Práctico 4
Derivación de Programas Imperativos

Algoritmos y Estructuras de Datos I
2^{do} cuatrimestre 2024

Laboratorio 1 (Intercambio de variables) Traducir e implementar en lenguaje C el siguiente programa que intercambia los valores de dos variables `x` e `y` de tipo `Int`.

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

Laboratorio 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) Para cada uno escribir un programa equivalente que sólo use secuencias de asignaciones simples.

b) Traducir los programas resultantes a C, respectivamente.

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

Laboratorio 3 (vocales) Crear un archivo `vocales.c` que contenga la función:

```
bool es_vocal(char letra)
```

que dado el caracter `letra` devuelve `true` si es una vocal y `false` en caso contrario. En la función `main` se le debe solicitar al usuario que ingrese un caracter y luego se debe mostrar un mensaje que indique si dicho caracter es una vocal o no según el resultado de la función `es_vocal()`. Tener en cuenta vocales mayúsculas y minúsculas.

NOTA: Definir una función que pida un caracter análoga a `pedir_entero()` pero para el tipo `char`.

NOTA: Recordar usar `%c` en vez de `%d` en el uso de `scanf()` y `printf()` para obtener / mostrar caracteres al usuario.

1. Para cada uno de los siguientes problemas, especificá utilizando pre y post condición y, derivá un programa que lo resuelva o definí un programa y demostrá que el programa es correcto.

   - (Mínimo) Calcular el mínimo entre dos variables enteras `x` e `y`.
   - (Valor Absoluto) Calcular el valor absoluto de un número entero.

Laboratorio 4 Traducir a Lenguaje C los programas definidos en el ejercicio 1. Incorporar al programa las pre y post condiciones utilizando el comando `assert`. 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. Especificar y derivar un programa que calcule el factorial de un número.

3. (Función suma_hasta) Dado $N \geq 0$ queremos un programa que devuelva la suma de los primeros n naturales. Especificá el problema utilizando pre y post condición y, derivá un programa que lo resuelva o definí un programa y demostrá que el programa es correcto.

Laboratorio 5 Definir en C una programa 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 (ver ejercicio 3). 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`.

Laboratorio 6 (Procedimiento intercambio). Hacer un programa en el archivo nuevo `intercambio_arreglos.c` que contenga la siguiente función:

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

que recibe un tamaño máximo de arreglo, un arreglo 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 estén en el rango correcto y luego imprimir en pantalla el arreglo modificado.

Laboratorio 7 (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 además de la función `main`:

- una que dado un tamaño máximo de arreglo y el arreglo, solicita los valores para el arreglo y los devuelve, guardándolos en el mismo arreglo `int a[]`; función con prototipo (también conocido como signatura o firma):

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

- otra que imprime cada uno de los valores del arreglo `int a[]`, de prototipo:

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

4. (Suma de los elementos de un arreglo) Dado un arreglo de enteros, especificar y derivar un programa que calcule la suma de todos los elementos del arreglo.

Laboratorio 8 (Arreglos, Función sumatoria). Hacer un programa en un archivo con nombre `sumatoria.c` que contenga la función

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

que recibe un tamaño máximo de arreglo y un arreglo como argumento, y devuelve la suma de los elementos del arreglo. En la función `main` pedir los datos del arreglo al usuario asumiendo un tamaño constante previamente establecido (en tiempo de compilacilación).

5. Sea A un arreglo de enteros.
   a) Especificar y derivar un programa que determine si todos los elementos de A son mayores a 0.
   b) Especificar y derivar un programa que determine si algún elemento de A es mayor a 0.
6. Especificar y derivar un programa que calcule la suma de los elementos pares de un arreglo de enteros.
7. Dado un arreglo $A : array[0, N)\ of\ Num$ con $N \geq 0$, contar cuántas veces coinciden dos elementos:

Const $N : Int, A : array\ [0, N)\ of\ Int$;
Var $r : Int$;
$\{P : N \geq 0\}$
S
$\{Q :\ r = \langle N\, i, j\ : 0 \leq i < j < N :\ A.i = A.j\ \rangle\}$

8. Dado un arreglo $A : array[0, N)\ of\ Num$ con $N \geq 0$, determinar si hay dos elementos que suman 8:

   Const $N : Int, A : array\ [0, N)\ of\ Int$;
   Var $r : Bool$;
   $\{P : N \geq 0\}$
   S
   $\{Q :\ r = \langle\, \exists\, i, j\ : 0 \leq i < j < N :\ A.i + A.j = 8\ \rangle\}$

9. Especificar y derivar: Dado un arreglo $a : array[0, N)\ of\ Num$ con $N \geq 0$ determinar si alguno de sus elementos es igual a la suma de los anteriores. Usar fortalecimiento de invariante.

10. Especificar y derivar: Dado un arreglo $a : array[0, N)\ of\ Num$ con $N \geq 0$ determinar si sus elementos son iguales al factorial de la posición. Usar fortalecimiento de invariante.

11. Especificar y derivar un programa imperativo que calcule Fibonacci de un número dado. Usar fortalecimiento de invariante.

12. (Máxima diferencia) Dado un arreglo de enteros, calcular la máxima diferencia entre dos de sus elemento (en orden, el primero menos el segundo).

    La especificación del programa es:

    Const $N : Int$;
    Var $a : array[0, N)\ of\ Int; r : Int$;
    $\{P : N \geq 2\}$
    S
    $\{Q : r = \langle \text{Max}\ \ p, q\ : 0 \leq p < q < N :\ a.p - a.q\ \rangle\}$

13. (Segmento de suma máxima) Dado un arreglo de enteros, calcular la suma del segmento de suma máxima del arreglo.

    La especificación del programa es:

    Const $N : Int$;
    Var $a : array[0, N)\ of\ Int; r : Int$;
    $\{P : N \geq 0\}$
    S
    $\{Q : r = \langle \text{Max}\ \ p, q\ : 0 \leq p \leq q \leq N :\ sum.p.q\ \rangle$
    $\quad [\![sum.p.q = \langle \sum i\ : p \leq i < q :\ a.i\ \rangle]\!]\}$

14. Dada la siguiente especificación:

    Const $M : Int, A : array[0, M)\ of\ Int$;
    Var $r : Int$;
    $\{P : M \geq 0\}$
    S
    $\{Q : r = \langle N\, p, q\ : 0 \leq p < q < M :\ A.p * A.q \geq 0\ \rangle\}$

    Decir en palabras qué hace el programa y derivarlo.

15. (Algoritmo de la división) Dados dos números, hay que encontrar el cociente y el resto de la división entera entre ellos.

    Ayuda: Para enteros $x \geq 0$ e $y > 0$, el cociente q y el resto r de la división entera de x por y están caracterizados por $x = q * y + r \wedge 0 \leq r \wedge r < y$. Por lo tanto, debemos derivar un programa S que satisfaga

    Const $x, y : Int$;
    Var $q, r : Int$;
    $\{P :\ \ x \geq 0 \wedge y > 0\}$ (precondición)
    S
    $\{Q :\ \ x = q * y + r \wedge 0 \leq r \wedge r < y\}$ (postcondición)

Laboratorio 9

(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.

16. Sea $N \geq 0$, especificar y derivar un programa que calcule el menor natural x que satisface $x^3 + x \geq N$.

    Ayuda: Especifique el problema (con pre y poscondición) de forma que en la poscondición queden conjunciones y asi poder utilizar la técnica "tomar término de la conjunción". Para ellos fijarse como se hace esto al especificar el Ejemplo 19.2 del libro.

17. Sea $N \geq 0$, especificar y derivar un programa que calcule el mayor natural x que satisface $x^3 + x \leq N$.

Ejercicios extra

18. Derive un programa para calcular el máximo común divisor entre dos enteros positivos. Utilice la siguiente especificación:

    Const $X, Y : Int$;
    Var $x, y : Int$;
    $\{X > 0 \wedge Y > 0 \wedge x = X \wedge y = Y\}$
    S
    $\{x = mcd.X.Y\}$

    Utilice como invariante $\{I : x > 0 \wedge y > 0 \wedge mcd.x.y = mcd.X.Y\}$.

    Para la derivación serán de utilidad las siguientes propiedades del mcd:

    a) $mcd.x.x = x$
    b) $mcd.x.y = mcd.y.x$
    c) $x > y \;\Rightarrow\; mcd.x.y = mcd.(x - y).y$
    d) $y > x \;\Rightarrow\; mcd.x.y = mcd.x.(y - x)$

19. Considere las siguientes definiciones recursivas de la función de exponenciación $exp.x.y$, especificada como $exp.x.y = x^y$:

    a) Definición de complejidad lineal:
    $$\begin{array}{lll} exp.x.y & = & (\;\; y = 0 \to 1 \\ & & \Box \; y \neq 0 \to x * exp.x.(y-1) \\ & &) \end{array}$$

    b) Definición de complejidad logarítmica:
    $$\begin{array}{lll} exp.x.y & = & (\;\; y = 0 \to 1 \\ & & \Box \; y \neq 0 \to (\;\; y \bmod 2 = 0 \to exp.(x*x).(y \div 2) \\ & & \qquad\qquad \Box \; y \bmod 2 = 1 \to x * exp.x.(y-1) \\ & & \qquad\qquad) \\ & &) \end{array}$$

Derive **dos** programas imperativos que calculen la exponenciación, cada uno utilizando una de las definiciones recursivas. Utilice la siguiente especificación:

Const $X, Y : Int$;
Var $x, y, r : Int$;
$\{x = X \land y = Y \land x \geq 0 \land y \geq 0\}$
S
$\{r = X^Y\}$

Utilice como invariante $\{I : y \geq 0 \land r * x^y = X^Y\}$.

20. Considere el ejercicio 5.

    a) ¿En el programa derivado en el ejercicio 5a se recorre todo el arreglo? Si la respuesta es afirmativa piense si esto es necesario. Si no lo es busque una manera de derivar un programa equivalente que no recorra innecesariamente todo el arreglo.

    b) Repita el mismo razonamiento sobre el ejercicio 5b y derive si es necesario el programa mejorado.

21. Derivar el siguiente programa

    Const $M : Int, A : array\ [0, M)\ of\ Int$;
    Var $r : Int$;
    $\{P : M \geq 0\}$
    S
    $\{Q : r = \langle \mathrm{N}\, i\ : 0 \leq i < M :\ \langle \sum j\ : 0 \leq j < i :\ A.j \rangle \leq i * A.i \rangle\}$

22. Derivar el siguiente programa

    Const $N : Int$;
    Var $a : array\ [0, N)\ of\ Int$;
    $r : Int$;
    $\{P : N > 0\}$
    S
    $\{Q : r = \langle \mathrm{Max}\ i\ : 0 \leq i < N \land \langle \forall j\ : 0 < j < i :\ a.(j-1) < a.j \rangle :\ i \rangle\}$

    Nota: Antes de derivar decir que debería hacer el programa.

    Nota: No se puede usar ∞ en el programa.

23. (Máxima diferencia cuadrada) Derivar un programa para la siguiente especificación:

    Const $N : Int$;
    Var $a : array[0, N)\ of\ Int; r : Int$;
    $\{P : N \geq 2\}$
    S
    $\{Q : r = \langle \mathrm{Max}\ p, q\ : 0 \leq p < q < N :\ (a.p - a.q)^2 \rangle\}$

Ejercicios de laboratorio

> **Laboratorio 10** (Múltiplos) . Hacer un programa en un archivo `multiplos.c` que contenga las siguientes funciones:
>
> ```
> bool todos_pares(int tam, int a[])
> bool existe_multiplo(int m, int tam, int a[])
> ```
>
> La primera recibe un tamaño máximo de arreglo y un arreglo como argumento devolviendo `true` cuando todos los elementos del arreglo `a[]` son numeros pares y `false` en caso contrario. La segunda determina si hay en el arreglo `a[]` algún elemento que es múltiplo de `m`. En la función `main` se debe pedir al usuario los elementos del arreglo (asumiendo un tamaño constante) y luego permitirle elegir qué función ejecutar. En caso que se elija la función `existe_multiplo()` se le debe pedir al usuario un valor para `m`.

Laboratorio 11 (Mínimos). Hacer un programa en un archivo con nombre `minimos.c` que contenga las siguientes funciones:

```
int minimo_pares(int tam, int a[])
int minimo_impares(int tam, int a[])
```

Estas funciones reciben un tamaño máximo de arreglo y un arreglo como argumentos, devolviendo el elemento par más pequeño del arreglo y el elemento impar más pequeño del arreglo respectivamente.

a) En la función `main` se debe pedir al usuario los elementos del arreglo (asumiendo un tamaño constante) y luego mostrar por pantalla:

- El resultado de `minimo_pares()`, para el arreglo ingresado
- El resultado de `minimo_impares()`, de nuevo, para el arreglo ingresado
- El elemento mínimo del arreglo ingresado (utilizando el resultado de ambas funciones para calcularlo).

Pueden definir alguna función auxiliar si les resulta necesario.

NOTA: Investigar las constantes definidas en la librería `<limits.h>` para definir el neutro de la operación *mínimo*

b) (Punto estrella) Hacer una segunda versión del programa en el archivo `minimos_estrella.c` y usar las funciones del ejercicio 10 en la fucnión `main` para que en caso de no haber elementos pares no se muestre el resultado de `minimo_pares()` y en caso de no haber impares no se muestre el resultado de `minimo_impares()`

Laboratorio 12 (Función prim_iguales) Programar en el archivo `prim_iguales.c` la función

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

que siendo `tam` el tamaño del arreglo `a[]` devuelve **la longitud** del tramo inicial más largo cuyos elementos son todos iguales (parecida a la función `primIguales` del Proyecto 1).

a) En la función `main` se le debe pedir al usuario los elementos del arreglo asumiendo un tamaño constante previamente establecido (en tiempo de compilación) y luego mostrar el resultado de la función `prim_iguales` por pantalla

b) (Punto Estrella) Mostrar por pantalla el mayor tramo inicial del arreglo `a[]` que tiene a todos sus elementos iguales.

Laboratorio 13 (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 prototipo:

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

donde la estructura `comp_t` se define como sigue:

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

La función toma un tamaño máximo de arreglo, el arreglo 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.

Laboratorio 14 (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 prototipo:

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

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.

Laboratorio 15 (Arreglo de asociaciones) En `asoc.c` programar la función

```
bool asoc_existe(int tam, struct asoc a[], clave_t c)
```

Donde la estructura `struct asoc` y los tipos `clave_t`, `valor_t` se definen como:

```
typedef char clave_t;
typedef int valor_t;

struct asoc {
    clave_t clave
    valor_t valor
};
```

El llamado a `asoc_existe(tam, a, c)` debe indicar si la clave `c` se encuentra en el arreglo de asociaciones `a[]`. En la función `main` pedir al usuario los datos del arreglo (asumiendo un tamaño constante) y luego pedir una clave. Finalmente usar la función `asoc_existe` para verificar la existencia de la clave ingresada y mostrar por pantalla un mensaje indicando si la clave existe o no en el arreglo de asociaciones.

Laboratorio 16 (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.

Laboratorio 17 (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(unsigned int longitud, persona_t arr[]);
persona_t persona_de_mayor_edad(unsigned int longitud, persona_t arr[]);
persona_t persona_de_menor_altura(unsigned int longitud, persona_t arr[]);
```

Las tres funciones toman como argumento una longitud máxima de arreglo y un arreglo de personas. 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:

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