UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

E.T.S.E

PROGRAMACIÓN II

Examen

Profesor: David E. Losada

Mayo/Junio 2012

Fecha: 5/6/2012.

Nombre y apellidos:

PARTE I (SIN MATERIAL)

1. (1 pto) Explica brevemente qué significa que la implementación de un TAD sea **opaca** y qué ventajas supone eso. Haz la explicación genérica (independiente de cualquier lenguaje de programación).

2. (1 pto) Se dispone de dos implementaciones distintas de un **TAD cola**. Una de ellas (implementación A) mediante un bloque contiguo de elementos en memoria, en la que la creación de la cola implica la reserva de un tamaño máximo de elementos (tamaño_max). En esta implementación A nunca podrá haber más de tamaño_max elementos encolados. Otra implementación (implementación B) es a través de una lista enlazada de nodos en la que cada vez que encolamos un elemento se crea dinámicamente el nodo correspondiente en memoria (esto es, la creación de la lista no pide memoria para elemento alguno, se pide memoria al introducir individualmente los elementos). Responde a las siguientes cuestiones de manera breve:
   - En igualdad del resto de circunstancias, si sabemos que el tamaño en bytes del TIPOELEMENTO a guardar en la cola es muy grande, ¿que implementación deberíamos preferir y por qué?
   - En igualdad del resto de circunstancias, si desconocemos el número típico de elementos que será necesario encolar y además sabemos que en tiempo de ejecución este número va a ser muy variable y puede llegar a ser muy grande, ¿que implementación deberíamos preferir y por qué?
   - En igualdad del resto de circunstancias, si sabemos que la operación más típica que se va a invocar sobre la cola es la de destrucción, pues la aplicación para que usamos el TAD va a estar constantemente creando y destruyendo colas, ¿que implementación deberíamos preferir y por qué?
   - Si la implementación A opta por mover todos los elementos del bloque continuo una posición a la izquierda cada vez que eliminamos el primero del bloque (esto es, cada vez que extraemos el primero de la cola), que operación es más eficiente: sacar el primero de la cola en la implementación A o sacar el primero de la cola en la implementación B? por qué?

3. Test (3 puntos. Cada respuesta correcta suma 0.2 puntos. Cada respuesta incorrecta resta 0.1 puntos)

    1. Cual de las siguientes afirmaciones es cierta respecto a un Tipo de Datos primitivo convencional (p.e. float o int) frente a un Tipo Abstracto de Dato (TAD)
        a. el TAD no tiene operaciones asociadas
        b. el TAD no oculta la implementación del tipo y el TD primitivo si.
        c. no existe ninguna diferencia entre ambos, son dos formas equivalentes de referirnos a lo mismo
        d. ninguna de las anteriores

    2. La parte semántica de la especificación formal de un Tipo Abstracto de Dato (TAD)
        a. describe el comportamiento esperado de las operaciones del TAD
        b. explicita los detalles de implementación de las operaciones del nuevo de tipo de datos
        c. explicita exclusivamente las operaciones (argumentos que tienen y valores que devuelven) del TAD
        d. describe de manera informal que tipo de argumentos se esperan en las operaciones del TAD

3. Dadas dos implementaciones distintas de un TAD (p.e. un TAD lista implementado como un bloque contiguo en memoria o como una lista enlazada de nodos), cual de las siguientes afirmaciones es cierta:
    a. puede ser que una implementación cumpla la especificación semántica y otra implementación no la cumple. En todo caso, ambas implementaciones serían correctas.
    b. puede ser que una implementación cumpla la especificación sintáctica y otra implementación no la cumple. En todo caso, ambas implementaciones serían correctas.
    c. puede ser que una implementación sea más eficiente que otra, aún realizando la misma labor
    d. ninguna de las anteriores

4. En referencia a las operaciones permitidas por un TAD pila, un TAD lista y un TAD cola...
    a. el TAD cola es el más flexible de los tres pues soporta una variedad más amplia de operaciones posibles
    b. el TAD lista es el más flexible de los tres pues soporta una variedad más amplia de operaciones posibles
    c. el TAD pila es el más flexible de los tres pues soporta una variedad más amplia de operaciones posibles
    d. ninguna de las anteriores

5. Una aplicación necesita almacenar objetos en una estructura que permita inserciones al principio de la estructura, inserciones al final de la estructura y, en ocasiones, inserciones en el medio de la estructura. ¿Qué TAD puedo usar?
    a. un TAD cola
    b. un TAD pila
    c. un TAD cola con prioridad
    d. un TAD lista

6. La búsqueda sin centinela
    a. tiene el mismo orden de complejidad que la búsqueda con centinela pero realiza más pasos
    b. realiza una búsqueda binaria y, por tanto, mejora el orden de complejidad con respecto a la búsqueda con centinela
    c. consume más memoria que la búsqueda con centinela
    d. tiene un orden de complejidad mayor que la búsqueda con centinela

7. La búsqueda binaria
    a. tiene orden de complejidad lineal
    b. realiza dos pasadas para buscar los datos
    c. tiene orden de complejidad sublineal
    d. ninguna de las anteriores

8. El algoritmo quicksort de ordenación
    a. es menos eficiente temporalmente que el algoritmo de la burbuja
    b. es un ejemplo de algoritmo lineal
    c. es menos eficiente temporalmente que el algoritmo de selección
    d. ninguna de las anteriores

9. ¿qué significa que se realice el análisis de algoritmos de manera asintótica?
    a. que se realiza un estudio de los algoritmos centrándonos en su comportamiento en el mejor de los casos
    b. que se realiza un estudio de los algoritmos centrándonos en su comportamiento en el peor de los casos
    c. que se realiza un estudio de los algoritmos centrándonos en su comportamiento en el caso promedio
    d. que se realiza un estudio de los algoritmos centrándonos en su comportamiento para tallas grandes

10. El orden de complejidad omega (Ω) nos da una idea de
    a. como estan acotados superiormente los casos del problema (peores casos)
    b. como estan acotados inferiormente los casos del problema (mejores casos)
    c. como estan acotados superior e inferiormente los casos del problema
    d. ninguna de las anteriores

11. Para solucionar el mismo problema disponemos de un algoritmo de orden n^3, otro de orden $n!$ y otro de orden 2^n . ¿Cuál elegimos?
    a. el cúbico (n^3)
    b. el cuadrático (2^n)
    c. el de orden $n!$
    d. ninguno es viable como solución computacional

12. Los algoritmos de fuerza bruta
    a. suelen implicar bajo coste computacional
    b. suelen dividir el espacio de búsqueda en subproblemas
    c. realizan una poda del espacio de búsqueda de soluciones
    d. ninguna de las anteriores

13. Un algoritmo voraz
    a. puede reconsiderar decisiones tomadas previamente mediante vuelta atrás
    b. suele ser más eficiente temporalmente que uno de fuerza bruta
    c. es un caso especial de ramificación y poda
    d. ninguna de las anteriores

14. Un algoritmo divide y vencerás
    a. siempre es más eficaz computacionalmente que una resolución directa del problema
    b. aplica una heurística para resolver siempre sólo uno de los subproblemas involucrados
    c. es un caso especial de algoritmos voraces
    d. ninguna de las anteriores

15. La programación dinámica
    a. consiste en sacrificar tiempo computacional para conseguir ahorrar uso de memoria
    b. consiste en sacrificar memoria para conseguir mejor tiempo computacional
    c. consiste en decidir dinámicamente (en tiempo de ejecución) entre ahorrar memoria o ahorrar tiempo
    d. ninguna de las anteriores

UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

E.T.S.E

PROGRAMACIÓN II

Examen

Profesor: David E. Losada

Mayo/Junio 2012

Fecha: 5/6/2012

Nombre y apellidos:_____

PARTE II (CON MATERIAL)

4. (1.4 puntos) Dado el siguiente programa...

```
int main()
{
float saldo=124;
float *p;
float *q;

p=&saldo;
procesa_saldo(p);
q=(float *)malloc (sizeof(float));
*q = *p;

if (saldo > 150)
   saldo = saldo * 1.2;
}
```

```
void procesa_saldo(float *p)
{

if (*p > 100)
   *p = *p + 50;
else
   *p = *p + 5;
}
```

Escribe el valor de las siguientes variables **justo antes de finalizar el programa**:

a) saldo
b) *p
c) *q

Escribe el **tipo de datos correspondiente a las siguientes expresiones**:

a) p
b) saldo
c) &p
d) *q

5. (2 puntos) Dado los siguientes programas, determina el **orden superior (O) de complejidad** de los mismos siendo la talla del problema el número N. Se supone que la operación de generación de un número aleatorio (rand) es una operación que se puede realizar de manera unitaria (1 único paso).

Programa A:

```
void main()
{
int x;
int v[N];
int encontrado;

for (i=0; i< N ; i++)
{
        x = rand();
        v[i]=x;
}

encontrado=0;
for (i=1; i<= N ; i++)
{
        x = rand();
        encontrado=encontrado+busca(v,x);
}

}
```

```
short busca(int v[N],int n)
{
int aux;
for (aux=0;aux<N;aux++)
    if v[aux]==n return 1;

return 0;
}
```

Programa B:

```
void main()
{
int x, encontrado;
int v[N];
for (i=0; i< N ; i++)
{
        x = rand();
        v[i]=x;
}
encontrado=0;
for (i=1; i<= N ; i++)
{
        x = rand();
        encontrado=encontrado+buscabin(v,x);
}
}
```

```
short buscabin(int v[N],int n)
{
int ini=0,fin=N-1,mitad;

mitad=(ini+fin)/2;
while ((ini<=fin)&&(v[mitad]!=n))
 {
 if (dato < v[mitad])
            fin=mitad-1;
 else
            ini=mitad+1;
 mitad=(ini+fin)/2;
 }
if (dato==v[mitad]) return 1;
else return 0;
}
```

Programa C:

```
....

int main()
{
....

k = f1(N);


}
```

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

for (aux=0;aux<100;aux++)
  for (aux2=0;aux2<num;aux2++)
      rdo += aux*aux2;

return (rdo);
}
```

Programa D:

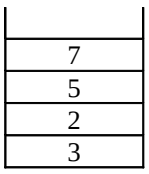
```
void main()
{
....

k = f1(N);

}
```

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

if (num == 1)
   return (1.1);

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

return(rdo);
}
```

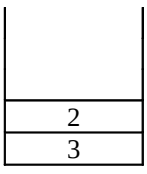
6. (1.6 puntos) Dado un TAD Pila, que cuenta con las conocidas operaciones de PilaVacia, EsVacia, Cima, Push y Pop, y dada una Pila que contiene números enteros escribe una función VaciaImparesEnElTope que lo que hace es eliminar elementos impares (valor impar) del tope de la pila hasta que se quede vacía o encuentre un par (quedaría ese par como tope de la pila). La funcionalidad del TAD Pila se supone que ya está disponible en la librería correspondiente (no hay que implementar las funciones, solo usarlas)

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

.......

}
```

Ejemplo:



Pila original



Pila tras ejecutar la función