Comenzado el	miércoles, 8 de julio de 2020, 16:02
Estado	Finalizado
Finalizado en	miércoles, 8 de julio de 2020, 17:42
Tiempo empleado	1 hora 40 minutos
Calificación	**0,4** de 2,0 (**18**%)

Pregunta 1

Finalizado

Puntúa 0,1 sobre 0,3

Considérese la clase sll_t<T>. Impleméntese un procedimiento **iterativo (no recursivo)** que invierta el orden de la lista usando un TAD cola (queue_l_t<T> o queue_v_t<T>). La cabecera del método sería:

```
void sll_t<T>::reverse(void);
```

Por ejemplo, si tenemos la siguiente sll_t<int>:

1 → 2 → 3 → 4 → 5 → 6

el resultado esperado en pantalla sería:

6 → 5 → 4 → 3 → 2 → 1

```
void sll_t<T>::reverse(void)
{
  sll_t<T>* aux_ = head_; // aux_ = 6
  while(aux_ != NULL)
  {
    q.push(aux -> get_data()); // metemos en cola aux
    aux_ = aux_ -> get_next(); // apuntamos al siguiente nodo
  }
  while(!q.empty())            // recorremos la cola
  {
    cout << q.front();         //imprimimos
    q.pop();                   // extraemos
  }
}
```

```
template <class T>
void
sll_t<T>::reverse()
{
  queue_l_t<sll_node_t<T>*> q;
  while (!empty())
    q.push(extract_head());
  while (!q.empty())
  {
    insert_head(q.front());
    q.pop();
  }
}
```

Pregunta 2

Finalizado

Puntúa 0,2 sobre 0,3

Disponemos de una clase *Vector3f* sin template que gestiona vectores de tres elementos de tipo *float*. Esta clase contiene un constructor por defecto que no admite parámetros y sus tres elementos pueden leerse o modificarse mediante los métodos públicos *get(int)* y *set(int, float),* donde el parámetro de tipo *int* es el índice del elemento (0, 1 ó 2).

Se pide crear un **operador** * que reciba dos parámetros de tipo *Vector3f* y devuelva el resultado de su producto vectorial como otro *Vector3f*, sabiendo que el producto vectorial se calcula de la siguiente forma:

$$u \times v = \begin{bmatrix} u_0 \\ u_1 \\ u_2 \end{bmatrix} \times \begin{bmatrix} v_0 \\ v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} u_1 * v_2 - u_2 * v_1 \\ u_2 * v_0 - u_0 * v_2 \\ u_0 * v_1 - u_1 * v_0 \end{bmatrix}$$

```
vector_t
operator*(const vector_t& u, const vector_t& v)
{
  assert (u.get_n() == v.get_n());
  vector_t<T> v3f(v.get_n());

  v3f.set(0 ,u.get(1)*v.get(2)- u.get(2)*v.get(1));
  v3f.set(1 ,u.get(2)*v.get(0)- u.get(0)*v.get(2));
  v3f.set(2 ,u.get(0)*v.get(1)- u.get(1)*v.get(0));

  return v3f;
}
```

```
Vector3f operator* (const Vector3f &u, const Vector3f &v) {
  Vector3f w;
  w.set(0, u.get(1) * v.get(2) - u.get(2) * v.get(1));
  w.set(1, u.get(2) * v.get(0) - u.get(0) * v.get(2));
```

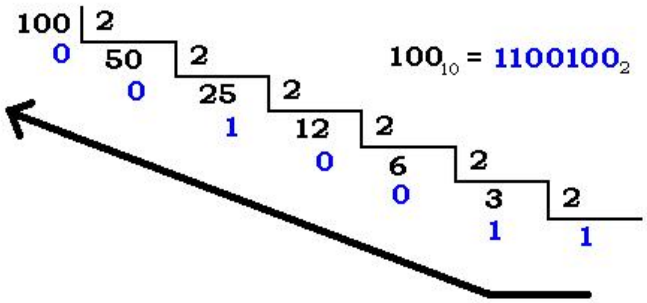
```
    w.set(2, u.get(0) * v.get(1) - u.get(1) * v.get(0));
    return w;
  }
```

Pregunta 3

Finalizado

Puntúa 0,0 sobre 0,7

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 recursivo de conversión de dicho número entero a binario usando una cola** (queue<T>) para guardar el número en binario, y cuya cabecera debe ser:

```
void a_binario_r(unsigned int n, queue<unsigned int>& q);
```

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

1100100

```
void a_binario_r(unsigned int n, queue<unsigned int>& q)
{
  while(n > 1 )
  {
    q.push(n %2);
    n /= 2;
  }
  q.push(n);

  // se imprime por pantalla el resultado de la cola
  while(!q.empty())
  {
    cout << q.front();
    q.pop();
  }
}
```

```
void a_binario_r(unsigned int n, queue<unsigned int>& q)
{
  if (n <= 1)
    q.push(n);
  else
  {
    a_binario_r(n / 2, q);
    q.push(n % 2);
  }
}
```

Pregunta 4

Finalizado

Puntúa 0,0 sobre 0,7

Disponemos de una clase abstracta *sll_t<T>* que gestiona listas enlazadas simples de nodos *sll_node_t<T>*.

La clase nodo *sll_node_t<T>* contiene un constructor de copia, un método *get_data()*, que permite leer su dato interno (de tipo *T*), y los métodos *get_next()* y *set_next(sll_node_t<T>*)*, que permiten leer y modificar el puntero al nodo siguiente.

La clase *sll_t<T>* tiene un único atributo privado *head_* de tipo *sll_node_t<T>** que apunta al primer elemento de la lista. Asimismo, dispone de los métodos *insert_head(sll_node_t<T>*)* y *extract_head()*, que permiten insertar o extraer de forma segura la cabeza de la lista, y los métodos *insert_after(sll_node_t<T>*, sll_node_t<T>*)* y *extract_after(sll_node_t<T>*)*, que permiten insertar o extraer de forma segura el nodo posterior al dado en el primer argumento.

Haciendo uso de estos métodos, el atributo *head_* y tantos punteros auxiliares como se considere necesario, se pide crear un método de la clase *sll_t<T>* y cree una copia de la lista invocante, ordenada de forma iterativa de menor a mayor valor de sus datos internos (manipulando sólo los enlaces entre nodos, no sus datos internos). Se asume que la clase *T* implementa los operadores básicos de comparación (<, >, ==, !=). El método a implementar debe devolver sólo el puntero (*sll_node_t<T>*)* al primer nodo de la nueva secuencia ordenada de nodos.

```
template <class T>
void
sll_t_<T>::copy_ord(const sll_t<T>&)
{
  sll_node_t<T>* nodo = head_, aux = head_;
  while(nodo != NULL)
  {
    aux = nodo -> get_next();
    if(aux < nodo)
    {
      nodo = lista.extract_head();
      aux = insert_after(nodo, aux);
      aux = aux -> set_next(nodo);
    }
  }
}
```

```
template <class T> sll_node_t<T>* sll_t<T>::sort(void) {
  sll_node_t<T> *new_head = NULL, *ptr1 = head_, *ptr2, *aux;
  while (ptr1 != NULL) {
    aux = new sll_node_t<T>(ptr1->get_data());
    if (new_head == NULL || new_head->get_data() > aux->get_data()) {
      aux->set_next(new_head);
```

```
        new_head = aux;
      } else {
        ptr2 = new_head;
        while (ptr2->get_next() != NULL && ptr2->get_next()->get_data() < aux->get_data())
          ptr2 = ptr2->get_next();
        aux->set_next(ptr2->get_next());
        ptr2->set_next(aux);
      }
      ptr1 = ptr1->get_next();
    }
    return new_head;
};
```

◄ ¿Te vas a presentar al examen de teoría (prueba objetiva) de JULIO?

Ir a...

Renuncia a la evaluación continua julio ►