LENGUAJE C++

Tema 2 – Elementos del Lenguaje

www.profmatiasg

Vectores Unidimensionales

- Los vectores (array o arreglo) unidimensionales son secuencias de valores del mismo tipo que se almacenan en localidades contiguas de memoria, según el orden del índice.

```
<tipo dato> <identificador>[tamaño];
```

- Ejemplo:

```
int valores[10];
float datos[5]={1.3, 2.8, 4.89, 0.0, 5.7};
int numeros[]={1, 5 , 9};
char nombre[]={'M', 'a', 't', 'i', 'a', 's',0};
```

- A diferencia de otros lenguajes los arrays en C++ comienzan por el elemento 0 y terminan en el n-1.

```
int a[6];
a[0]=23;
a[2]=a[0]+5;
for(int i=0;i<6;i++) cout << a[i] << endl;
```

Vectores Unidimensionales

- Un vector se identifica por su nombre, pero para el compilador este equivale a la dirección de memoria del primer elemento del vector, es decir:
- Ejemplo:

```
int num[50];
/*Para el compilador:
num  es igual a &num[0]
La dirección del elemento 0 */
```

- Para acceder a un elemento de un vector se debe especificar el nombre del vector, seguido de la posición (el índice) que ocupa dicho elemento dentro del vector entre corchetes.
- El almacenamiento de los elementos de un vector se determina en tiempo de compilación.
- La cantidad de memoria en bytes viene dada por:

```
bytes_totales = sizeof(tipo) * tamaño;
```

Matrices

- Un vector puede tener N dimensiones, dependiendo de las limitaciones de la memoria, a estos se los llama matrices y su declaración es la siguiente:

```
<tipo dato> <identificador>[dim1] [dim2]…[dimN];
```

- Ejemplo:

```
float vtasSemana[10][5]; //las ventas de los 10 vendedores x día de la semana.
double cubo[3][3][3];
int matrix[3][4];
```

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

Indice derecho columnas

Indice izquierdo filas

matrix [1][2]

Tipos de estructuras

- Una estructura es una agrupación de datos (posiblemente) heterogéneos (de distintos tipos), que se denomina bajo un único nombre, proporcionando un medio eficaz de mantener junta la información relacionada.
- Cada dato de una estructura es llamado campo o miembro.
- Cada campo de una estructura tiene un nombre que lo identifica y un tipo de datos asociado.
- Una estructura puede tener campos que sean a su vez estructuras.
- El lenguaje C++ tiene cuatro tipos de estructuras de datos:
  - Registro o estructura (struct).
  - Unión de campos (union).
  - Tipo enumerado (enum).
  - Clases (class).

Struct

- Al declarar una estructura se indica el nombre y tipo de todos sus campos.
- De manera opcional se puede dar nombre a la estructura y/o declarar variables de ese tipo de estructura (mínimo: alguna de las dos cosas).
- Es diferente declarar una estructura que una variable de tipo estructura. (Se puede hacer ambas cosas a la vez)

```
struct nombre_estr { campos };/*
    Sólo estruc. */
struct { campos } nombre_var;/*
    Sólo variable */
struct nombre_estr nombre_var2;/
    * Sólo variable */
struct nombre_estr2 { campos }
    nombre_var3; /* estruc. y
    variable */
```

Ejemplos:

```
struct ficha {
char nombre[20], apellido1[20];
int edad; float nota;
};
struct {int a, b;} dos_enteros;
struct ficha alumno1, alumno2;
struct complejo {float real,
    imaginaria;} nro_complejo;
```

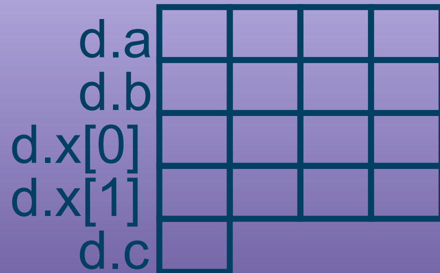
Ejemplo de Struct

```
#include <iostream>
using namespace std;

struct persona
{
    char   Nombre[60];
    char   Apellido[40];
    string Telefono;
    int    DNI;
    float Altura;
};

int main()
{
   persona matias;
   cout<<"Nombre:";
   cin>>matias.Nombre;
   cout<<"Apellido:";
   cin>>matias.Apellido;
   cout<<"Telefono:";
   cin>>matias.Telefono;
   cout<<"Numero de DNI:";
   cin>>matias.DNI;
   cout<<"Altura en metros:";
   cin>>matias.Altura;

   cout<<matias.Nombre<<endl;
   cout<<matias.Apellido<<endl;
   cout<<matias.Telefono<<endl;
   cout<<matias.DNI<<endl;
   cout<<matias.Altura<<endl;
   return 0;
}
```

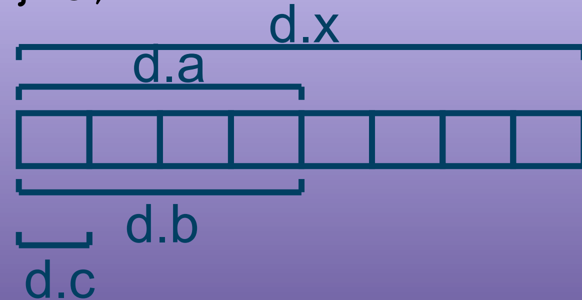
Union

- Un union es similar a un struct, pero todos los campos comparten la misma memoria.

```
struct datos
{
    int a,b;
    int x[2];
    char c;
} d;
```




```
union datos
{
    int a,b;
    int x[2];
    char c;
} d;
```



Ejemplo de Union

```
#include<iostream>
using namespace std;
union product
{
  int productoid;
  char nombre[20];
  float precio;
};
int main()
{
  union product obj;
  cout << "Ingrese ID de Producto: ";
  cin >> obj.productoid;
  cout << "Ingrese nombre del producto: ";
  cin >> obj.nombre;
  cout << "Ingrese precio del producto: ";
  cin >> obj.precio;
  cout << "ID Producto: " << obj.productoid << endl;
  cout << "Nombre: " << obj.nombre << endl;
  cout << "Precio: " << obj.precio;
  return 0;
}
```

```
#include<iostream>
using namespace std;
union union_product
{
  int productoid;
  char nombre[20];
  float precio;
};
struct struct_product
{
  int productoid;
  char nombre[20];
  float precio;
};
int main()
{
  union union_product union_obj;
  struct struct_product struct_obj;
  cout << "Tamaño de la union: " <<
sizeof(union_obj) << endl; //20
  cout << "Tamaño de la estructura:
" << sizeof(struct_obj);//28
  return 0;
}
```

Enum

- Las enumeraciones son conjuntos de constantes numéricas definidas por el usuario.

```
enum color {rojo, verde, azul} fondo;
color letras, borde=verde;

enum tipo_empleado {contratado=1,
                    temporal=2,
                    becario=3};
enum dias {lunes, martes, miercoles, jueves, viernes, sabado,domingo};
dias i, j, k;

i = martes;
for (j=lunes; j<=viernes; j++)
…;
```

Definición de nuevos tipos

- Las sentencias **typedef** se usan para definir nuevos tipos en base a tipos ya definidos:

```
#include <iostream>
using namespace std;

typedef unsigned char Tsexo;
typedef string Tnombre;
int main()
{
    Tsexo s;
    Tnombre n;
    cout << "ingrese su nombre: ";
    cin >> n;
    cout << "Sexo F/M/O: ";
    cin >> s;
    cout << n << endl << s;
    return 0;
}
```

Vectores de estructuras

- Pueden crearse vectores de estructuras, donde cada componente del vector será una estructura.
- Para declarar un vector de estructuras, se debe definir primero la estructura y luego declarar un variable vector de dicho tipo.
- Acceso: con los corchetes se indica el elemento del vector al que acceder, en el que luego puede indicarse un nombre de campo para acceder a un miembro de la estructura.

```
#include <iostream>
#define CANT 10
using namespace std;
struct datos
{    char nombre[30];
     char direccion[30];
     int n_empleado,edad;
     float sueldo;
};
int main()
{    datos persona[CANT];
     for (int i=0; i<CANT; i++)
     {    cout <<"Ingrese nro de empleado: "<< endl;
          cin >> persona[i].n_empleado;
          cout << "Ingrese  nombre: " << endl;
          cin >> persona[i].nombre;
          cout << "Ingrese edad: " << endl;
          cin >> persona[i].edad;
          cout << "Ingrese dirección: " << endl;
          cin >> persona[i].direccion;
          cout << "Ingrese sueldo: " << endl;
          cin >> persona[i].sueldo;
     }
     return 0;
}
```

Funciones

Además de las funciones de biblioteca, el programador puede definir sus propias funciones que realicen determinadas tareas.

- Función: Secuencia de instrucciones agrupadas bajo un mismo nombre que realizan una tarea determinada.
- La función se ejecutará tantas veces como se la llame mediante su nombre.
- Ventajas:
  - Facilita la reutilización de código, aumentando la productividad del programador.
  - Descomposición de un problema en subproblemas más sencillos → se disminuye la complejidad del problema.
  - El uso de funciones mejora la estructura del programa, haciéndolo más legible y entendible.
- Elementos de una función:
  - Nombre: identificador de la función por el cual es invocada.
  - Argumentos o parámetros: datos que se le pasan a la función para que opere.
  - Valor de retorno: resultado de haber procesado la función.

Declaración y definición

- Para poder hacer uso de una función es necesario que ésta esté definida o declarada con antelación.
  - Declaración de la función: Únicamente la cabecera o prototipo de la función (antes de main):

Tipo del resultado

```
int suma (int, int);
```

Tipos de los parámetros

  - Definición de la función: Todo el código de la función, su implementación (luego de main):

Necesario para devolver valor

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

Llamada a funciones

- Para que se ejecute una función, esta debe ser invocada mediante su nombre, seguido por los argumentos entre paréntesis.

```
nombre(arg1, arg2, ...);
```

- Los parámetros pueden ser variables, constantes, expresiones o el resultado de llamadas a otras funciones.
- Si la función invocada devuelve un valor, este podrá ser almacenado en una variable o utilizado como operando en alguna expresión.

```
variable = funcion();
```

Paso de parámetros a funciones

- En la definición de la función se especifica la lista de parámetros o argumentos que recibe y sus tipos.
- Puede no tener ningún argumento
- La sintaxis de la lista de argumentos es:

```
(tipo1 argumento1, tipo2 argumento2, ...)
```

- Al invocar (llamar) a la función se le deben pasar tantos argumentos como reciba y del tipo correcto.
- Existen dos formas de paso de parámetros:

- **Por valor**. Se pasa como parámetro un valor (puede ser una constante, una variable, el resultado de una operación …) este valor es utilizado en la función pero no es modificado su original ya que se realiza una copia a la variable asignada en el argumento.
- **Por referencia**. Se pasa como parámetro la dirección en memoria de una variable. Esta variable podrá ser modificada desde dentro de la función. Los vectores y matrices también se pasan por referencia. Si se usa el modificador `const` se realiza una referencia constante.

Paso de parámetros a funciones

PASO POR VALOR	PASO POR REFERENCIA

`void intercambiar(int x, int y)`
`{`
`int temp;`
`temp = x;`
`x = y;`
`y = temp;`
`}`
 	
`int main ()`
`{`
`int a=5, b=6;`
`intercambiar(a,b);`
`cout << "el valor de a: " << a << " el valor de b: " << b;`
`return 0;`
`}`
	`void intercambiar(int *px, int *py)`
	`{`
	`int temp;`
	`temp = *px;`
	`*px = *py;`
	`*py = temp;`
	`}`
	
	`int main ()`
	`{`
	`int a=5, b=6;`
	`intercambiar(&a,&b);`
	`cout << "el valor de a: " << a << " el valor de b: " << b;`
	`return 0;`
	`}`

Paso de parámetros por defecto

- En C++ es factible indicar un parámetro por defecto, tal que si no es pasado en la llamada a la función, se toma el valor asignado para su utilización dentro de la función.
- Los argumentos con valores por defecto deben estar al final de la lista.

```
double modulo(double x[], int n=3);
```

- En C++ se puede invocar así:
  - `v = modulo(x, n);`
  - `v = modulo(x);`

Sobrecarga de funciones

- En C++ se pueden definir en el mismo ámbito varias funciones con el mismo nombre siempre que se diferencien en el número o tipo de parámetros, a esto se llama **sobrecarga**.
- No pueden diferir sólo en el tipo de retorno. Tampoco en que un argumento se pase por valor en una función y por referencia en la otra.
  - `int max(int, int);`
  - `int max(int, int, int);`
  - `int max(int *, int);`
  - `int max(float, int);`
  - `int max(int, float);`
  - `float max(float, float);`

Entrada / Salida de datos

- Los flujos de bytes están definidos en la librería <iostream>
- La E/S en C++ ocurre en flujos, que son una secuencia de bytes que producen o consumen información.
  - Entrada: `cin >> [variable] >> [variable] >> ... ;`
  - Entrada solo 1 carácter: `cin.get(variable);`
  - Salida: `cout << [elemento] << [elemento] << ... ;`

```
#include <iostream>
#include <iomanip> //para los manipuladores
using namespace std;
int main()
{
    string nombre;
    float sueldo;
    cout << "ingrese su nombre: ";
    cin >> nombre;
    cout.left;
    cout << "Sueldo: ";
    cin >> sueldo;
    cout << "Nombre: " << nombre << endl;
    cout << "Sueldo: $" <<
               setprecision(4) << sueldo;
    cin.get(); //pausa
    return 0;
}
```

Manipuladores de E/S de datos

- Variables y/o métodos miembro que controlan el formato.
- Pueden tener argumentos (iomanip) o no (iostream).
- Sólo afectan al flujo al que se aplican.
- No guardan la configuración anterior.
  - **`endl`**: imprime un '\n' y se vacía el buffer de salida.
  - **`flush`**: vacía el buffer de salida.
  - **`setw(int w)`**: establece la anchura mínima de campo.
  - **`dec, hex, oct`**: convierten a decimal, hexa u octal.
  - **`setprecision(int p)`**: establece el número de cifras significativas
  - **`left`** o **`right`**: la salida se alinea a la izquierda o a la derecha.
  - **`Uppercase`**: los caracteres de formato aparecen en mayúsculas
  - **`Scientific`**: notación científica para coma flotante

Bibliografía & Licencia

- *Como programar en C++, 9na Ed*, Deitel, H.M. y Deitel, P.J., Pearson
- *Programación en C++, Un enfoque práctico*, Joyanes Aguilar, L., McGraw-Hill.
- *Thinking in C++, 2da Ed*, Bruce Eckel, Prentice Hall PTR.

- Este documento se encuentra bajo Licencia Creative Commons Attribution – NonCommercial - ShareAlike 4.0 International (CC BY-NC-SA 4.0), por la cual se permite su exhibición, distribución, copia y posibilita hacer obras derivadas a partir de la misma, siempre y cuando se cite la autoría del **Prof. Matías E. García** y sólo podrá distribuir la obra derivada resultante bajo una licencia idéntica a ésta.

- Autor:

Matías E. García

Prof. & Tec. en Informática Aplicada

www.profmatiasgarcia.com.ar

info@profmatiasgarcia.com.ar