Taller de Lenguajes I – 2024
Programador Universitario / Licenciatura en Informática / Ingeniería en Informática
Trabajo Práctico N° 3

OBJETIVOS

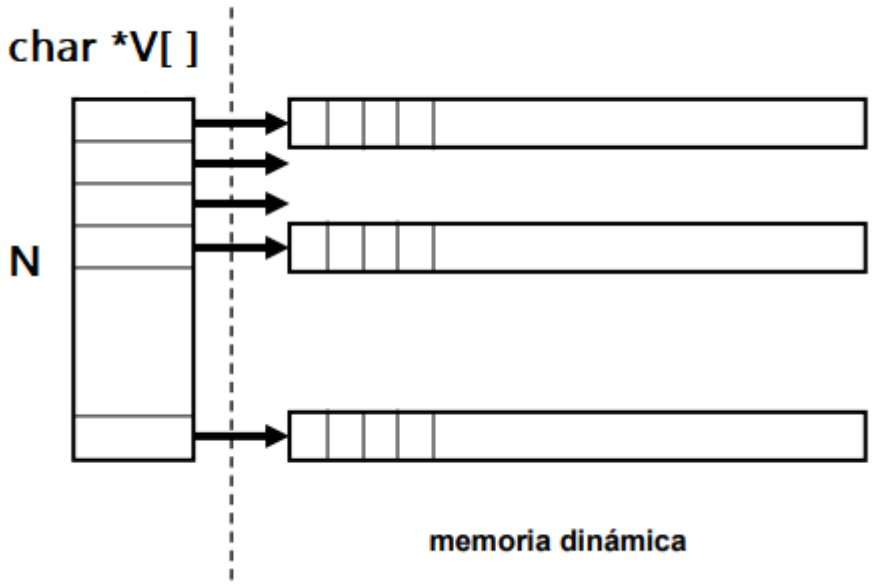
- Memoria dinámica y reserva de memoria
- Estructuras de datos dinámicos

1) Copie el siguiente enlace en su navegador: https://tinyurl.com/tl1-2024-tp3 para crear el repositorio donde subirá el **Trabajo Práctico Nro. 3**, realice los pasos ya aprendidos para *clonar* el repositorio en su máquina y poder comenzar a trabajar de forma *local*.

   Nota. No se olvide de incluir el archivo .gitignore en la raíz del repositorio para excluir los archivos .exe, .obj y .tds del mismo.

2) Una empresa necesita simular la producción de los próximos 5 años para tal fin necesita generar una matriz (estática) de 5x12 donde cada fila corresponde a un año y cada columna es un mes. Ud. debe realizar las siguientes tareas:
   a. Cargue dicha matriz con valores aleatorios entre 10000 y 50000.
   b. Muestre por pantalla los valores cargados
   c. Saque el promedio de ganancia por año y muestrelos por pantalla
   d. Obtenga el valor máximo y el valor mínimo obtenido informando el “año” y el “mes” de cuándo ocurrió.

3) Escriba un programa que solicite 5 nombres, los cargue en un vector de punteros char y una vez cargados sean listados por pantalla (Todo implementando reserva dinámica de memoria) (Lea las notas 1 y 2)



Nota 1: La librería más común para trabajar con cadenas de caracteres es string.h .Algunas de sus funciones más importantes son:

• `strlen(<cadena>)` : Devuelve la longitud de la cadena sin tomar en cuenta el caracter de final de cadena.
• `strcpy(<cadena_destino>, <cadena_origen>)` : Copia el contenido de *<cadena_origen>* en *<cadena_destino>*.
• `strcat(<cadena_destino>, <cadena_origen>)` : Concatena el contenido de *<cadena_origen>* al final de <>.

• `strcmp(<cadena1>, <cadena2>)` : Compara las dos cadenas y devuelve un 0 si las dos cadenas son iguales, un número negativo si <*cadena1*> es menor que <*cadena2*> y un número positivo (mayor que cero) si <*cadena1*> es mayor que <*cadena2*>

Nota 2: Cuando trabaje con cadena de caracteres (punteros a char) tenga en cuenta que primero tiene que hacer la reserva de memoria según la cantidad de caracteres que quiera guardar y luego utilizando la función *strcpy()* para copiar el contenido. Cuando se cargue por pantalla se puede valer de una variable auxiliar (Buff) para leer los datos y luego con ella cargar la variable asociada

Nota 3:

Para cargar un arreglo de caracteres con una “frase” use el comando gets(<cadena>)
Tenga en cuenta que para para mostrar en pantalla un array de caracteres podemos hacerlo de diversas maneras:

- Dentro de un bucle, desde el primer caracter (indice 0) hasta el último carácter (lo que nos devuelve la función strlen):

```
for(i=0; i<strlen(cadena); i++)
    printf("%c",cadena[i]);
```

- Utilizando el carácter de conversión %s:

```
printf("%s",cadena);
```

- Utilizando puts:

```
puts(cadena);
```

Ejemplos resueltos

```
//Ejemplos resueltos
///Ejemplo 1 - Trabajando  con cadena de caracteres
//cargar cadena con gets()
// imprimir frases por pantalla con printf() y puts()

#include <stdio.h>
#include <stdlib.h>
int main() {
    char frase [100];
    char texto[] = "Ud escribió:";
    printf ("Escriba una frase: ") ;
    gets (frase);
    printf ("%s\n", texto) ;
    printf ("%s\n", frase) ;//forma 1 de escribir
    puts (frase);//forma 2 de escribir
    return 0;
}
//Ejemplo 2 - Trabajando  con cadena de caracteres con reserva de
memoria

#include <stdio.h>
#include <stdlib.h>

int main() {
 char *Buff;
   Buff= (char *) malloc(100*sizeof(char));
```

```
   printf ("Ingrese una frase\n");
   gets(Buff);
   printf ("%s\n", Buff) ;//forma 1 de escribir
   puts (Buff);//forma 2 de escribir
   free(Buff);
   return 0;
}

//Ejemplo 3 - reserva dinámica de memoria con variable auxiliar

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
 char *Buff; //variable auxiliar
 char *Nombre;
 char *Apellido;

   Buff= (char *) malloc(100*sizeof(char));
   printf("Ingrese sus nombres: ");
   gets(Buff);

   Nombre= (char *) malloc((strlen(Buff)+1)*sizeof(char));
   strcpy(Nombre,Buff);

   printf("Ingrese sus Apellidos: ");
   gets(Buff);
   Apellido= (char *) malloc((strlen(Buff)+1)*sizeof(char));
   strcpy(Apellido,Buff);

   printf("escribiendo usando printf\n");
   printf ("%s\n", Buff) ;
   printf ("%s\n", Nombre) ;
   printf ("%s\n", Apellido) ;

   printf("escribiendo usando puts\n");
   puts (Buff);
   puts (Nombre);
   puts (Apellido);
//libero la memoria reservada en todos los punteros
  free(Buff);
  free(Nombre);
  free(Apellido);

  return 0;
}
```

4) Considere la siguiente situación: En una distribuidora necesita implementar la carga de productos de sus preventistas, los cuales recolectan los productos que sus clientes van necesitando. El sistema que se utiliza en la empresa es desarrollado por equipos de programadores donde cada equipo se encarga de una tarea específica. Usted forma parte del equipo de programación que se encargará de hacer el módulo para los preventistas:

Tareas:

Cada preventista puede visitar hasta 5 clientes, los cuales por cuestiones operativas solo puede pedir hasta 10 productos

Las estructuras de datos necesarias son las siguientes:

```
char *TiposProductos[]={"Galletas","Snack","Cigarrillos","Caramelos","Bebidas"};
struct Producto {
            int ProductoID;     //Numerado en ciclo iterativo
            int Cantidad;     // entre 1 y 10
            char *TipoProducto;  // Algún valor del arreglo TiposProductos
            float PrecioUnitario; // entre 10 - 100
};
struct Cliente {
       int ClienteID;                    // Numerado en el ciclo iterativo
       char *NombreCliente;     // Ingresado por usuario
       int CantidadProductosAPedir; // (aleatorio entre 1 y 5)
       Producto *Productos //El tamaño de este arreglo depende de la variable
                                 // "CantidadProductosAPedir"
};
```

i) Desarrollar una interfaz por consola donde se solicite al usuario la cantidad de clientes.

ii) Solicitar al usuario la carga de los clientes creados dinámicamente en el paso anterior.

iii) A medida que se dé de alta cada cliente, Generar aleatoriamente la cantidad de productos asociados al cliente y sus características.

Ej: producto cargado nro. 2

```
Producto {
    ProductoID=2
    Cantidad = 1;
    *TipoProducto = "Snack";
    PrecioUnitario = 100;
  }
```

iv) Implemente una función que calcule el costo total de un producto. Esta función debe recibir como parámetro el producto y devolver el resultado de calcular la Cantidad por el PrecioUnitario.

v) Mostrar por pantalla todo lo cargado. Incluyendo un total a pagar por cliente (sumatoria del costo de todos los productos)

5) Modifique el ejercicio nro 3 para que, en lugar de ingresar un número fijo de nombres, el usuario pueda indicar previamente la cantidad de nombres que ingresará a continuación.

6) ¿Qué es una wiki?¿Para que suele usarse? Investigue, cómo usar la wiki del github. Cargue en la wiki de su repositorio todas las respuestas.

7) En la Wiki de su repositorio: explique con sus propias palabras cuál es la diferencia entre memoria dinámica y estática. ¿Cuáles son las características de cada una? ¿En qué casos se debe usar cada una?