Algoritmos y Estructuras de Datos I - Laboratorio
Proyecto 4

Programación Imperativa en C

El objetivo de este proyecto es:

- Extender los conceptos relacionados con el desarrollo de programas en lenguaje C en base al formalismo visto en el teórico/práctico de la materia.
- Introducir el uso de la librería `assert.h` para garantizar el cumplimiento de estados.
- Familiarizarse con el uso de Arreglos en el lenguaje "C";
- Definir tipos abstractos nuevos básicos, utilizando el comando `struct`.

1. Lenguaje "C"

A lo largo de todo el proyecto, se utilizará el lenguaje C, y algunas herramientas como el `GDB: The GNU Project Debugger`, para ayudar a la comprensión del concepto de estado y del paradigma imperativo.

En el caso del lenguaje "C", para poder ejecutar un programa, lo vamos a tener que "compilar", y de esa manera generamos un archivo binario que podrá ser ejecutado en la computadora.

Cómo compilar en C:

Para compilar un archivo `.c` escribir en la terminal:

```
$> gcc -Wall -Wextra -std=c99 miarchivo.c -o miprograma
```

Para ejecutar escribir:

```
$> ./miprograma
```

Para compilar para gdb se debe agregar el flag -g al momento de compilar .c escribir en la terminal:

```
$> gcc -Wall -Wextra -std=c99 -g miarchivo.c -o miprograma
```

Ejercicios

1. (Funciones en C, Assert) Escribí los programas siguientes:

   a) `ejercicio01.c` que lee una variable `n` de tipo `int` e imprime por pantalla "hola" `n` veces. En esta ocasión el programa debe utilizar dos funciones a definir (además de la función `main`). Programá en un archivo `.c` la función de prototipo:

   ```
   void holaHasta(int n)
   ```

que dado un `int n`, imprime `n` veces "Hola". (Usar una bucle `while`). La función `main` pide un número en la entrada antes de llamar `holaHasta` (¿qué función podes usar ya implementada?). Usá la función `assert` (ver teórico) para chequear que $x > 0$.

b) Los ejemplos que siguen a continuación han sido vistos en el teórico práctico. Para cada uno de ellos, se debe obtener la pre y post condición y la derivación. Luego, se debe traducir a Lenguaje C el programa y las pre y post condiciones utilizando el comando `assert`.

- (Mínimo) Cálculo del mínimo entre dos variables enteras `x` e `y`. El programa en C se debe llamar `minimo.c`.
- (Valor Absoluto) Especificar un programa que calcule el valor absoluto de un número entero. Verificar que el programa es correcto, y luego traducir el programa a C en un archivo nuevo llamado `absoluto.c`.
- (Intercambio de variables) Considera el siguiente programa que intercambia los valores de dos variables `x` e `y` de tipo `Int`.

```
z := x;
x := y;
y := z;
```

Nota: En todos los casos el programa en C, debe solicitar los valores de las variables de entrada, e imprimir el resultado para que lo pueda ver el usuario.

2. (Asignaciones múltiples) Considerar las siguientes asignaciones múltiples

```
{Pre: x = X, y = Y}
x, y := x + 1, x + y
{Post: x = X + 1, y = X + Y}
```

```
{Pre: x = X, y = Y, z = Z}
x, y, z := y, y + x + z, y + x
{Post: x = Y, y = Y + X + Z, z = Y + X}
```

a) Escribir un programa equivalente que sólo use secuencias de asignaciones simples.

b) Traducir los programas resultantes a C en archivos nuevos llamados `multiple1.c` y `multiple2.c` respectivamente.

Recordar: Como C no tiene asignaciones múltiples, siempre será necesario traducirlas primero a secuencias de asignaciones simples.

3. (Función suma_hasta) Crear un archivo llamado `suma_hasta.c`, que contenga la función

```
int suma_hasta(int N)
```

que toma un número entero `N` como argumento, y devuelve la suma de los primeros `N` naturales. En la función `main` pedir al usuario que ingrese el entero `N`, si es negativo imprimir un mensaje de error, y si es no negativo imprimir el resultado devuelto por `suma_hasta`.

Ayuda: La función puede hacer un ciclo o directamente usar la fórmula de Gauss.

4. (Algoritmo de la división) Crear un archivo llamado `division.c` que contenga la siguiente función:

```
struct div_t division(int x, int y){
...
}
```

donde la estructura `div_t` se define como

```
struct div_t {
  int cociente;
  int resto;
};
```

Esta función recibe dos enteros no negativos (divisor no nulo) y devuelve el cociente junto con el resto de la división entera. En la función `main` pedir al usuario los dos números enteros, imprimir un mensaje de error si el divisor es cero, o imprimir tanto el cociente como el resto en otro caso.

5. (Arreglos, entrada-salida) Escribir un programa que solicite el ingreso de un arreglo de enteros `int a[]` y luego imprime por pantalla. El programa debe utilizar dos nuevas funciones con prototipo además de la función `main`:

   - una que dado un arreglo y su tamaño, solicita los valores del arreglo y los devuelve en el mismo arreglo `int a[]` con prototipo:

     ```
     void pedirArreglo(int a[], int n_max)
     ```

   - otra que imprime cada uno de los valores de arreglo `int a[]`.

     ```
     void imprimeArreglo(int a[], int n_max)
     ```

6. (Arreglos, Función sumatoria). Hacer un programa en un archivo con nombre `sumatoria.c` que contenga la función

   ```
   int sumatoria(int a[], int tam)
   ```

   que recibe un arreglo y su tamaño como argumento, y devuelve la suma de sus elementos. En la función `main` pedir los datos del arreglo al usuario asumiendo un tamaño constante.

7. (positivos). Hacer un programa en un archivo `positivos.c` que contenga las siguientes funciones:

   ```
   bool existe_positivo(int a[], int tam)
   bool todos_positivos(int a[], int tam)
   ```

   Estas funciones reciben un arreglo y su tamaño como argumento, y devuelven respectivamente verdadero si existe un número positivo en el arreglo o si todos los elementos son positivos. En la función `main` pedirle al usuario los elementos del arreglo (asumiendo un tamaño constante) y luego permitirle elegir qué función ejecutar.

8. (Procedimiento intercambio). Hacer un programa en el archivo nuevo `intercambio_arreglos.c` que contenga la siguiente función:

   ```
   void intercambiar(int a[], int tam, int i, int j)
   ```

   que recibe un arreglo, su tamaño y dos posiciones como argumento, e intercambia los elementos del arreglo en dichas posiciones. En la función `main` pedirle al usuario que ingrese los elementos del arreglo y las posiciones, chequear que las posiciones esten en el rango correcto y luego imprimir en pantalla el arreglo modificado.

9. (Función cuantos). Hacer un programa en un archivo nuevo `cuantos.c` que calcula cuántos elementos menores, iguales y mayores a un número hay en un arreglo. La función tiene el siguiente tipo:

   ```
   struct comp_t cuantos(int a[], int tam, int elem)
   ```

   donde la estructura `comp_t` se define como sigue:

   ```
   struct comp_t {
     int menores;
     int iguales;
     int mayores;
   };
   ```

   La función toma un arreglo, su tamaño y un entero, y devuelve una estructura con tres enteros que respectivamente indican cuántos elementos menores, iguales o mayores al argumento hay en el arreglo. La función `cuantos` debe contener un único ciclo.

10. (Función stats). Hacer un programa en un archivo nuevo `stats.c`, que calcula el mínimo, el máximo, y el promedio de un arreglo no vacío de números flotantes (tipo `float`). La función tiene el siguiente tipo:

```
struct datos_t stats(float a[], int tam)
```

donde la estructura `datos_t` se define como sigue:

```
struct datos_t {
  float maximo;
  float minimo;
  float promedio;
};
```

La función pedida debe implementarse con un único ciclo. En la función `main` pedir al usuario los datos del arreglo e imprimir en pantalla los tres valores devueltos por la función.

11. (Función nesimo_primo) En un archivo nuevo `primo.c` hacer una función

```
int nesimo_primo(int N)
```

que devuelve el n-ésimo primo.

   a) En la función `main` pedir al usuario que ingrese el entero `n`, si es negativo imprimir un mensaje de error, y si es no negativo imprimir el resultado devuelto por `nesimo_primo`.

   b) Modificar la función `main`, para que al ingresar un valor negativo, solicite un nuevo valor hasta que se ingrese un `n` no negativo.

12. (Punto estrella). Se define el tipo `persona_t` como sigue:

```
typedef struct _persona {
    char *nombre;
    int edad;
    float altura;
    float peso;
} persona_t;
```

Definir las siguientes funciones:

```
float peso_promedio(persona_t arr[], unsigned int longitud);
persona_t persona_de_mayor_edad(persona_t arr[], unsigned int longitud);
persona_t persona_de_menor_altura(persona_t arr[], unsigned int longitud);
```

Las tres funciones toman como argumento un arreglo de personas y su longitud. Devuelven respectivamente el promedio de peso, la persona de mayor edad y la persona de menor altura que se encuentra en el arreglo. Ayuda: Para probar las funciones, hacer una función `main` como la siguiente:

```
int main(void) {
    persona_t p1 = {"Paola", 21, 1.85, 75};
    persona_t p2 = {"Luis", 54, 1.75, 69};
    persona_t p3 = {"Julio", 40, 1.70, 80};
    unsigned int longitud = 3;
    persona_t arr[] = {p1, p2, p3};
    printf("El peso promedio es %f\n", peso_promedio(arr, longitud));
    persona_t p = persona_de_mayor_edad(arr, longitud);
    printf("El nombre de la persona con mayor edad es %s\n", p.nombre);
    p = persona_de_menor_altura(arr, longitud);
    printf("El nombre de la persona con menor altura es %s\n", p.nombre);
    return 0;
}
```