```cpp
template <class T>
class sll_t {
private:
  sll_node_t<T>* head_;

public:
  sll_t(void);
  virtual ~sll_t(void);

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

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

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

  sll_node_t<T>* search(const T& d) const;
};
```

```cpp
template <class T>
class sll_node_t {
private:
  sll_node_t<T>* next_;
  T              data_;

public:
  sll_node_t(void);
  sll_node_t(const T& data);
  ~sll_node_t(void);

  void set_next(sll_node_t<T>* next);
  sll_node_t<T>* get_next(void) const;

  void set_data(const T& data);
  const T& get_data(void) const;

  ostream& write(ostream& os) const;
};
```

EJERCICIO 1. Se solicita crear una nueva clase *template Pila* llamada *template<class T> class stack_sll_t* que contenga lo siguiente:

(a) Atributo privado de la clase para implementar una pila mediante una *sll_t*

(b) Método *Push*

(c) Método *Pop*

(d) Método *Top*

(e) Método *Empty*

Solución:

```cpp
private:
```

```cpp
  sll_t<T> l_;
```

```cpp
template<class T> void stack_sll_t<T>::push(const T& dato) {
  sll_node_t<T>* nodo = new sll_node_t<T>(dato);
  assert(nodo != NULL);
  l_.insert_head(nodo);
}

template<class T> void stack_sll_t<T>::pop(void) {
  assert(!empty());
  delete l_.extract_head();
}

template<class T> const T& stack_sll_t<T>::top(void) const {
  assert(!empty());
  return l_.get_head()->get_data();
}

template<class T> bool stack_sll_t<T>::empty(void) const {
  return l_.empty();
}
```

EJERCICIO 2. Se tiene un programa principal que utiliza una pila *stack* P. Utilizando sólo los métodos propios de una pila, desarrolla una función recursiva llamada *sumaElementos* que reciba por parámetros una pila y sume los elementos de la misma de forma recursiva.

```cpp
int sumaElementos(stack_t<int>& p);
```

?

Solución:

```
int sumaElementos(stack_t<int>& p) {
  int cima;
  if (p.empty()) return 0;
  else {
    cima = p.top();
    p.pop();
    return cima + sumaElementos(p);
  }
}
```

EJERCICIO 3. Implementar el procedimiento:

```
void sll_union(const sll_t<int>& A, const sll_t<int>& B, sll_t<int>& C);
```

que realiza la unión (tipo conjunto) de dos listas no ordenadas A y B con elementos no repetidos, y devuelve el resultado en la lista C.

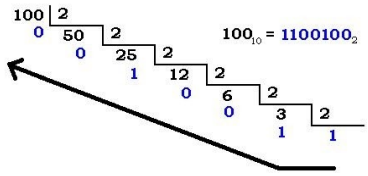
Por ejemplo, si A={2,1,4,3} y B={1,5,3,6}, el resultado sería C={6,3,5,1,4,2}.

Solución:

```
void sll_union(const sll_t<int>& A, const sll_t<int>& B, sll_t<int>& C)
{ sll_node_t<int> *ptr = A.head();
  // inserta los elementos de A que no están en B
  while (ptr != NULL)
  { if (B.search(ptr->get_data()) == NULL)
      C.insert_head(new sll_node_t<int>(ptr->get_data()));
    ptr = ptr->get_next();
 }

 // inserta todos los elementos de B
 ptr = B.get_head();
 while (ptr != NULL)
 { C.insert_head(new sll_node_t<int>(ptr->get_data()));
   ptr = ptr->get_next();
 }
}
```

EJERCICIO 4. El clásico algoritmo de conversión de un número *n* en base decimal (base 10) a otra base *b*, consiste en dividir *n* entre la base *b* sucesivamente mientras el dividendo sea mayor o igual que *b* y, a continuación, se coge el último cociente hasta el primer resto en orden inverso. Por ejemplo, si se convierte el número 100 en base 10 a base 2, las operaciones a realizar ser ían las siguientes:



Dado un número entero positivo *n* y una base entera *b*, se pide desarrollar un algoritmo recursivo de conversión de dicho número entero a cualquier base *b* (con 1 < *b* < 10) usando una cola (*queue_t*<*T*>) para guardar el número en dicha base, y cuya cabecera debe ser:

```
void to_base(const unsigned n, const unsigned short b, queue_t<unsigned>& q);
```

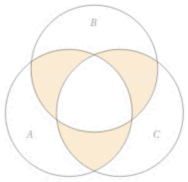
Usando el ejemplo anterior, si el valor del entero es 100, y la base es 2, el resultado esperado por pantalla sería: 1100100. En caso de ser base 8, el resultado sería: 144.

NOTA: se valorará la eficiencia del algoritmo y el esquema recursivo implementado (caso base y caso general).

Solución:

```
void to_base(const unsigned n, unsigned short b, queue<unsigned>& q)
{
  if (n < b) q.push(n);
  else
  { to_base(n / b, b, q);
    q.push(n % b);
  }
}
```

EJERCICIO 5. Desarrollar el procedimiento que calcule el resultado sombreado en la figura usando solo el tipo block_t y las operaciones sobre bits AND (&), OR (|) y NOT (~).




```
block_t op1(const block_t A, const block_t B, const block t_C);
```

Por ejemplo, dados los conjuntos A = {1, 2, 3, 4, 7}, B = {3, 4, 5, 6} y C = {4, 6, 7, 8, 9}, el resultado esperado sería el siguiente: op1(A, B, C) -> { 3, 6, 7 }

Solución:

```
block_t op1(const block_t A, const block_t B, const block_t C)
{
  return ((A & C) | (A & B) | (B & C)) & ~ (A & (B & C);
}
```

Última modificación: miércoles, 24 de abril de 2024, 15:25