TP Nº 6: Estructuras de datos compuestas

Objetivos:

1. Entender el concepto de Estructuras de datos.
2. Reconocer la clasificación de los tipos de datos.
3. Conocer las posibilidades y limitaciones de los tipos de datos que se utilizan en programación.
4. Ser capaces de seleccionar los tipos de datos adecuados para los datos que se desea modelar con variables
5. Aprender a crear tipos de datos definidos por el usuario.
6. Incorporar el concepto de cadena de caracteres y aprender a manipular las funciones relacionadas
7. Aprender a crear y manipular datos con estructura o datos de tipo registro.
8. Consolidar el método de resolución de problemas y la modularización de las soluciones.

Resolver los ejercicios, respetando en cada caso, las siguientes consignas:

- Aplicar el método de resolución de problemas: definir el objetivo, pensar una estrategia y descomponerla en partes.
- Codificar la solución en lenguaje C.
- Realizar la verificación con casos de prueba representativos del problema. Tener en cuenta que es necesario identificar de antemano cuáles son los resultados esperados para cada caso de prueba.

1. TIPOS DE DATOS DEFINIDOS POR EL USUARIO

En C es posible crear nuevos **tipos de datos personalizados** a través de la palabra reservada **typedef**.

El tipo personalizado que se declara en un **typedef** aparece luego del tipo de dato primitivo que toma como base. Por ejemplo, la declaración:

```
typedef int tEntero;
```

hace del nombre **tEntero** un sinónimo del tipo de dato **int**.

Una vez creado este nuevo tipo de dato, puede ser utilizado para declarar variables de este tipo. Por ejemplo, la siguiente instrucción:

```
tEntero len, maxlen;
```

declara las variables len y maxlen del tipo tEntero.

El tipo **tEntero** puede utilizarse en declaraciones, casts (conversión de tipos), etc., exactamente de la misma manera en que lo podría hacer int

EJERCICIOS

1.1. Escribir un programa que disponga de la declaración de un tipo de dato *tNumeroReal* que permita renombrar al tipo de dato *float*. Declarar una variable del tipo *tNumeroReal*, asignarle un valor e imprimir por pantalla su contenido.

1.2. Escribir un programa que disponga de la declaración de un tipo de dato *tCaracter* que permita renombrar al tipo de dato *char*. Declarar una variable del tipo *tCaracter*, asignarle un valor e imprimir por pantalla su contenido.

Un **tipo de dato compuesto o estructurado** está construido en base a los tipos de datos primitivos.

Permite almacenar un conjunto de elementos bajo una estructura particular, darle un único nombre, pero con la posibilidad de acceder en forma individual a cada componente.

Ejemplo de datos estructurados son:

→ Cadenas
→ Registros
→ Arreglos

2. TIPO DE DATO CADENA

La información de texto se almacena en constantes y en variables de tipo **cadena**.

Una cadena de caracteres se define como un conjunto ordenado de caracteres (letras, números, signos de puntuación, espacios en blanco, etc.).

Por ejemplo: “La vida es bella!”.

Internamente una cadena de caracteres se almacena en posiciones consecutivas de memoria, asignando un byte para almacenar el código ASCII de cada uno de los caracteres que la componen, es decir un carácter por posición de memoria.

Una variable de cadena se declara con el tipo de dato char seguida de la longitud máxima de la cadena de caracteres entre corchetes.

Sintaxis en C:

char <identificador> [<longitud máxima>];

Ejemplo: char apellidoYnombre [50];

Es posible crear un tipo de dato personalizado para las cadenas de caracteres, como el siguiente:

```
typedef char string[50];
```

y luego utilizar ese tipo para declarar variables:

string apellidoYnombre;

Es importante entender cómo se comportan las cadenas cuando se utilizan como argumentos en funciones. Al pasar una cadena de caracteres como parámetro a una función, en realidad se pasa la dirección del primer carácter de esa cadena.
Esto significa que, aunque no se use una sintaxis de puntero explícita, la cadena se está pasando por referencia de manera implícita.
Por eso, las funciones pueden acceder y modificar el contenido original de la cadena, siempre que esta no sea una constante. En este contexto, las declaraciones **char cadena[]** y **char *cadena** son equivalentes.

La biblioteca estándar de ANSI C consta de 24 ficheros cabecera que pueden ser incluidos en un proyecto de programación con una simple directiva. A continuación, se muestran las cabeceras que contienen funciones de manejo de **cadenas**.

Funciones de manejo de cadena <string.h>		Sintaxis
strcpy	Copiar una cadena de caracteres (fuente) en el lugar que ocupaba otra (destino).	strcpy(<variable_destino>, cadena_fuente>)
strcat	Copia una cadena (fuente) en otra (destino) sin destruir ésta. Una detrás de la otra.	strcat(<cadena_destino>, cadena_fuente>)
strcmp	Compara dos cadenas. Si son iguales, devuelve 0. Un número negativo si cadena1 es menor (alfabéticamente) que cadena2, y un número positivo si es mayor.	strcmp(cadena1, cadena2)
strncmp	Compara n caracteres entre dos cadenas.	strncmp(cadena1, cadena2, n).
strlen	Devuelve un valor entero correspondiente a la longitud de la cadena.	strlen(cadena)
Funciones de conversión de minúscula/mayúscula <ctype.h>		**Sintaxis**
tolower	Convierte un dato carácter a minúscula.	tolower (letra)
toupper	Convierte un dato carácter a mayúsculas.	toupper(letra)

scanf

En C, la función *scanf* es útil para leer datos desde la entrada estándar (generalmente el teclado) pero su uso con cadenas de caracteres puede ser limitado si solo utilizamos el especificador %s, ya que este solo lee hasta el primer espacio en blanco, impidiendo capturar frases o textos con espacios.

Para superar esta limitación, se utilizan los *scansets* (especificadores como *%[^\n]*) que permiten leer secuencias de caracteres que incluyen espacios y otros símbolos hasta un carácter específico, como el salto de línea.

Dado **char str [10]**; algunos *scansets* son los siguientes:

```
scanf("%s", str);      // Lee una palabra (hasta el primer espacio)

scanf("%9s", str);     // Lee hasta 9 caracteres o hasta el primer espacio espacio

scanf("%[^\n]", str);  // Lee hasta un salto de línea (incluye espacios)

scanf("%9[^\n]", str);  // Lee hasta 9 caracteres o hasta un salto de línea (incluye espacios)
```

fgets

Otra función que nos permite ingresar cadenas de texto por teclado es *fgets* que pertenece a la librería *stdio.h*. Esta función se usa para leer una línea completa de texto (incluyendo espacios) desde la entrada estándar (teclado) y guardarla en una cadena de caracteres.

Además, permite especificar el tamaño máximo a leer para evitar desbordamientos. Por ejemplo:

```c
#include <stdio.h>
int main() {
   char texto[50];
   printf("Ingrese una línea de texto: ");

   // Lee hasta 49 caracteres o hasta '\n'
   fgets(texto, sizeof(texto), stdin);

   printf("Texto ingresado: %s", texto);
   return 0;
}
```

2.1. Para emitir los certificados de aprobación de un curso de Python, se dispone de una planilla en la que se informan en columnas diferentes el nombre y el apellido de los participantes. Escribir un programa que lea dos datos correspondientes al nombre y apellido, respectivamente. Utilice una función que reciba como parámetro el nombre y el apellido y concatene en una sola cadena el apellido y nombre, separados por coma. Mostrar en pantalla la nueva cadena y la longitud de esta. El programa termina cuando el usuario indique una condición de fin de ingreso de datos.

2.2. El formato de presentación de un artículo científico exige que el título no exceda la cantidad de 20 palabras y esté escrito en mayúsculas. Escribir un programa que ingrese por teclado el título del trabajo (el título termina con un punto). Escriba una función que reciba como parámetro el título y retorne la cantidad de palabras que contiene. Además, utilice otra función que convierta el título a mayúsculas. Al final muestre en pantalla el título en mayúscula e indique con una leyenda si excede las 20 palabras.

2.3. Escribir un programa que permita ingresar por teclado una frase (la frase termina con un punto) y cuente e informe la cantidad de veces que aparece una vocal en la cadena (por ejemplo, cuántas veces aparece la vocal “a”). La vocal se ingresa por teclado. Resuelva el ejercicio utilizando al menos una función.

2.4. Escribir un programa que permita ingresar por teclado una frase y detecte y cuente las letras en mayúsculas. Recuerde que en ASCII las mayúsculas toman valores decimales comprendidos entre 65 y 90.

2.5. Escribir un programa que determine si una cadena es un palíndromo, es decir, si se puede leer de izquierda a derecha y viceversa. Por ejemplo: “ANILINA”, “RADAR”.

3. REGISTROS

Los **registros** o **datos con estructura** permiten almacenar diferentes tipos de datos bajo una misma variable.

Un registro es un “contenedor” de datos de diferentes tipos.

Cada dato que tiene dentro es conocido como campo.

Una estructura de registro se declara con la palabra reservada **struct**, cuyo formato o sintaxis en C es **el siguiente:**

```c
struct {
        tipo1 dato1;
        tipo2 dato2;
        /* …otros campos del registro */
 } nombre_de_variable
```

Ejemplos:

Declarar una variable registro:

```
struct {
        short dia;
        short mes;
        int anio;
    } fecha;
```

Y para asignar valores:

```
fecha.dia = 11;
fecha.mes = 05;
fecha.anio = 2019;
```

Y para acceder a un valor específico del registro basta con:

```
int anioActual = fecha.anio;
```

EJERCICIOS

Para la resolución de estos ejercicios, considerar los tipos de datos y su tamaño en bytes indicados en la siguiente tabla:

Tabla 1: Tipos de datos y su tamaño en bytes

	Tamaño en bytes
Entero corto (short)	2
Entero (int)	4
Real (Float)	4
Real (Double)	8
Carácter (*char*)	1
Cadena	Se indica la cantidad de bytes entre corchetes

3.1. Definir los campos necesarios para modelar un registro de alumnos de la FaCENA (Facultad de Ciencias Exactas y Naturales y Agrimensura) que serán utilizados por el sistema SIU Guaraní. Luego, determinar el tipo y el tamaño de cada campo. Por último, calcular la longitud del registro.

La longitud de un registro se determina sumando la cantidad de bytes de cada campo. Para ello, se debe tener en cuenta los tipos de datos usuales en C y, en el caso de los numéricos, la cantidad de bytes que utiliza cada uno, considerando los valores de la Tabla 1.

Solución ejemplo:

DNI	Apellido	Nombre	Edad	Código de la localidad de procedencia	Código de la carrera que cursa

DNI → Identificador: **dni**
Tipo de dato: **long**
Longitud: **4**

Apellido → Identificador: **apellido**
Tipo de dato: **char[50]**
Longitud: **50**

Nombre → Identificador: **nombre**
Tipo de dato: **char[50]**
Longitud: **50**

Edad → Identificador: **edad**
Tipo de dato: **short**
Longitud: **2**

Código de Localidad de procedencia → Identificador: **codLocalidad**
Tipo de dato: **short**
Longitud: **2**

Código de la carrera que cursa → Identificador: **codCarrera**
Tipo de dato: **short**
Longitud: **2**

Longitud del registro = 4 + 50 + 50 + 2 + 2 + 2 = **110**

3.2. Declarar una estructura compuesta que permita modelar un registro de películas y series que serán utilizados por NETFLIX para llevar el control sobre el interés de los clientes (No más de 7 campos). Luego:

- Determinar el tipo y el tamaño de cada campo
- Calcular la longitud del registro
- Asignarle valores, y mostrar por pantalla.

3.3. Crear una estructura compuesta para almacenar datos de super héroes: nombre de super héroe, nombre terrestre, poder, universo (editorial). Asignarle valores, y mostrar por pantalla.

3.4. Un supermercado dispone de la siguiente información relacionada con sus productos: código de producto, descripción, precio de costo. Desea obtener el precio sugerido, que se calcula aplicando el 30% al precio de costo. Mostrar luego todos los datos del producto, desde la estructura compuesta.

Nota:

Utilizar la declaración de una estructura compuesta que permita modelar los datos de los productos. El precio sugerido también debe formar parte de la estructura diseñada