UNIAT

Estructuras de Datos

C/C++

Introducción

José Luis Alonzo Velázquez

UNIAT

Sesión 1

Evaluación

Tipo	Frecuencia	Porcentaje de la evaluación final
Tareas	$\approx$ 1 por semana	40 %
Practicas	$\approx$ 1 por semana	0 %
Proyecto	1	30 %
Exámenes	3 exámenes	30 %

Noticias y Material del Curso

Información sobre la clase centralizada en mi página web

`https://github.com/pepemxl/CursoEstructurasDeDatos2020`

1. tareas y clases en pdfs,
2. notificaciones diversas, errata sobre las tareas. . .

Por clase tendremos a lo sumo 2 descansos de aproximadamente 10 min.

Sobre la clase: Tareas

1. Tienen 1 semana para terminarlas (redondeado al día siguiente), por ejemplo:
2. jueves 7 octubre, 15h45 $->$ jueves 14 octubre, 23h59.
3. Penalidad para retraso: -1pt/día.
4. Formato de entrega de tareas via mail:
   ApellidoPaterno_PrimerNombre_Tarea##.zip
   ejemplo: Razo Rodriguez Daniel Alberto tarea 3
   el formato correspondiente seria: Razo_Daniel_Tarea03.zip

bool x	x es a Booleano (valor true and false).
char x	x is a character (usually 8 bits).
short x	x is a short int (usually 16 bits).
int x	x is the default integer type.
float x	x is a floating-point number.
double x	x is a double-precision floating-point number.
void *p	p is a pointer to raw memory.
const T x	x is a constant (immutable) version of T.
long T x	x is a long T.
unsigned T x	x is an unsigned T.
signed T x	x is a signed T.

Preprocesador

Un **preprocesador** es un programa separado que es invocado por el compilador antes de que comience la traducción real. Un preprocesador de este tipo puede eliminar los comentarios, incluir otros archivos y ejecutar sustituciones de macros.

Preprocesador

Un **preprocesador** es un programa separado que es invocado por el compilador antes de que comience la traducción real. Un preprocesador de este tipo puede eliminar los comentarios, incluir otros archivos y ejecutar sustituciones de macros.

El preprocesador de C++ es el preprocesador para el lenguaje de programación C. Es el primer programa invocado por el compilador y procesa directivas como #include, #define y #if. Estas directivas no son específicas de C. En realidad pueden ser usadas con cualquier tipo de archivo.

Etapas

El preprocesamiento es el primer paso en la etapa de compilación de un programa esta propiedad es única del compilador de C.
El preprocesador tiene más o menos su propio lenguaje el cual puede ser una herramienta muy poderosa para el programador.

Todas las directivas de preprocesador o comandos inician con un #.

Las que nos conciernen en este momento son **#include**, **#define**.

Ejemplo típico

```
1 #include <stdio.h>
2
3 int main (void){
4     printf (" Hola Mundo !\n");
5     return 0;
6 }
```

Ejemplo típico

```
1 #include <stdio.h>
2
3 int main (void){
4     printf (" Hola Mundo !\n");
5     return 0;
6 }
```

En este ejemplo se incluye todo lo que hay en la librería stdio.h, así mismo nosotros podemos crear librerías, e incluirlas de esta manera solo que en lugar de los signos <> usaremos comillas dobles “mi_libreria.h”.

Ejemplo

```
1 #include <stdio.h>
2 #include "mi.c"
```

mi.c

```
1
2 int main(){
3     for (int n=0; n < 10 ; n++){
4         printf("El valor de n es: %d\n",n);
5     }
6     return 0;
7 }
```

Cuando se indica <archivo> se le dice al compilador que busque donde están los archivos incluidos o "include" del sistema. Usualmente los sistemas con UNIX guardan los archivos en el directorio */usr/include*.

Si se usa la forma "archivo" es buscado en el directorio actual, es decir, donde el programa esta siendo ejecutado.

Los archivos incluidos usualmente contienen los prototipos de las funciones y las declaraciones de los archivos cabecera (header files) y no tienen código de C (algoritmos).

#define

La directiva **#define** se usa para definir constantes o cualquier sustitución de macro.

#define

La directiva **#define** se usa para definir constantes o cualquier sustitución de macro.

Su formato es el siguiente:

#define <nombre de macro> <nombre de reemplazo>

#define

La directiva **#define** se usa para definir constantes o cualquier sustitución de macro.

Su formato es el siguiente:

#define <nombre de macro> <nombre de reemplazo>

Ejemplo

```
#define FALSO 0
#define VERDADERO 1
```

#define

La directiva **#define** se usa para definir constantes o cualquier sustitución de macro.

Su formato es el siguiente:

#define <nombre de macro> <nombre de reemplazo>

Ejemplo

#define FALSO 0
#define VERDADERO 1

Observación

La directiva #define tiene otra poderosa característica: el nombre de macro puede tener argumentos. Cada vez que el compilador encuentra el nombre de macro, los argumentos reales encontrados

Hay dos tipos de macros: las que son como objetos y las que son como funciones. Las que se asemejan a funciones toman parámetros mientras que las que se asemejan a objetos no. La forma de definir un identificador como una macro de cada tipo es, respectivamente:

Sintaxis de **define**

#define <identificador> <lista de tokens a reemplazar>
#define <identificador>(<lista de parámetros>) <lista de tokens a reemplazar>

Ejemplo de define

```
1 #include <stdio.h>
2
3 #define PI 3.14159
4
5 int main(){
6     printf("El valor de PI es: %d\n",PI);
7     return 0;
8 }
```

Ejemplo de define

```
1 #include <stdio.h>
2
3 #define f(x) x*x+2
4
5 int main(){
6     printf("El valor de f(3) es: %d\n",f(3));
7     return 0;
8 }
```

Ejemplo típico de define

```
1  #include <stdio.h>
2
3  #define REP(i,n) for(int i=0;i<n;i++)
4
5  int main(){
6      REP(i,10){
7          printf("El valor de n es: %d\n",n);
8      }
9      return 0;
10 }
```

Un arreglo en C/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. Más adelante veremos esto mas a detalle...

Sintaxis

```
1    <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

```
1 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

Para crear arreglos en C++, hay que indicar:

1. el tipo de los elementos (ejemplo, int, char, double, bool o un tipo definido por el programador)
2. el nombre del arreglo
3. la cantidad de dimensiones y sus tamaños; cada dimensión comienza con el signo [seguido por el tamaño de la dimensión y termina con el signo]

Para crear un arreglo de una dimensión, el formato es el siguiente:

```
1 <tipo de los elementos> <nombre del arreglo> [ <tamaño
      primera dimensión> ]
```

Para crear arreglos en C++, hay que indicar:

1. el tipo de los elementos (ejemplo, int, char, double, bool o un tipo definido por el programador)
2. el nombre del arreglo
3. la cantidad de dimensiones y sus tamaños; cada dimensión comienza con el signo [seguido por el tamaño de la dimensión y termina con el signo]

Para crear un arreglo de una dimensión, el formato es el siguiente:

```
1 <tipo de los elementos> <nombre del arreglo> [ <tamaño
      primera dimensión> ]
```

Cantidad de elementos en un arreglo

Para determinar la cantidad de elementos en un arreglo, hay que multiplicar el tamaño de cada una de las dimensiones indicados en

Observaciones

Para nombrar un elemento en un arreglo hay que indicar el nombre del arreglo, seguido de tantas parejas de corchetes [ ] como dimensiones declaradas. Dentro de cada pareja de corchetes tiene que haber un índice. La combinación de los índices indica la posición del elemento dentro del grupo. El valor mínimo para un índice es 0. El valor máximo es uno menos que el tamaño de la dimensión correspondiente en la declaración del arreglo.

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

```
1 Declaración
2     char m[2][3]; // forma una tabla de dos filas y tres
       columnas
3     // cada fila es un arreglo de una dimensión
4     // la declaración indica que hay dos arreglos de una
       dimensión
5 Nombre del grupo
6     m // indica la localización del grupo en la memoria
7 Nombre de las filas
8     m[0] // primera fila -> indica la localización de la
       fila dentro del grupo
9     m[1] // segunda fila -> indica la localización de la
       fila dentro del grupo
```

Ejemplo #2 Continuación

```
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
```

Ejemplo #3 Arreglo de tres dimensión

```
1  Declaración
2      double w[2][2][3]; // forma dos tablas de dos filas
       y tres columnas
3      // cada fila es una secuencia de tres elementos
4      // cada fila es un arreglo de una dimensión
5      // la declaración indica que hay dos arreglos de dos
        dimensiones
6      // los arreglos de dos dimensiones tienen a su vez
       dos arreglos de una dimensión
7  Nombre del grupo
8      w // indica la localización del grupo en la memoria
9  Nombre de las tablas
10     w[0] // primera tabla // indica la localización de
       la tabla dentro del grupo
11     w[1] // segunda tabla. indica la localización de la
       tabla dentro del grupo
```

Ejemplo #3 Continuación

```
Nombre de las filas
    w[0][0] // primera tabla, primera fila
    w[0][1] // primera tabla, segunda fila
    w[1][0] // segunda tabla, primera fila
    w[1][1] // segunda tabla, segunda fila
Nombre de los elementos
    w[0][0][0] // primer elemento
    w[0][0][1] // segundo elemento
    w[0][0][2] // tercer elemento
    w[0][1][0] // cuarto elemento
    w[0][1][1] // quinto elemento
    w[0][1][2] // sexto elemento
    w[1][0][0] // séptimo elemento
    w[1][0][1] // octavo elemento
    w[1][0][2] // noveno elemento
    w[1][1][0] // décimo elemento
    w[1][1][1] // undécimo elemento
```

Problema para clase

Crear un arreglo con 5 elementos e imprimir en pantalla todos los elementos del arreglo.

Cadenas

A diferencia de otros lenguajes de programación que emplean un tipo denominado cadena string para manipular un conjunto de símbolos, en C, se debe simular mediante un arreglo de caracteres, en donde la terminación de la cadena se debe indicar con nulo. Un nulo se especifica como '\0'. Por lo anterior, cuando se declare un arreglo de caracteres se debe considerar un carácter adicional a la cadena más larga que se vaya a guardar. Por ejemplo, si se quiere declarar un arreglo cadena que guarde una cadena de diez caracteres, se hará como:

```
char cadena[11];
```

Inicializaciones

Se pueden hacer también inicializaciones de arreglos de caracteres en donde automáticamente C asigna el carácter nulo al final de la cadena, de la siguiente forma:
char nombre_arr[tam]=“cadena”;

Inicializaciones

Se pueden hacer también inicializaciones de arreglos de caracteres en donde automáticamente C asigna el carácter nulo al final de la cadena, de la siguiente forma:
char nombre_arr[tam]=“cadena”;

Ejemplo

Por ejemplo, el siguiente fragmento inicializa cadena con “hola”:
char cadena[5]=“hola”;
El código anterior es equivalente a:
char cadena[5]='h','o','l','a','\0';

Para asignar la entrada estándar a una cadena se puede usar la función scanf con la opción %s (observar que no se requiere usar el operador &), de igual forma para mostrarlo en la salida estándar.

Ejemplo

```
1 int main(){
2     char nombre[15], apellidos[30];
3
4     printf("Introduce tu nombre: ");
5     scanf("%s",nombre);
6     printf("Introduce tus apellidos: ");
7     scanf("%s",apellidos);
8     printf("Usted es %s %s\n",nombre,apellidos);
9 }
```

El lenguaje C no maneja cadenas de caracteres, como se hace con enteros o flotantes, por lo que lo siguiente no es válido:

```
1 int main(){
2     char nombre[40], apellidos[40], completo[80];
3
4      nombre="Angelina";                    /* Ilegal */
5      apellidos="Jolie";              /* Ilegal */
6      completo="Actriz"+nombre+appellidos; /* Ilegal */
7 }
```

Ejemplos de declaración de cadenas de caracteres

```
1 char *cadena_hola="Hola";//Igual al siguiente
2 char cadena_hola[]="Hola";//Igual al siguiente
3 char otro_hola[]={'H','o','l','a','\0'};
4 char vector[]={'H','o','l','a'}; /*Un vector de 4
      elementos,
5     con los elementos 'H','o','l' y 'a' */
6 char espacio_cadena[1024]="Una cadena en C";
7 char cadena_vacia[]="";
```

Cómo vimos anteriormente al declarar un arreglo se define la cantidad de elementos que puede contener, en el caso de las cadenas se debe tener en cuenta el espacio adicional necesario para el \0. Viendo el ejemplo, tanto cadena_hola y otro_hola tienen un largo 5 y cadena_vacia tiene un largo de 1.

Buscar las siguientes funciones y ver para que sirven, i.e., crear ejemplos de cada una de ellas.

- strcat
- strcpy
- strlen
- strcmp

Entre las funciones que provee la biblioteca estándar de C, las más importantes son:

- largo = strlen(cadena) // Para obtener el largo de una cadena

Entre las funciones que provee la biblioteca estándar de C, las más importantes son:

- largo = strlen(cadena) // Para obtener el largo de una cadena
- strcpy(destino, origen) // Copia el contenido de origen en destino // destino debe ser lo suficientemente grande

Entre las funciones que provee la biblioteca estándar de C, las más importantes son:

- largo = strlen(cadena) // Para obtener el largo de una cadena
- strcpy(destino, origen) // Copia el contenido de origen en destino // destino debe ser lo suficientemente grande
- strcat(destino, origen) // Agrega el contenido de origen al final de destino // destino debe ser lo suficientemente grander

Entre las funciones que provee la biblioteca estándar de C, las más importantes son:

- largo = strlen(cadena) // Para obtener el largo de una cadena
- strcpy(destino, origen) // Copia el contenido de origen en destino // destino debe ser lo suficientemente grande
- strcat(destino, origen) // Agrega el contenido de origen al final de destino // destino debe ser lo suficientemente grander
- resultado = strcmp(cadena1, cadena2) // Compara dos cadenas // devuelve un valor menor, igual o mayor que 0 según si cadena1 es menor, // igual o mayor que cadena2, respectivamente.

Entre las funciones que provee la biblioteca estándar de C, las más importantes son:

- largo = strlen(cadena) // Para obtener el largo de una cadena
- strcpy(destino, origen) // Copia el contenido de origen en destino // destino debe ser lo suficientemente grande
- strcat(destino, origen) // Agrega el contenido de origen al final de destino // destino debe ser lo suficientemente grander
- resultado = strcmp(cadena1, cadena2) // Compara dos cadenas // devuelve un valor menor, igual o mayor que 0 según si cadena1 es menor, // igual o mayor que cadena2, respectivamente.
- posicion = strchr(cadena, caracter) // Devuelve la posición en memoria de la primer // aparición de caracter dentro de cadena

Entre las funciones que provee la biblioteca estándar de C, las más importantes son:

- largo = strlen(cadena) // Para obtener el largo de una cadena
- strcpy(destino, origen) // Copia el contenido de origen en destino // destino debe ser lo suficientemente grande
- strcat(destino, origen) // Agrega el contenido de origen al final de destino // destino debe ser lo suficientemente grander
- resultado = strcmp(cadena1, cadena2) // Compara dos cadenas // devuelve un valor menor, igual o mayor que 0 según si cadena1 es menor, // igual o mayor que cadena2, respectivamente.
- posicion = strchr(cadena, caracter) // Devuelve la posición en memoria de la primer // aparición de caracter dentro de cadena
- posicion = strstr(cadena,subcadena) // Devuelve la posición en memoria de la primer // aparición de subcadena dentro de cadena

Ejemplo: Función que compara cadenas
Librería necesaria string.h

```
1  bool compara_cadenas(char s1[],char s2[]){
2      int n=strlen(s1);
3      int m=strlen(s2);
4      if(n!=m){
5          return false;
6      }
7      for(int i=0;i<n;i++){
8          if(s1[i]!=s2[i]){
9              return false;
10         }
11     }
12     return true;
13 }
```

Bubble Sort

El método de ordenación de burbuja (Bubble Sort en inglés) es un sencillo algoritmo de ordenamiento. Funciona revisando cada elemento de la lista que va a ser ordenada con el siguiente, intercambiándolos de posición si están en el orden equivocado. Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que la lista está ordenada. Este algoritmo obtiene su nombre de la forma con la que suben por la lista los elementos durante los intercambios, como si fueran pequeñas "burbujas". También es conocido como el método del intercambio directo. Dado que solo usa comparaciones para operar elementos, se lo considera un algoritmo de comparación, siendo el más sencillo de implementar.

Seudocódigo

```
1  procedimiento bubbleSort( A:lista de items a ordenar )
2      n = longitud(A)
3      repetir
4          bandera_cambio = false
5          for i = 1 hasta n-1
6              if A[i-1] > A[i] entonces
7                  intercambia( A[i-1], A[i] )
8                  bandera_cambio = true
9              end if
10         end for
11     hasta que no halla cambio
12 end procedimiento
```

 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