# 270005 - PROGRAMACIÓ 2 (Curs Total)

| | |
|---|---|
| **Començat el** | dilluns, 27 abril 2020, 12:30 |
| **Estat** | Acabat |
| **Completat el** | dilluns, 27 abril 2020, 13:51 |
| **Temps emprat** | 1 hora 20 minuts |
| **Qualificació** | **6,40** sobre 10,00 (**64**%) |

Pregunta **1**
Completa
Puntuació 0,40 sobre 2,00
Marca la pregunta

Temps estimat: 15 minuts

Escriu una funció subst_sum que, donat un vector v d'enters v = (v0, ..., v_(n-1)) no buit, substitueix el primer element del vector per la suma de tots els seus elements. La teva solució ha de utilitzar una funció d'immersió i_subst_sum recursiva. Implementa la funció d'immersió, i implementa la función subst_sum utilitzant la funció i_subst_sum, respectant les especificacions donades:

```
// Pre: v = (V_0, ..., V_(N-1)) i N > 0
void subst_sum(vector& v);
// Post: v = (S, V_1, ..., V_(N-1)), ón S = V_0 + ... + V_(N-1)


// Pre: v= (V_0, ..., V_(N-1)), N > 0, 0 <= i < N, s = V_0 + ... + V_(i-1)
void i_subst_sum(vector& v, int i, int& s);
// Post: v = (s,V_1, ..., V_(N-1)), s = V_0 + ... + V_i + ... + V_(N-1)
```

```
void subst_sum(vector& v)
{
    int s;
    int i = 0;
    i_subst_sum(v, i, s);
    v[0]=s;
}
```

```
void i_subst_sum(vector& v, int i, int& s)
{
    if (i > v.size()) return;
    else
    {
    s = s + v[i];
    ++i;
    i_subst_sum(v,i,s);
    }
}
```

```
// Pre: v = (V_0, ..., V_(N-1)) i N > 0

void subst_sum(vector& v) {


   int sum = 0;

   i_subst(v, 0, sum);

}
// Post: v = (S, V_1, ..., V_(N-1)), ón S = V_0 + ... + V_(N-1)


// Pre: v= (V_0, ..., V_(N-1)), N > 0, 0 <= i <= N, s = V_0 + ... + V_(i-1)
void i_subst_sum(vector& v, int i, int& s) {

    if (i == v.size()-1) v[0] = s+v[i];

    else {

       s += v[i];

       i_subst_sum(v, i+1, s);

    }

}
// Post: v = (s,V_1, ..., V_(N-1)), s = V_0 + ... + V_i + ... + V_(N-1)
```

Comentari:
i_subst_sum está mal. No contiene ninguna llamada recursiva, por lo que difícilmente calculará algo que se parezca a lo que se pide. Además, en el if se pregunta i > v.size(), si no fuera cierto y se cumpliera que i == v.size(), la instrucción s = s + v[i]; sería errónea. Y También es errónea porque s no se inicializa nunca.

Pregunta **2**
Completa
Puntuació -0,50 sobre 0,50
Marca la pregunta

Temps estimat: 3 minuts

Donada la següent funció

```
void simdif(const vector<int>& A, const vector<int>& B,
                    vector<int>& C) {
  int i = 0; int j = 0;
  while (i < A.size() and j < B.size()) {
     if (A[i] < B[j]) { C.push_back(A[i]); ++i; }
     else if (A[i] > B[j]) { C.push_back(B[j]); ++j; }
     else { ++i; ++j; }
  }
   ...
}
```

Quina de les següents funcions és una **funció de fita** valida per a demostrar la terminació del bucle principal?

Trieu-ne una:

- a. A.size()-i
- b. C.size()
- c. i+j
- d. B.size()-j
- e. min(A.size()-i,B.size()-j)
- f. A.size()-i > 0 i B.size()-j > 0 (seleccionada)
- g. max(A.size()-i,B.size()-j)
- h. A.size()+B.size()-i-j

Respuesta incorrecta.

La resposta correcta és: A.size()+B.size()-i-j

Pregunta **3**

Completa

Puntuació 1,00 sobre 1,00

Marca la pregunta

Tiempo estimado: 8 minutos

Tenemos un vector de enteros, y queremos saber la longitud de la subsecuencia más larga de números consecutivos iguales y dónde se inicia dicha subsecuencia. En caso de empate nos interesa la que empiece antes.

Considera el siguiente algoritmo:

```
// Pre: v.size() > 0
int ini=0;
int lmax=1;
int i=1;
int ini_act=0;
while (i < v.size()) {
  if (v[i] != v[i-1]) {
     if (lmax < i-ini_act-1) {
       ini=ini_act; lmax=i-ini_act-1;
     }
  }
  ++i;
}
// Post: la subsecuencia de números iguales más larga en v tiene longitud
// "lmax" y se inicia en la posición "ini" del vector; en caso de empate
// "ini" es la menor posición en la que se inicia una subsecuencia
// de repeticiones de longitud máxima
```

¿Cúal de las siguientes afirmaciones es cierta?

Trieu-ne una:

- a. El algoritmo no es correcto. Hay casos en los que se cumple la precondición, pero no se cumple la postcondición al terminar el bucle.
- b. El algoritmo es correcto, tal como puede verificarse usando el invariante
  "la subsecuencia de números iguales más larga en v[0..i] tiene longitud
  "lmax" y se inicia en la posición "ini" del subvector; en caso de empate
  "ini" es la menor posición en la que se inicia una subsecuencia
  de repeticiones de longitud máxima del subvector, 0 <= i <= v.size(), 0 <= ini <= ini_act < v.size()
- c. El algoritmo no es correcto, hay casos en los que el bucle no termina nunca; pero si el algoritmo termina entonces da la respuesta correcta.
- d. El algoritmo no es correcto, debería funcionar con vectores vacíos tambien. Solamente será necesario poner lmax = 0; y ini = -1 antes del bucle para arreglarlo.
- e. El algoritmo no es correcto, debería empezar inicializando i = 0; y comparando v[i] con v[i+1] en cada iteración, no v[i] con v[i-1].

La resposta correcta és: El algoritmo no es correcto. Hay casos en los que se cumple la precondición, pero no se cumple la postcondición al terminar el bucle.

Pregunta **4**

Completa

Puntuació 2,00 sobre 2,00

Marca la pregunta

Tiempo estimado: 10 minutos

Diremos que dos BinTree son isomorfos si hay alguna manera de intercambiar arboles izquierdos y derechos de uno de ellos tantas veces como sea necesario de tal manera que los dos árboles sean iguales.

Por ejemplo

son isomorfos, pero

no lo son.

Completa la función dada más abajo para que cumpla su especificación, es decir, indica qué instrucciones deben reemplazar los lugares indicados con números ([---n---]).

```
bool isomorfos(const BinTree<int> & a, const BinTree<int> & b) {
   bool res;
   if (a.empty() or b.empty()) { [--------- 1 ------------] }

   else if (a.value() != b.value()) { [ ------- 2 ----------] }

   else { [ ------- 3 ------- ] };

   return res;
}
```

Cada uno de los lugares señalados (1, 2 y 3) consiste en una o más instrucciones consecutivas y simples del estilo
`res = ...;`

```
1: if(a.empty() and b.empty()) res = true;
   else res = false;

2: res = false;

3: bool res11, res12, res21, res22;
   res11 = isomorfos(a.left(), b.left());
   res12 = isomorfos(a.left(), b.right());
   res21 = isomorfos(a.right(), b.left());
   res22 = isomorfos(a.right(), b.right());
   res = (res11 and res22) or (res12 and res21);
```

1: res = a.empty() and b.empty():
2: res = false;
3: res = isomorfos(a.left(), b.left()) and isomorfos(a.right(), b.right());
   res = res or isomorfos(a.left(), b.right()) and isomorfos(a.right(), b.left());

Comentari:

Pregunta **5**

Completa

Puntuació 1,50 sobre 2,00

Marca la pregunta

Tiempo estimado: 15 minutos

Tenemos que diseñar un procedimiento parte_lista que, dados una lista l de enteros y un valor entero x, modifica la lista l para que todos los elementos menores o iguales que x estén delante de todos los elementos mayores que x y nos devuelve un iterador al primer elemento >x, o a l.end() si no hay ningún elemento > x en l. El orden relativo entre los elementos de l no importa y no tiene porque coincidir con el que tuvieran en L.

```
// Pre: l = L
list<int>::iterator parte_lista(list<int>& l, int x);
// Post: l es una permutación de L, it = parte_lista(l,x)==l.end()
// si no hay elementos mayores que  x en L o it apunta a un elemento > x,
// todos los elementos en l[it:) son > x, y todos los elementos
// de l[:it) son <= x
```

Tu solución ha de ser iterativa y debe preservar necesariamente este invariante:

- l es una permutación de L **y**
- todos los elementos de l[:it1) son menores o iguales que x **y**
- todos los elementos de l[it2:) son mayores que x,

siendo L el valor original de la lista l e it1 e it2 iteradores a elementos de l.

```
list::iterator parte_lista(list& l, int x)
{
    list::iterator it1 = l.begin();
    list::iterator it2 = l.end();

    while(it1 != it2)
    {
        if(*it1 <= x) it1++;
        else
        {
            l.insert(it2, *it1);
            l.erase(*it1);
            it1++;
        }
```

```
    }
}
```

```
// Pre: l = L
list<int>::iterator parte_lista(list<int>& l, int x) {

    list<int>::iterator it1 = l.begin();

    list<int>::iterator it2 = l.end();

    while (it1 != it2) {

       if (*it1 <= x) ++it1;

       else {

           it2 = l.insert(it2, *it1);

           it1 =  l.erase(it1);

       }

    }

    return it2;

}

// Post: l es una permutación de L, it = parte_lista(l,x)==l.end()
// si no hay elementos mayores que  x en L o it apunta a un elemento > x,
// todos los elementos en l[it:) son > x, y todos los elementos
// de l[:it) son <= x
```

Comentari:
Mal el else.

Pregunta **7**
Completa
Puntuació 1,00 sobre 1,50
Marca la pregunta

Tiempo estimado: 15 min

Completa la función dada más abajo para que cumpla su especificación, es decir, indica qué instrucciones o expresiones deben reemplazar los lugares indicados con números ([---n---]).

```
// Pre: it=IT apunta a un elemento x_i de l o it == l.end()
// y l = L = [x_1,...,x_N] donde N >= 0
void reverse_prefix(list& l, list::iterator it) {
      list::iterator p = [------- 1 --------];
      while ([------ 2 ------]) {
            [---- 3 -----];
      };
};
// Post: l está formada por el prefijo invertido de L
// entre L.begin() e IT, y a continuación, L[IT:);
// esto es, l = [ x_{i-1},...,x_1,x_i,...,x_N]
```

En una solución correcta

1: es una expresión simple
2: es una expresion booleana
3: es una o más instrucciones simples

y el coste del bucle es proporcional al número de elementos que tiene el prefijo de L que se ha de invertir. Además está prohibido añadir expresiones o instrucciones fuera de los lugares indicados (p.e. no se pueden añadir más inicializaciones o instrucciones al terminar el bucle) y se valorará negativamente cualquier solución que modifique *it o *p (por ejemplo, intercambiar *p con *it).

```
1: l.begin()

2: p != it

3: l.insert(it, *p);
   p = l.erase(p);
```

1: l.begin()
2: p != it
3: l.insert(l.begin(), *p); // o l.push_front(*p);
p = l.erase(p);
Alternativa (1: y 2: como arriba)
3: l.insert(it, *p); --it;
p = l.erase(p);

Comentari:
3: MAL no invierte nada

Pregunta **8**
Completa
Puntuació 0,50 sobre 0,50
Marca la pregunta

Tiempo estimado: 2 minutos

Si tenemos la función que se muestra a continuación y se le da como entrada el árbol de abajo, ¿qué se imprime en pantalla?

```
void mystery(const BinTree& a) {
      if (not a.empty()) {
          cout << a.value() <<  endl;
          mystery(a.right());
          mystery(a.left());
      }
}
```


Trieu-ne una:

- a. gato, mono, conejo, leon, pato, cebra, vaca
- b. imprime una permutación de las 7 palabras pero no es ninguna de las otras secuencias
- c. cebra, pato, leon, vaca, conejo, mono, gato
- d. gato, mono, leon, conejo, pato, vaca, cebra
- e. la función no se ejecuta corretamente y el programa aborta
- f. la función no compila, aunque se han hecho todos los #include's necesarios
- g. conejo, mono, cebra, pato, vaca, leon, gato
- h. vaca, leon, cebra, pato, gato,  mono, conejo
- i. gato, leon, mono, vaca, pato, conejo, cebra

La teva resposta és correcta.
Les respostes correctes són: gato, mono, conejo, leon, pato, cebra, vaca, la función no compila, aunque se han hecho todos los #include's necesarios

Pregunta **9**

Completa

Puntuació 0,50 sobre 0,50

Marca la pregunta

Tiempo estimado: 2 minutos

Queremos añadir a la clase CjtEstudiants una operación actualiza_notas que dado un vector de N reales actualiza la nota de los primeros N estudiantes de un CjtEstudiants.

```
class CjtEstudiants {
public:
          ...
          actualiza_notas(...)
private:
         ...
};
```

¿Cúal es la cabecera apropiada para actualiza_notas?

Trieu-ne una:

- a. static CjtEstudiants actualiza_notas(vector<double> v);
- b. static CjtEstudiants actualiza_notas(CjtEstudiants& c, vector<double> v);
- c. static void actualiza_notas(vector<double>& v);
- d. void actualizar_notas(const vector<double> v);
- e. void actualizar_notas(const vector<double>& v);
- f. void actualiza_notas(vector<double>s& v) const;
- g. CjtEstudiants actualiza_notas(vector<Estudiant>& v) const;
- h. void actualizar_notas(vector<double> v);

Respuesta correcta

La resposta correcta és: void actualizar_notas(const vector<double>& v);

# 1r Parcial de Teoria PRO2 Q2-2019-2020

LEED MUY ATENTAMENTE ESTAS INSTRUCCIONES

Una vez da comienzo el examen, éste se ha de completar de manera ininterumpida: es muy importante no cerrar la ventana del navegador. Si se termina el tiempo, todas las respuestas almacenadas quedan registradas, no es imprescindible haber pulsado explícitamente el botón de envío.

El examen contiene 8 preguntas: hay una novena pregunta que no puntúa. Una vez avanzáis de una pregunta a la siguiente no podréis retroceder para revisar las respuestas dadas a preguntas anteriores. Verificad con cuidado vuestra respuestas antes de pulsar el botón "Página siguiente".

En cada pregunta damos una estimación de tiempo máximo a dedicarle. La suma de estas estimaciones es 70 minutos, es decir, que tendríais un margen de unos 20 minutos. Usad ese margen con cuidado y tened en cuenta las estimaciones como una orientación que conviene respetar.

Algunas de las preguntas son de respuesta múltiple. En caso de responder correctamente obtendréis la puntuación indicada; si la respuesta es incorrecta la puntuación será "restada". Si se deja sin responder, no hay ninguna penalización. En las preguntas abiertas no funciona "cut & paste".

Si os saca de la sesión o da un error al intentar acabar el cuestionario, que no cunda el pánico. Reiniciar sesión en ATENEA, entrar de nuevo en el examen y os permitirá continuar con el intento en el punto en que se quedó.

En casos de máxima urgencia por algún incidente durante la realización del examen enviad un correo electrónico a Conrado Martínez (conrado@cs.upc.edu) o Borja Valles (valles@cs.upc.edu) o telefonead al 93 413 7849.

Intents permesos: 1

Aquest qüestionari es va tancar el dilluns, 27 abril 2020, 14:00

Límit de temps: 1 hora 30 minuts

## Resum dels vostres intents anteriors

### Resum dels vostres intents anteriors

| Estat | Qualificació / 10,00 | Revisió |
|---|---|---|
| Acabat<br>Enviat dilluns, 27 abril 2020, 13:51 | 6,40 | Revisió |

La vostra qualificació final en aquest qüestionari és 6,40/10,00.