```
template <class T>
class sll_t {
public:
  sll_t(void);
  ~sll_t(void);

  void insert_head(sll_node_t<T>*);
  sll_node_t<T>* extract_head(void);
  sll_node_t<T>* get_head(void) const;

  void insert_after(sll_node_t<T>*, sll_node_t<T>*);
  sll_node_t<T>* extract_after(sll_node_t<T>*);

  bool empty(void) const;
  void write(ostream&) const;

private:
  sll_node_t<T>* head_;
};
```

```
template <class T>
class dll_t {
public:
  dll_t(void);
  ~dll_t(void);

  dll_node_t<T>* get_tail(void) const;
  dll_node_t<T>* get_head(void) const;
  int get_size(void) const;

  bool empty(void) const;
  void push_back(dll_node_t<T>*);
  void push_front(dll_node_t<T>*);
  dll_node_t<T>* pop_back(void);
  dll_node_t<T>* pop_front(void);
  dll_node_t<T>* erase(dll_node_t<T>*);
  dll_node_t<T>* find(const T&) const;

  ostream& write(ostream& = cout) const;

private:
  dll_node_t<T>* head_;
  dll_node_t<T>* tail_;
  int            sz_;
};
```

```
template<class T>
class vector_t
{
public:
  vector_t(const int = 0);
  vector_t(const vector_t<T>&);
  ~vector_t(void);

  void resize(const int);
  int get_size(void) const;
  T& at(const int);
  T at(const int) const;
  T& operator[](const int);
  T operator[](const int) const;
  vector_t<T>& operator=(const vector_t<T>&);

  ostream& write(ostream& os = cout) const;
  istream& read(istream& is = cin);

private:
  T*  base_;
  int n_;
  void build_(const int);
  void destroy_(void);
};

template<class T> ostream& operator<<(ostream&, const vector_t<T>&);
```

EJERCICIO 1. Escribe una función en C++ con la siguiente cabecera:

```
vector_t<int> rotate(const vector_t<int>& v, const int r)
```

que devuelve un nuevo vector *vector_t<int>*, y que contiene los elementos del vector original rotados *r* veces hacia la derecha. Por ejemplo, si v= [10,21,32,43,54,65,76,87,98], el resultado esperado para la llamada a rotate(v, 5) sería [54,65,76,87,98,10,21,32,43].

Solución:

```
vector_t<int> rotate(const vector_t<int>& v, const int r)
{ int n = v.get_size();
  vector_t<int> resultado(n);

  for (int i = 0; i < n; i++)
    resultado[(i + r) % n] = v[i];

  return resultado;
}
```

EJERCICIO 2. Escribe una función en C++ con la siguiente cabecera:

```
bool es_palindroma(const vector_t<char>& cadena);
```

y verifique si la cadena de entrada es **palíndroma**. Un palíndromo es una palabra o frase que se lee igual de izquierda a derecha y de derecha a izquierda, sin tener en cuenta los espacios ni los caracteres especiales. La función **debe utilizar una pila** (*stack_v_t<char>* o *stack_l_t<char>*) para realizar la verificación. Por ejemplo, el resultado esperado sería *true* para la siguiente cadena de entrada:

['Y','O','H','A','G','O','Y','O','G','A','H','O','Y']

Solución:

```
bool es_palindroma(const vector_t<char>& cadena)
{ stack_l_t<char> pila;

  // llenar la pila con la primera mitad de la palabra
  int mitad = cadena.get_size() / 2;
  for (int i = 0; i < mitad; i++)
    pila.push(cadena[i]);

  // si la longitud es impar, omitir el carácter central
  int inicio = mitad;
  if (cadena.size() % 2 != 0) inicio++;

  bool palindroma = true;
  for (int i = inicio; i < cadena.get_size() && palindroma; i++)
    if (pila.top() == cadena[i]) pila.pop();
    else                         palindroma = false;

  return palindroma;
}
```

EJERCICIO 3. Dado un tamaño *n*, y un objeto de la clase *vector_t<char>* de longitud *n*, se pide desarrollar un procedimiento **recursivo** en C++ con la siguiente cabecera:

```
void all_bin(vector_t<char>& binario, const int i = 0);
```

que genere por pantalla todas las cadenas de números binarios de longitud *n*. Por ejemplo, si ejecutamos *all_bin(binario)* con un tamaño *n* igual a 4, el resultado debería ser:

0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111

NOTA: El último parámetro es el índice que itera sobre la cadena de caracteres que, por defecto, empieza en 0. En la evaluación de este ejercicio se tendrá muy en cuenta la implementación del caso base y del caso general, y la eficiencia de los bucles (si fueran necesarios).

Solución:

```
void all_bin(vector<char>& binario, const int i = 0)
{
  if (i == binario.size())
    cout << binario << " ";
  else
  { binario[i] = '0';
    all_bin(binario, i + 1);
    binario[i] = '1';
    all_bin(binario, i + 1);
  }
}
```

EJERCICIO 4. Para la clase *dll_t<int>*, desarrollar el método especializado

```
template<> dll_node_t<int>* dll_t<int>::find(const int v);
```

que busca un valor dentro de la lista, devolviendo el puntero del nodo que lo contiene o NULL si no lo encuentra.

Solución:

```
template<>
dll_node_t<int>* dll_t<int>::find(const int v)
{ dll_node_t<int> *ptr = get_head();
  while (ptr != NULL && ptr->get_data() != v)
    ptr = ptr->get_next();
  return ptr;
}
```

EJERCICIO 5. Para la clase *sll_t<T>*, desarrollar el método especializado que elimina y libera todos los elementos de **posiciones pares** de la lista, y cuya cabecera es:

```
template <class T> void sll_t<T>::erase_evens(void);
```

Solución:

```
template <class T>
void sll_t<T>::erase_evens(void)
{ sll_node_t<T>* aux = get_head();

  while (aux != NULL && aux->get_next() != NULL)
  { delete erase_after(aux); // Borrar el siguente a aux y liberarlo
    aux = aux->get_next(); // Solo adelantar aux un nodo, par ya borrado
  }
}
```

EJERCICIO 6. Considérese la función de Ackermann. Implementar un algoritmo recursivo que calcule el valor de la función para cualquier par de términos.

$$A(x, y) \equiv \begin{cases} y + 1 & \text{if } x = 0 \\ A(x - 1, 1) & \text{if } y = 0 \\ A(x - 1, A(x, y - 1)) & \text{otherwise.} \end{cases}$$

Solución:

```
int Ackerman(int x, int y)
{ if (x == 0) return y + 1;
  else
  { if (y == 0) return Ackerman(x - 1, 1);
    else        return Ackerman(x - 1, Ackerman(x, y - 1));
  }
}
```

Última modificación: jueves, 2 de mayo de 2024, 14:58