Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
Computación II (CI-2126)

LABORATORIO 4 (TDA COMPLEJO)
Posibles soluciones

1.- Dadas la siguientes especificaciones sintácticas del TDA número complejo elabore las semánticas.

Los *números complejos* (C) son el conjunto construido por *pares ordenados* del conjunto de los números reales (R), que tienen la forma **a + b•i** donde *a* y *b* son elementos del conjunto R y el término *i* se conoce como *unidad imaginaria*, teniendo además el valor de $(-1)^{1/2}$.

```
/* El objeto Número Complejo es un par ordenado (a, b) donde "a" es la parte real y "b" la imaginaria de la forma (a + bxi) . Se apoya en el TDA Número Real. Algunas de sus operaciones son:

construir_complejo: void ---> Complejo
destruir_complejo: Complejo ---> void
llenar_complejo: Real x Real ---> Complejo
mostrar_complejo: Complejo ---> Cadena de caracteres
parte_real: Complejo ---> Real
parte_imaginaria: Complejo ---> Real
clonar_complejo: Complejo ---> Complejo
verifica_igualdad: Complejo x Complejo ---> lógico
sumar_complejo: Complejo x Complejo ---> Complejo
multiplicar_complejo: Complejo x Complejo ---> Complejo

*/
```

Posible solución:

```
/* Especificación semántica del TDA Número Complejo:

PRE:
        complejo construir_complejo ( )
POST: construir_complejo devuelve una estructura de datos con parte real e imaginaria que de ser llenada correctamente representará un número complejo.


PRE: se dispone de un objeto TDA Número Complejo c
        void destruir_complejo (complejo c)
```

POST: Se liberan los recursos vinculados con la estructura de datos que representó a un número complejo.

PRE: Se dispone de dos objetos TDA número real.

complejo llenar_complejo (real r1, real r2)

POST: la función devuelve una estructura de datos TDA complejo con valores asignados, su parte real y su parte imaginaria.[&]

PRE: Se dispone de un objeto TDA número complejo.

cadena de caracteres mostrar_complejo (complejo c)

POST: Devuelve una cadena de caracteres de la forma a + bxi

PRE: Se provee un número complejo, es decir una estructura TDA complejo que previamente se llenó

real parte_real (complejo c)

POST: retorna la parte real del número complejo.

PRE: Se provee un número complejo C, es decir una estructura TDA complejo que previamente se llenó

real parte_imaginaria (complejo c)

POST: retorna la parte imaginaria del número complejo.

PRE: Se proveen un número complejo C1.

complejo clonar_complejo (complejo c)

POST: la función hace un clon de la parte real e imaginaria del número complejo C.[*]

PRE: Se proveen un número complejo C1 y un número complejo C2.

lógico verifica_igualdad (complejo c1, complejo c2)

POST: la función devuelve "verdad" si C1 es igual a C2. De lo contrario devuelve "falso".

PRE: Se proveen un número complejo C1 y un número complejo C2.

complejo sumar_complejo (complejo c1, complejo c2)

POST: la función devuelve un nuevo número complejo que contiene la suma de C1 y C2.[#]

PRE: Se proveen un número complejo C1 y un número complejo C2.

complejo multiplicar_complejo (complejo c1, complejo c2)

& Esta función internamente invoca a la función "construir_complejo"

* Esta función internamente invoca a la función "construir_complejo"

Esta función internamente invoca a la función "construir_complejo"

POST: la función devuelve un nuevo número complejo que contiene el producto de C1 y C2.[%]
*/

2.- Proponga dos (2) estructuras de datos, de memoria estática, posibles para implementar un objeto del TDA número complejo en lenguaje C.

Posible solución:

/* Alternativa 1: un vector de dos posiciones. La primera posición [0] contiene la parte real y la segunda [1] la parte imaginaria */

```
float Vector_complejo [2];
```

/* Alternativa 2: una estructura de dos campos */

```
struct complejo
{
        double a;       /* Parte Real */
        double b;       /* Parte Imaginaria */
};
```

3.- Desarrolle la implementación del TDA Complejo.

Posible solución: El archivo “complejo.h” puede contener:

```
# include <string.h>

        /* --- Estructura Complejo ---*/
        typedef struct TDA_complejo {
                double a; /*Parte real */
                double b; /* Parte imaginaria */
        } complejo;


        /*--- Prototipos ---*/
        complejo construir_complejo (void);
        void destruir_complejo (complejo c);
        complejo llenar_complejo (double r1, double r2);
        void mostrar_complejo (complejo c);
        double parte_real (complejo c);
        double parte_imaginaria (complejo c);
        complejo clonar_complejo (complejo c);
        int verifica_igualdad (complejo c1, complejo c2);
```

% Esta función internamente invoca a la función “construir_complejo”

```
        complejo sumar_complejo (complejo c1, complejo c2);
        complejo multiplicar_complejo (complejo c1, complejo c2);


        /* --- Rutinas --- */
        complejo construir_complejo (void)
        {
                complejo tempo;

                return (tempo);
        }

        void destruir_complejo (complejo c)
        {
                /* Realmente no se elimina el complejo. Es una simulación. Si se hubiera hecho con
memoria
                   dinámica tendría más sentido */
                c.a=0;
                c.b=0;
        }

        complejo llenar_complejo (double r1, double r2)
        {
                /* Realmente la función "construir_complejo" no se requiere ya que el sistema
                   provee los recursos que requiere el complejo. Pero se hace para mantener
                   la coherencia con el diseño */
                complejo tempo;

                tempo=construir_complejo();
                tempo.a=r1;
                tempo.b=r2;
                return (tempo);
        }

        void mostrar_complejo (complejo c)
        {
                fprintf (stdout, "\nEl numero complejo es: ");
                fprintf (stdout, "%f", parte_real(c));
                fprintf (stdout, " + ");
                fprintf (stdout, "%f", parte_imaginaria(c));
                fprintf (stdout, "xi");
        }

        double parte_real (complejo c)
        {
                return (c.a);
        }

        double parte_imaginaria (complejo c)
        {
                return (c.b);
        }

        complejo clonar_complejo (complejo c)
        {
                /* Realmente la función "construir_complejo" no se requiere ya que el sistema
                   provee los recursos que requiere el complejo. Pero se hace para mantener
                   la coherencia con el diseño */
                complejo tempo;
```

```
        tempo=construir_complejo();
        tempo.a=c.a;
        tempo.b=c.b;
        return (tempo);
}

int verifica_igualdad (complejo c1, complejo c2)
{
        /* Devuelve "verdad" (1) si los dos complejos son iguales */
        if (c1.a != c2.a)
                return (0);
        else {
                if (c1.b != c2.b)
                        return (0);
                else
                        return (1);
        }
}

complejo sumar_complejo (complejo c1, complejo c2)
{
        /* Realmente la función "construir_complejo" no se requiere ya que el sistema
           provee los recursos que requiere el complejo. Pero se hace para mantener
           la coherencia con el diseño */
        complejo tempo;

        tempo=construir_complejo();
        tempo.a=c1.a+c2.a;
        tempo.b=c1.b+c2.b;
        return (tempo);
}

complejo multiplicar_complejo (complejo c1, complejo c2)
{
        /* Realmente la función "construir_complejo" no se requiere ya que el sistema
           provee los recursos que requiere el complejo. Pero se hace para mantener
           la coherencia con el diseño */
        complejo tempo;

        tempo=construir_complejo();
        tempo.a=(c1.a*c2.a)-(c1.b*c2.b);
        tempo.b=(c1.a*c2.b)+(c1.b*c2.a);
        return (tempo);
}
```

4.- Usando la implementación que realizó para la pregunta previa, resuelva el siguiente problema: “Dados dos (2) números complejos, verifique si su suma y su multiplicación son iguales.”

Posible solución:

```
#include <stdio.h>
#include <stdlib.h>
#include "complejo.h"

int main ()
{
        complejo num1, num2;
```

```
        double a, b;

        /* Se construye el primer número complejo */
        fprintf (stdout, "\nIntroduzca la parte real del número complejo: ");
        scanf ("%lf", &a);
        fprintf (stdout, "\nIntroduzca la parte imaginaria del número complejo: ");
        scanf ("%lf", &b);
        num1=llenar_complejo (a, b);
        mostrar_complejo(num1);

        /* Se construye el segundo número complejo */
        fprintf (stdout, "\nIntroduzca la parte real del número complejo: ");
        scanf ("%lf", &a);
        fprintf (stdout, "\nIntroduzca la parte imaginaria del número complejo: ");
        scanf ("%lf", &b);
        num2=llenar_complejo (a, b);
        mostrar_complejo(num2);

        if (verifica_igualdad (sumar_complejo (num1, num2), multiplicar_complejo (num1, num2)))
                fprintf (stdout, "\nLa suma y la multiplicacion de ambos numeros complejos son iguales");
        else
                fprintf (stdout, "\nLa suma y la multiplicacion de ambos numeros complejos son diferente");


        fprintf(stdout, "\nSuma es: ");
        mostrar_complejo (sumar_complejo(num1, num2));

        fprintf(stdout, "\nProducto es: ");
        mostrar_complejo (multiplicar_complejo(num1, num2));

        exit (0);
}
```

19 de Septiembre de 2014

MTS/mts