Comenzado el	lunes, 8 de junio de 2020, 10:00
Estado	Finalizado
Finalizado en	lunes, 8 de junio de 2020, 11:40
Tiempo empleado	1 hora 39 minutos
Calificación	**0,4** de 2,0 (**18**%)

Pregunta 1

Finalizado

Puntúa 0,1 sobre 0,7

Decimos que un **palíndromo** es una palabra o frase que se lee igual de izquierda a derecha que de derecha a izquierda. Por ejemplo: "Yo hago yoga hoy". Supongamos que la frase se guarda en un objeto de la clase `vector_t<char>` normalizada a mayúsculas, sin acentos y sin espacios (por ejemplo, “YOHAGOYOGAHOY”).

Se pide, desarrollar un **algoritmo recursivo** que retorne `true` si dada una frase dentro de un `vector_t<char>` es **palíndroma**, o `false` si no lo es, y con la siguiente cabecera:

```
bool is_pal(const vector_t<char>& string, int left, int right)
```

```
bool is_pal(const vector_t<char>& string, int left, int right)
{
  //caso base
  //si el tamaño de la cadena es menor de dos caracteres, es palindroma
  if(string.size() < 2)
   return true

  //caso general
  //recorremos la mitad de la dimension del vector
  for(int i = 0; i < (string.get_size() / 2); i++)
  {
    // si la parte del vector left != right, hacemos el cambio
    if(string[left] != string[right])
    {
    //creamos un auxiliar para realizar el cambio y otro para retornar que no es palindroma
    char aux;
    aux = string[right]; // aux es el elemento de la derecha
    string[left] = string[right]; // el elemento de la posicion izquierda = a la posicion de la
    string[right] = aux;
    b = is_pal( string[i]; left +1, right -1); // recorremos la siguiente posicion del vector
    return false; // no es palindroma
    }

    // en caso de que los caracteres que se comparan son iguales, no realizamos el cambio y llam
    if(string[left] == string[right])
    {  b = is_pal( string[i]; left +1, right -1)
      return false;
    }
  }
}
```

```
bool is_pal(const vector_t<char>& string, int left, int right)
{ if (left >= right) return true;
  return (string[left] == string[right] && is_pal(string, left + 1, right - 1));
}
```

Pregunta 2

Finalizado

Puntúa 0,1 sobre 0,3

Disponemos de una clase `vector_t`, que contiene la siguiente implementación del producto escalar:

```
template<class T> T vector_t<T>::scal_prod(const vector_t<T>& v) const
```

y de una clase `matrix_t` con los siguientes métodos públicos:

- Un constructor que recibe como parámetros su **número de filas** `M` y su **número de columnas** `N`
- Dos métodos de lectura `get_m()` y `get_n()` que permiten recuperar dichos valores `M` y `N`, respectivamente.
- Un método de escritura `set(i, j, v)` que permite fijar el valor del elemento de la fila `i` y la columna `j` de la matriz invocante al valor v, comprobando internamente que: `1 <= i <= M` y `1 <= j <= N`
- Dos métodos de lectura `row(i)` y `col(j)` que devuelven respectivamente la fila `i` o la columna `j` de la matriz invocante como objetos de tipo `vector_t`.

Usando sólo los métodos mencionados, se **pide implementar el producto de dos matrices de forma externa** a la clase `matrix_t` (como un operador, una función independiente o como parte de la función `main`). El cálculo debe **crear una matriz nueva sin modificar los operandos de la multiplicación**.

```
template<class T> T
vector_t<T>::scal_prod(const vector_t<T>& v) const
{ //comprobamos que las matrices sean de la misma dimension
  assert(get_n() == v.get_m());
  matrix_t<T> B;
  B.resize(get_m(), v.get_n());
  for(int i = 1; i <= get_m(); i++)
  {
    for(int j = 1; j <= v.get_n(); j++)
    {
      B.set(i, j, v) += get_row(i)*B.get.col(i);
    }
  }
}
```

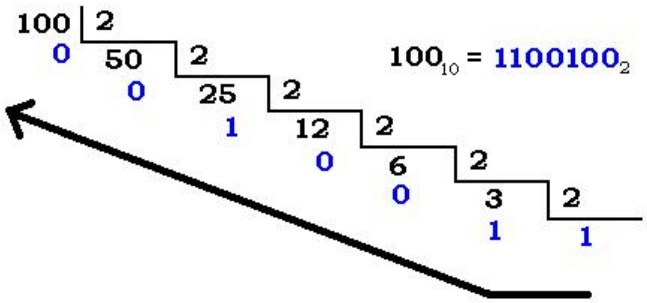
```
Matrix<T> c(a.get_m(), b.get_n());
assert(a.get_n() == b.get_m());
for (int i = 1; i <= a.get_m(); i++)
  for (int j = 1; j <= b.get_n(); j++)
    c.set(i, j, a.row(i).scal_prod(b.col(j)));
```

Pregunta 3

Finalizado

Puntúa 0,1 sobre 0,3

El clásico algoritmo de **conversión de un número en base decimal (base 10) a binario (base 2)**, consiste en dividir el número entre 2 sucesivamente mientras el dividendo sea mayor que 1 y, a continuación, se coge el último cociente hasta el primer resto en orden inverso a como aparecen en la división.



Dado un número entero positivo, se pide desarrollar un **algoritmo de conversión de dicho número entero a binario usando una pila** (`stack_l_t<T>` o `stack_v_t<T>`) mostrando el resultado final en pantalla, y cuya cabecera será:

```
void a_binario(unsigned int e);
```

Usando el ejemplo anterior, si el valor del entero es 100, el resultado esperado por pantalla sería:

```
1100100
```

```
void a_binario(unsigned int e) // se pasa el decimal
{
  int resto = 0, divisor = 0, cociente = 0, cont = 0;     // creamos acumulador del resto y un c
  stack_l_t<T> stack_;         // pila
  while(e > 2 )                // mientras que el dividendo sea mayor que el divisor
  {
    cociente = e / 2;          // cociente = decimal / 2
    resto = e %2;              // en resto de el decimal % 2
    stack_.push(resto);        // guardamos el resto del cociente en la pila
    cont++;                    // aumentamos el acumulador
    e =  cociente;             // seguimos diviendo pero el decimal es el nuevo cociente
  }

  // se imprime por pantalla el resultado de la pila
  for(int i = 0; i < cont; i++)
  {
    if(!stack_.empty())
    {
      int b = stack_.top(); stack_.pop();
      os << b << " ";
    }
  }
}
```

```
void a_binario(unsigned int n)
{
  stack_l_t<unsigned> s;
  while (n > 1)
  {
    s.push(n % 2);
    n /= 2;
  }
  s.push(n);

  while (!s.empty())
  {
    cout << s.top();
    s.pop();
  }
}
```

Pregunta 4

Finalizado

Puntúa 0,1 sobre 0,7

Se ha definido una clase nodo de lista enlazada simple `sll_t` con un método `get_next()`, que devuelve un puntero al siguiente elemento de la lista, y un método `set_next(p)`, que fija el nodo señalado por el puntero `p` como el elemento siguiente. Disponemos también de una clase que gestiona la lista enlazada completa, con un método `get_head()`, que devuelve un puntero al primer elemento de la lista, y un método `set_head(q)`, que fija el nodo señalado por el puntero `q` como primer elemento.

Se pide implementar un método para la clase lista que **invierta el orden de todos sus nodos**, usando solamente bucles, punteros auxiliares, y los métodos `get_head` y `set_head` de la clase lista y `get_next` y `set_next` de la clase nodo.

```
void sll_t<T>::invertir(sll_node_t<T>& q,sll_node_t<T>& p) const
{
  sll_ode_t<T>* aux = get_head(), i = get_tail();
  //recorremos la lista
  while(aux != NULL)
  {
   //apuntamos a los siguientes nodos y lo fijamos
   aux -> set_next(p);
   i -> set_head(q);
   aux -> set_next(i);
   i -> set_next(aux)
   // siguiente nodos
   aux = aux -> get_next();
   i = i -> get_next();
  }
}
```

```
void lista::invert()
{
  nodo* new_head = NULL;
  nodo* aux;
  while (get_head() != NULL) {
    aux = get_head();
```

```
    set_head(aux->get_next());
    aux->set_next(new_head);
    new_head = aux;
  }
  set_head(new_head);
}
```

◄ Prueba objetiva (examen de teoría) del 1 de junio de 2020 (1er llamamiento)

Ir a...

Renuncia a la evaluación continua junio ►