UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

E.T.S.E

PROGRAMACIÓN II

Examen

Profesor: David E. Losada **Mayo 2010**

Fecha: 31/5/2010.
Nombre y apellidos:_____

1. Test (3.2 puntos. Cada respuesta correcta suma 0.2 puntos. Cada respuesta incorrecta resta 0.1 puntos)

1. Un Tipo Abstracto de Dato (TAD) permite
   a. definir la sintaxis, semántica e implementación de un nuevo de tipo de datos
   b. hacer explícitos los detalles de programación asociados a un nuevo de tipo de datos
   c. abstraerse de las operaciones que se harán sobre el tipo (esto es, no se concretan)
   d. ninguna de las anteriores

2. Un Tipo Abstracto de Dato (TAD) se gestiona de manera opaca para
   a. ocultar los detalles de implementación de las operaciones
   b. ocultar la interfaz del tipo
   c. ocultar la interfaz y detalles de implementación del tipo
   d. ninguna de las anteriores

3. Un TAD pila se implementa de dos maneras alternativas (A y B). La alternativa A es una lista enlazada de elementos con un puntero a la cima de la pila. La alternativa B es mediante un bloque de memoria contigua.

   ¿cual de las siguientes afirmaciones es cierta?

   a. un problema con la alternativa A es que tengo que decidir cuantos elementos reservo a priori en la lista enlazada
   b. un problema con la alternativa B es que necesito un puntero adicional para cada valor almacenado en la pila
   c. un problema con la alternativa B es para decidir cuantos elementos reservo memoria en el bloque
   d. ninguna de las anteriores

4. Un TAD lista se implementa de dos maneras alternativas (A y B). La alternativa A es una lista enlazada de elementos con un puntero al primer elemento de la lista. La alternativa B es mediante una lista doblemente enlazada con un puntero al primer elemento de la lista y doble enlace entre cada elemento de la lista (hacia delante y hacia atrás).

   ¿cual de las siguientes afirmaciones es cierta?
   a. la alternativa A me facilita la gestión de los accesos, tanto para inserciones como para eliminaciones de elementos de la lista
   b. la alternativa A es más eficaz pues los elementos están dispuestos de manera contigua en memoria
   c. un problema con la alternativa B es que utiliza más memoria que la A.
   d. ninguna de las anteriores

5. La complejidad espacial es una estimación de
    a. el tiempo que tardará en ejecutarse un programa
    b. la memoria que utilizará el programa cuando se ejecuta
    c. el tiempo y memoria que utilizará un programa al ejecutarse
    d. ninguna de las anteriores

6. El análisis de algoritmos se suele realizar de manera asintótica, esto quiere decir
    a. que sólo importan los lazos del programa. Lo demás se descarta.
    b. que nos fijamos en las funciones del programa que hagan cálculos matemáticos relacionados con límites.
    c. que nos fijamos en el comportamiento del algoritmo para tallas grandes del problema
    d. ninguna de las anteriores

7. Se dispone de un algoritmo logarítmico y un algoritmo lineal que ambos resuelven el mismo problema...
    a. el logarítmico es siempre mejor
    b. para tallas pequeñas el lineal pudiera ser mejor
    c. para tallas grandes el lineal será mejor
    d. ninguna de las anteriores

8. Un algoritmo toma 2N+100 pasos para resolver un problema y otro toma $\sqrt{N} + 27$
    a. el segundo algoritmo es sublineal
    b. en general, para resolver distintos tipos de problemas con distintas tallas, escogeremos el primer algoritmo
    c. el segundo algoritmo es superlineal
    d. ninguna de las anteriores

9. El orden de complejidad nos da una idea de
    a. el comportamiento del algoritmo en el mejor de los casos
    b. el comportamiento del algoritmo en el caso promedio
    c. el comportamiento del algoritmo en el caso de tallas pequeñas
    d. ninguna de las anteriores

10. El algoritmo de búsqueda lineal es un ejemplo clásico de
    a. estrategia divide y vencerás
    b. estrategia fuerza bruta
    c. encadenamiento hacia atrás
    d. ninguna de las anteriores

11. El algoritmo de búsqueda binaria es un algoritmo
    a. cuadrático
    b. lineal
    c. superlineal
    d. ninguna de las anteriores

12. Una estrategia divide y vencerás
    a. es siempre mejor que una estrategia iterativa (por ejemplo, búsqueda binaria es mejor que búsqueda lineal)
    b. puede dar lugar a mayor coste que una estrategia iterativa (depende de los costes de descomposición, combinación, etc.)
    c. se puede aplicar siempre (sea como sea el problema)
    d. ninguna de las anteriores

13. Un algoritmo voraz
    a. nunca reconsidera las decisiones que toma
    b. puede reconsiderar las decisiones que toma en caso de que llegue a un camino sin salida
    c. explora siempre todas las posibilidades antes de tomar cualquier decisión
    d. ninguna de las anteriores

14. Un algoritmo voraz
    a. está garantizado que si hay solución óptima llega a ella
    b. nunca encuentra la solución óptima
    c. suele ser bastante eficiente
    d. ninguna de las anteriores

15. La programación dinámica
    a. consigue mejor complejidad temporal a costa de sacrificar complejidad espacial
    b. consigue mejor complejidad espacial a costa de sacrificar complejidad temporal
    c. escoge dinámicamente entre optimizar la complejidad espacial o la temporal
    d. ninguna de las anteriores

16. La verificación de programas
    a. permite asegurar que no hay errores en el software
    b. debe asegurar cubrir los menos casos posibles con el mayor número de ejemplos de prueba
    c. puede permitir descubrir algunos errores del software, no necesariamente todos
    d. ninguna de las anteriores

2. (1.5 puntos) Construye la especificación formal del TAD Complejo con las operaciones Crea, Conjugado, Suma, Producto, ParteReal y ParteImaginaria.

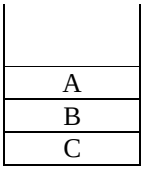
El cuerpo de los números complejos responde a las siguientes definiciones:

a) Se define número complejo a todo par ordenado (a,b) donde a y b son números reales (se suele representar como a+bi)
b) Dado el número complejo (a,b), se dice que a es su parte real y b su parte imaginaria
c) Llamaremos conjugado del número complejo (a,b) al número complejo (a,-b)
d) Se define la suma de complejos como (a1,b1)+(a2,b2)=(a1+a2,b1+b2)
e) Se define el producto de complejos como (a1,b1)*(a2,b2)=(a1*a2-b1*b2 , a1*b2+b1*a2)

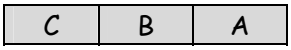
3. (2 puntos) Dado un TAD Pila, que cuenta con las conocidas operaciones de PilaVacia, EsVacia, Cima, Push y Pop, y dada una Pila que contiene números enteros escribe una función VaciaHasta3 que lo que hace es eliminar elementos de la pila hasta que la cima tenga el valor 3 (este valor de 3 se dejaría en la pila) o la pila quede totalmente vacía.

```
// rellena los argumentos y su tipo que recibe la función
void VaciaHasta3 (                )
{
// rellena las instrucciones necesarias

.....

}
```

4. (1.8 puntos) Se dispone de una pila y se quiere extraer sus elementos y meterlos en una cola de modo que el elemento en la base de la pila sea el primero de la cola, y el elemento en la cima de la pila sea el último de la cola (y los elementos intermedios están en sus posiciones respectivas según indica la figura). La cola hay que crearla dentro de la función (no viene creada de fuera). Al finalizar la función la cola debe contener todos los elementos originalmente en la pila y la pila debe quedar vacía.
Realizar una función que utilizando las operaciones conocidas en los TADs Pila (PilaVacia, EsVacia, Cima, Push y Pop) y Cola (ColaVacia, EsColaVacia, primeroCola, añadirCola, eliminarCola) realice esta operación. No se puede utilizar ningún otro TAD que no sea Pila/Cola. Sugerencia: Utilizar una pila auxiliar. El tipo de los elementos denominadlo TElemento.



Pila original



Cola que se quiere construir

```
// rellena los argumentos y tipo que recibe la función
void crea_Cola_dada_una_Pila(                    )
{
// rellena las instrucciones necesarias

.....

}
```

5. (1.5 puntos) Dado los siguientes programas, determina el orden de complejidad de los mismos siendo la talla del problema el número N

Programa A:

```
....

int main()
{
....

k = f1(N)+f2(N);


}
```

```
float f1(int num)
{
int aux, aux2;
float rdo=0;

for (aux=0;aux<num;aux++)
  for (aux2=0;aux2<num;aux2++)
      rdo += aux*aux2;

return rdo;
}
```

```
float f2(int num)
{
int aux;
float rdo=0;

for (aux=0;aux<num;aux++)
  rdo += aux;

return rdo;
}
```

Programa B:

```
....

int main()
{
....

k = f1(N);

k = k + f2(N);
}
```

```
float f1(int num)
{
float rdo;

if (num < 5)
   return (num*1.1)

rdo= 2*f1(num-1)-f1(num-2);

return(rdo);
}
```

```
float f2(int num)
{
int aux;
float rdo=0;

for (aux=0;aux<num;aux++)
  rdo += aux;

return rdo;
}
```