FIEC

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

PROGRAMACIÓN APLICADA A LA AUTOMATIZACIÓN

M.Sc. Alexander Prieto León

Conferencia V

Unidad II Programación orientada a objetos y estructuras de datos aplicadas a automatización

2.1 Extensiones de C++ a C y memoria dinámica.

OBJETIVOS

Objetivos:

- Definir el lenguaje C++ como un super-conjunto del lenguaje C.
- Emplear algunas de las extensiones básicas de C++ a C.
- Definir el concepto de memoria dinámica.
- Saber operar con memoria dinámica.
- Manejar arreglos dinámicos.

BIBLIOGRAFÍA

Bibliografía:

- Deitel and Deitel. Como programar en C/C++. Segunda edición o superior.
- García de Jalón, J.; y otros. Aprenda C++ como si estuviera en primero. Universidad de Navarra.

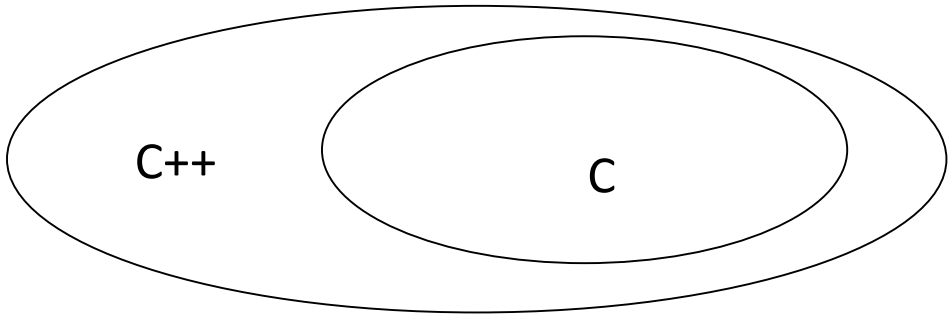
En la clase anterior:

- **En Matlab: Acceso a matrices.**

Syntax	Meaning
A(:,j)	is the j_{th} column of A.
A(i,:)	is the i_{th} row of A.
A(:,:)	is the equivalent two-dimensional array. For matrices this is the same as A.
A(j:k)	is A(j), A(j+1),...,A(k).
A(:,j:k)	is A(:,j), A(:,j+1),...,A(:,k).
A(:,:,k)	is the k_{th} page of three-dimensional array A.

C++ como superconjunto del lenguaje C

- ¿Cómo se puede definir el término superconjunto de un lenguaje o superlenguaje?



- Se extiende en cuanto:
  - a las palabras claves,
  - a la sintaxis y estructura.

Características:

- C++ fue desarrollado en los laboratorios Bell, al igual que el lenguaje C y originalmente fue llamado C con clases (1980).
- El nombre C++, consistente en el nombre del lenguaje C seguido del operador de incremento de C, significa que C++ es una versión mejorada y ampliada (incrementada) del lenguaje C.
- A partir de ahora se utilizarán ficheros con la extensión .cpp para los programas, en lugar de con la extensión .c.
- Todos los ejemplos realizados en lenguaje C en Introducción a la Computación, se pueden compilar en C++.

➢ Flujo de entrada/salida :

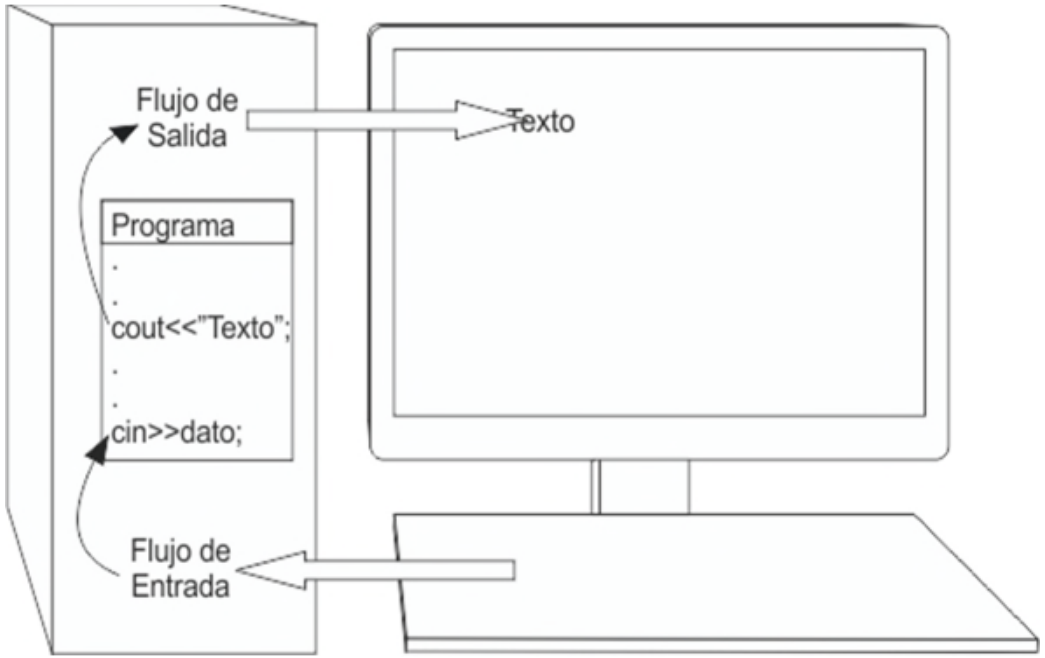
C y C++	C++
#include <stdio.h> … int n; printf("Teclee un número entero:"); scanf("%d",&n); printf("Ha tecleado el numero %d\n", n); …	#include <iostream.h> … int n; cout << "Teclee un número entero:" ; cin >> n; cout << "Ha tecleado el numero " << n << '\n'; …

- Con flotantes se puede establecer la precisión mediante, ejemplo:

  cout << setprecision(5);

Flujo de entrada/salida :

- La biblioteca **iostream** contiene los elementos de C++ relacionados con el manejo de los flujos de entrada y de salida. A través de la variable **cout** maneja el flujo de salida y a través de la variable **cin** maneja el flujo de entrada.



Declaraciones en C++:

- En C++ las declaraciones de variables puede aparecer en cualquier lugar siempre que aparezcan antes de ser utilizadas. (Las funciones siempre deben ser definidas fuera de otros módulos de programación, o sea, fuera del main() o programa principal y fuera del interior de otras funciones)

- El alcance de las variables locales en un bloque **{ }** se extiende desde que es declarada hasta que aparece la llave de cierre del bloque de forma similar.

Creación de nuevos tipos de datos:

- C++ proporciona la capacidad de crear tipos de datos definidos por el usuario mediante el uso de las palabras reservadas **enum**, **struct**, y algunas otras aun no tratadas.

En C y C++	En C++

typedef struct {
TipoElem info;
TNodo*prox;
} **TNodo;**
 	
TNodo MiNodo;
	struct TNodo {
	TipoElem info;
	TNodo* prox;
	};
	
	TNodo MiNodo;

➢ El símbolo & como seudónimo o alias:

Ejemplo:

```
int count = 1;      // Declara la variable count de tipo int con el valor inicial 1
int &c = count;     // Declara una  variable c que ocupa la misma localización de
                    // memoria que count y c es por tanto, un alias o seudónimo
                    // de count
```

Con estas declaraciones podemos referirnos a la variable declarada en la primera línea con cualquiera de los dos nombres: c ó count. Así, por ejemplo, **++c;** equivale a **++count;**

Parámetros por referencia:

- **En el lenguaje C no existen los parámetros por referencia**, pero se simulan pasando como parámetro un **puntero por valor**, el cual contiene la dirección de la variable cuyo contenido queremos que sea modificado por la función. (**Referencia simulada**). **Ejemplo:**

En C y C++	En C++

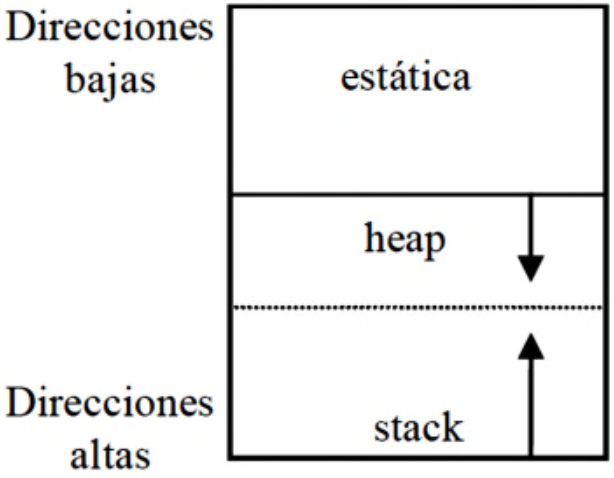
typdef struct {
unsigned norma_diaria;
} TTrabajador;
 	
void incrementa(TTrabajador *L)
{
L -> norma_diaria++;
}
 	
int GetNorma (TTrabajador L)
{
return L.norma_diaria;
}
…
TTrabajador MiTrabajador;
incrementa(&MiTrabajador);
	struct TTrabajador {
	unsigned norma_diaria;
	};
	
	void incrementa(TTrabajador &L)
	{
	L.norma_diaria++;
	}
	
	int GetNorma (const TTrabajador& L)
	{
	return L.norma_diaria;
	}
	…
	TTrabajador MiTrabajador;
	incrementa(MiTrabajador);

Memoria dinámica

Arreglos dinámicos.

¿En qué memoria se cargan las variables?

- La zona estática para datos, que permite almacenar variables globales durante la ejecución de un programa.
- El stack que permite almacenar los argumentos y variables locales durante la ejecución de las funciones.
- **El heap que permite almacenar variables adquiridas dinámicamente durante la ejecución de un programa.**



- **Problemas:**
  - La memoria asignada a los programas puede ser insuficiente en aplicaciones complejas.
  - No se puede cambiar el tamaño de los arreglos una vez creados.
  - El tamaño de los arreglos debe ser creado usando una constante.

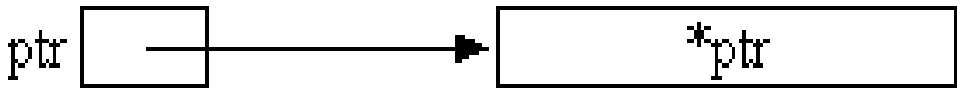
- **Solución:**
  - Asignación dinámica de la memoria.

El operador new.

- Considerando que T puede ser cualquier tipo de datos predefinido del lenguaje C/C++ o cualquiera de los tipos de datos que hemos definido hasta el momento.
- Para crear una variable dinámica de tipo T y obtener su dirección en un puntero que llamaremos ptr, podemos programar las siguientes instrucciones:

En C++
T *ptr ; ptr = **new** T ;

- Si existe suficiente memoria, una vez que se ejecute esta última instrucción, en la variable ptr se copiará la dirección de la variable de tipo T

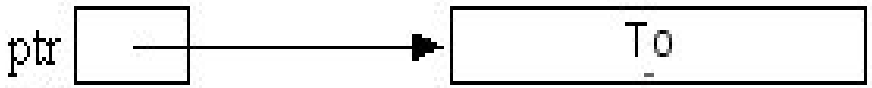


El operador new.

- Valor inicial:

ptr = new T (To);

- En este caso, al crearse la variable dinámica será inicializada con el valor To, por lo que el contenido de lo apuntado por ptr será el valor To, o sea:



- Observe que con una instrucción como:

T *ptr = new T (To) ; <=> **T* ptr = new T ;**
***ptr = To;**

El operador delete.

- Para devolver la memoria previamente asignada con new a una variable dinámica, si la dirección de esta variable está en un puntero que hemos llamado **ptr**, entonces lo haremos ejecutando una instrucción como la siguiente:

delete ptr;

Errores de asignación dinámica de memoria.

- No siempre la memoria dinámica que se pide estará disponible. (Heap reservado al comienzo del programa)

- **Se devuelve NULL en error de asignación.**

Ejemplo:

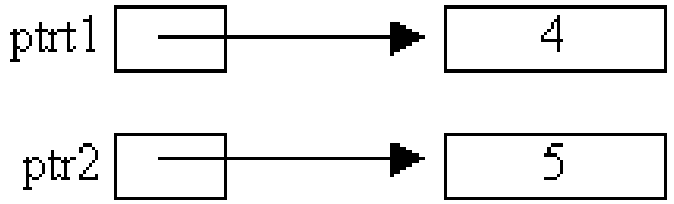
C++

```
int *ptr1 = new int ;
 if  (ptr1)
    *ptr1 = 4;
 else
    printf("Error de asignación dinámica\n")
```

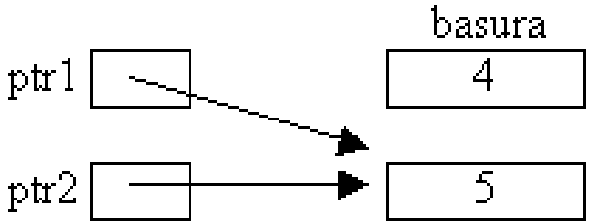
Las variables dinámicas y el operador de asignación predefinido.

- Si tenemos las siguientes instrucciones:

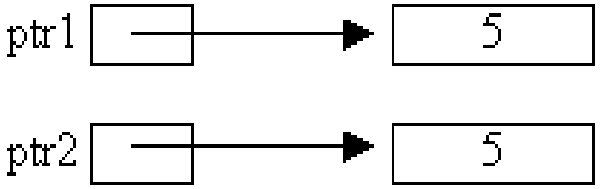
```
int *ptr1 = new int( 4 );
int *ptr2 = new int( 5 );
```



- Si asignamos un puntero a otro:



ptr1 = ptr2



***ptr1 = *ptr2 ó ptr1[0] = ptr2[0]**

- Lo correcto sería:

```
delete ptr1;
ptr1 =  ptr2;
```

Asignación dinámica de memoria para arreglos. Operadores new[] y delete [].

- Para un tipo de dato T, ***podemos crear un arreglo dinámico de elementos de tipo T*** *con:*

  ptr=new T [N];

donde **N** es una variable, expresión o constante entera que indica la cantidad máxima de elementos que puede contener el arreglo y **ptr** es un puntero al tipo de datos **T. ptr** apunta al primer elemento del arreglo.

- Para la destrucción del arreglo dinámico se utiliza:

  delete [] ptr;

Ejemplo:

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

int main(void)
{
  unsigned short int n;
  int *ptr;
  int *ptr1;

  cout<<"Deme la cantidad de elementos
del arreglo a crearse: ";
  cin>>n;

        ptr=new int[n];
        ptr1=ptr;

  if(ptr == NULL)
  {
     cout<<"No se pudo reservar memoria ";
     exit(0);
   }
```

```
for (int i=0; i<n; i++)
   {
      cout<<"\nValor de elemento "<< i <<": ";
      cin>>ptr[i];
   }

 float sum=0;
 for (int i=0;i<n;i++,ptr1++)
    sum+=*ptr1;
 cout<<"El promedio es: "<< (float)sum/n;

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

CONCLUSIONES