tienen relación con el plano Cartesiano; nada que ver con matemática

## En resumen, un arreglo:

- No es una variable; es un grupo de variables conocidas como elementos
- Cada elemento ocupa una posición dentro del grupo
- Todos los elementos son del mismo tipo
- El nombre del arreglo indica donde se localiza el grupo en la memoria de la computadora
- Los arreglos se clasifican de acuerdo a las dimensiones que tengan
- Las dimensiones no tienen relación con el plano Cartesiano; nada que ver con matemática
- Las dimensiones indican como están organizados los elementos dentro del grupo

## En resumen, un arreglo:

- No es una variable; es un grupo de variables conocidas como elementos
- Cada elemento ocupa una posición dentro del grupo
- Todos los elementos son del mismo tipo
- El nombre del arreglo indica donde se localiza el grupo en la memoria de la computadora
- Los arreglos se clasifican de acuerdo a las dimensiones que tengan
- Las dimensiones no tienen relación con el plano Cartesiano; nada que ver con matemática
- Las dimensiones indican como están organizados los elementos dentro del grupo
- Los arreglos de dos dimensiones pueden visualizarse como tablas

## En resumen, un arreglo:

- No es una variable; es un grupo de variables conocidas como elementos
- Cada elemento ocupa una posición dentro del grupo
- Todos los elementos son del mismo tipo
- El nombre del arreglo indica donde se localiza el grupo en la memoria de la computadora
- Los arreglos se clasifican de acuerdo a las dimensiones que tengan
- Las dimensiones no tienen relación con el plano Cartesiano; nada que ver con matemática
- Las dimensiones indican como están organizados los elementos dentro del grupo
- Los arreglos de dos dimensiones pueden visualizarse como tablas
- Los valores que se guarden en el arreglo se almacenan en los elementos ya que los elementos son las variables

## Ejemplo #1 Arreglo de una dimensión

```
Declaración
    int a[3]; // forma una secuencia de tres elementos
Nombre del grupo
    a
Nombre de los elementos
    a[0] -> primer elemento
    a[1] -> segundo elemento
    a[2] -> tercer elemento
```

## Ejemplo #2 Arreglo de dos dimensión

```
Declaración
    char m[2][3]; // forma una tabla de dos filas y tres columnas
    // cada fila es un arreglo de una dimensión
    // la declaración indica que hay dos arreglos de una dimensión
Nombre del grupo
    m // indica la localización del grupo en la memoria
Nombre de las filas
    m[0] // primera fila -> indica la localización de la fila dentro del grupo
    m[1] // segunda fila -> indica la localización de la fila dentro del grupo
Nombre de los elementos
    m[0][0] // primer elemento
    m[0][1] // segundo elemento
    m[0][2] // tercer elemento
    m[1][0] // cuarto elemento
    m[1][1] // quinto elemento
    m[1][2] // sexto elemento
```

 Como Programar en C/C++, Deitel (Prentice Hall), 2da Edición.

 Programming Principles and Practice Using C++, Bjarne Stroustrup.

 http://www.codeblocks.org

 http://www.wxwidgets.org

 (O'Reilly) Practical C Programming (3rd Edition)

 http://www.cplusplus.com