

## Práctica de ejercicios # 8 - Modelo imperativo

Estructuras de Datos, Universidad Nacional de Quilmes

4 de noviembre de 2020

### **Aclaraciones:**

- *Los ejercicios fueron pensados para ser resueltos en el orden en que son presentados. No se saltee ejercicios sin consultar antes a un docente.*
- *Recuerde que puede aprovechar en todo momento las funciones que ha definido, tanto las de esta misma práctica como las de prácticas anteriores.*
- *Pruebe todas sus implementaciones, al menos en una consola interactiva.*
- *Es sumamente aconsejable resolver los ejercicios utilizando primordialmente los conceptos y metodologías vistos en videos publicados o clases presenciales, dado que los exámenes de la materia evaluarán principalmente este aspecto. Si se encuentra utilizando formas alternativas al resolver los ejercicios consulte a los docentes.*

### Ejercicio 1

Grafique la memoria resultante luego de ejecutar los siguientes programas.

```
1. int x = 0;
   int y = 2;
   x = x+y;

2. int x = 0;
   int y = 0;
   while(y < 5) {
       x += y;
       y++;
   }

3. int y = 0;
   bool b = true;
   while(b) {
       y++;
       b = !b;
   }
```

### Ejercicio 2

Indicar el propósito de los siguientes procedimientos o funciones, dando algunos ejemplos de uso junto con su resultado.

```
1. // Precondición: c1 < c2
   void printFromTo(char c1, char c2) {
       for(int i = 0; c1 + i <= c2; i++) {
           cout << c1 + i << ", ";
       }
   }
```

```

        cout << endl;
    }

2. \\ Precondición: n >= 0
int fc(int n) {
    int x = 1;
    while(n > 0) {
        x = x * n;
        n--;
    }
    return x;
}

3. \\ Precondición: n <= m
int ft(int n, int m) {
    if (n == m) {
        return n;
    }
    return n + ft(n+1, m);
}

```

### Ejercicio 3

Dada la estructura de pares representada como **struct** en C++, definir las siguientes funciones sobre pares. Recuerde probar sus implementaciones en un procedimiento main.

```

struct Par {
    int x;
    int y;
}

// Propósito: construye un par
Par consPar(int x, int y);

// Propósito: devuelve la primera componente
int fst(Par p);

// Propósito: devuelve la segunda componente
int snd(Par p);

// Propósito: devuelve la mayor componente
int maxDelPar(Par p);

// Propósito: devuelve un par con las componentes intercambiadas
Par swap(Par p);

// Propósito: devuelve un par donde la primer componente
// es la división y la segunda el resto entre ambos números
Par divisionYResto(int n, int m);

```

### Ejercicio 4

Dar dos implementaciones para las siguientes funciones, una iterativa y otra recursiva, y utilizando la menor cantidad posible de variables. Recuerde que puede definir subtareas en caso de que sea estrictamente necesario.

1. `void printN(int n, string s)`  
Propósito: imprime `n` veces un string `s`.
2. `void cuentaRegresiva(int n)`  
Propósito: imprime los números desde `n` hasta 0, separados por saltos de línea.
3. `void desdeCeroHastaN(int n)`  
Propósito: imprime los números de 0 hasta `n`, separados por saltos de línea.
4. `int mult(int n, int m)`  
Propósito: realiza la multiplicación entre dos números (sin utilizar la operación `*` de C++).
5. `void primerosN(int n, string s)`  
Propósito: imprime los primeros `n` char del string `s`, separados por un salto de línea.  
Precondición: el string tiene al menos `n` char.
6. `int pertenece(char c, string s)`  
Propósito: indica si un char `c` aparece en el string `s`.
7. `int apariciones(char c, string s)`  
Propósito: devuelve la cantidad de apariciones de un char `c` en el string `s`.

### Ejercicio 5

Dada la estructura de fracciones representada como `struct` en C++, definir las siguientes funciones sobre fracciones. Recuerde probar sus implementaciones en un procedimiento `main`.

```
struct Fraccion {
    float numerador;
    float denominador;
}

// Suponer que el denominador no es cero

// Propósito: construye una fraccion
Fraccion consFraccion(float numerador, float denominador);

// Propósito: devuelve el numerador
float numerador(Fraccion f);

// Propósito: devuelve el denominador
float denominador(Fraccion f);

// Propósito: devuelve el resultado de hacer la división
float division(Fraccion f);

// Propósito: devuelve una fracción que resulta de multiplicar las fracciones
// (sin simplificar)
Fraccion multF(Fraccion f1, Fraccion f2);

// Propósito: devuelve una fracción que resulta
// de simplificar la dada por parámetro
Fraccion simplificada(Fraccion p);

// Propósito: devuelve la primera componente
Fraccion sumF(Fraccion f1, Fraccion f2);
```