Práctica 5: Repaso final de C

Cátedra Programación II Octubre 2023

1. Gestión de la memoria y punteros

Ejercicio 1. Escriba un programa que declare algunas variables locales e imprima las direcciones de memoria de las mismas. Pruebe declarar un arreglo de caracteres y verifique que las direcciones de sus elementos son contiguas.

Ejercicio 2. Implemente una función `void setzerozero(int [])` que ponga en cero el primer elemento del arreglo recibido. Verifique desde la función llamante que efectivamente modifica este valor. ¿Por qué pasa esto? ¿No se llama a la función por valor?

Ejercicio 3. Implemente una función `setin(int *)` que toma un puntero a un entero y reemplaza el entero apuntado por un `1` si el entero apuntado era diferente a `0`, y `0` en caso contrario.

Ejercicio 4. Implemente una función `void swap(int *, int *)` que, dados dos punteros a variables, intercambie el contenido de las variables apuntadas.

Ejercicio 5. Explique el tipo de la función `malloc`, ¿qué valor retorna la función en caso de que no pueda reservar el espacio solicitado?

Ejercicio 6. Implemente una función `char *getnewline(void)` que ingrese una linea por teclado (hasta `\n`) y devuelva un puntero a la cadena ingresada.

Ejercicio 7. Escriba un programa que reserve un espacio de memoria de 100 bytes y luego lo libere dos veces. ¿Se produce algún error?

Ejercicio 8. Implemente las siguientes funciones que reciben como argumento punteros a función:

- **a)** `int apply(int (*)(int), int)` que toma un puntero a función y un entero, aplica la función al entero y retorna el valor dado.
- **b)** `void applyin(int (*)(int), int *)` que toma un puntero a función y un puntero a un entero y reemplaza el entero apuntado por el valor de ejecutar la función apuntada sobre el valor apuntado.
- **c)** `void recorre(VisitorFunc, int [], int)` que toma un puntero a una función, un arreglo de enteros y su longitud y aplica la función a cada elemento del arreglo. `VisitorFunc` está definido por `typedef void (*VisitorFunc)(int)`.
- **d)** Pruebe las funciones anteriores pasándoles como parámetro las siguientes funciones:

  i. Para **a** y **b**:
  ```
  int sucesor (int n) {
      return n+1;
  }
  ```

  ii. Para **c**:
  ```
  void imprimir (int n) {
      printf("%d \n", n);
  }
  ```

Ejercicio 9. Para cada uno de los programas que se listan a continuación, elija la salida que supone que ocurrirá al ejecutarlo, luego compruebe si estaba en lo cierto. ¿Por qué se obtuvo cada resultado?

a)

```c
#include<stdio.h>

void nullify(int* a){
  a = NULL;
}

int main(){
  int a[67];
  a[0] = 123;
  printf("%d\n", a[0]);
  nullify(a);
  printf("%d\n", a[0]);
  return 0;
}
```

Resultado:

I. 123
   Segmentation fault

II. 123
    123

III. 123
     basura

IV. Segmentation fault

V. Ninguna de las anteriores

b)

```c
#include<stdio.h>

int main(){
  char *ptr = "hola mundo";
  ptr[0] = 's';
  printf("%s\n", ptr);
  return 0;
}
```

Resultado:

I. hola mundo

II. sola mundo

III. Segmentation fault

IV. Ninguna de las anteriores

2. Estructuras y punteros

Ejercicio 10. Suponga que tiene que implementar un juego que requiere utilizar un mazo de cartas. Implemente los siguientes puntos para crear una representación del mismo que luego

pueda utilizarse para el juego:

a) Defina una estructura `Carta` para representar una carta de la baraja española (represente el palo con una enumeración).
b) Cree un arreglo de 48 cartas, llámelo mazo y llénelo con las cartas correspondientes.
c) Implemente una función `struct Carta azar(struct Carta[], int)` que reciba un mazo y la longitud del mismo y devuelva una carta al azar del mazo pasado.

Ejercicio 11. Implemente una estructura para representar puntos en el plano y una función `medio` que, dados dos de estos puntos, calcule el punto medio.

Ejercicio 12. Cree una representación para una agenda de contactos:

a) Defina una estructura `Contacto` que tenga como campos: una cadena para el nombre de una persona, una cadena para el número de teléfono y un entero sin signo para llevar la edad de la persona.
b) Implemente una función `struct Contacto crear_contacto()` que pida por teclado los datos pertinentes, rellene una estructura `Contacto` y la devuelva.
c) Implemente una función `void actualizar_edad(struct Contacto *)` que, dado un puntero a una estructura `Contacto`, pida una nueva edad por teclado y actualice la estructura.
d) Defina una estructura `Agenda` que almacene un arreglo de estructuras `Contacto` y un entero para llevar la cantidad de estructuras completadas del arreglo.
e) Escriba un programa que permita:
   i. Dar de alta un contacto: `void alta_contacto(struct Agenda *);` pide por teclado los datos del contacto a agregar (función `crear_contacto`) e inserta el resultado en la agenda.
   ii. Modificar la edad de un contacto: `void modificar_edad(struct Agenda *);` pide por teclado el nombre del contacto a modificar para identificarlo y luego actualiza su edad (función `actualizar_edad`).
   iii. Ver los datos de los contactos cargados: `void imprimir_contactos(struct Agenda *);`.
f) Implemente una función `double prom(struct Agenda *);` que devuelva el promedio de la edad de los contactos dentro de la agenda.