Estructuras de Datos y de la Información

Ingeniería Técnica en Informática de Gestión. Curso 2007/2008

Ejercicios del Tema 3

Las implementaciones de los TAD se realizarán como clases de C++. En cada caso, se formulará el invariante de la representación y las especificaciones pre/post de los procedimientos y funciones que realicen las operaciones del TAD. En el caso de los TADs genéricos, la implementación se realizará mediante clases parametrizadas (*templates* o plantillas).

1. Desarrolla un TAD *TComplejo* que ofrezca el tipo *TComplejo* junto con las operaciones adecuadas para construir números complejos y calcular con ellos.

2. Desarrolla un TAD *TMonomio* para representar los monomios como coeficiente y grado, con las operaciones de creación, consultar el grado y consultar el coeficiente.

   A continuación, utilizando el TAD *TMonomio* desarrolla un TAD *TPolinomio* que ofrezca el tipo de los polinomios, con operaciones para crear el polinomio nulo, añadir un nuevo monomio a un polinomio, consultar el coeficiente de un determinado grado, consultar el grado del polinomio, sumar polinomios y evaluar un polinomio para un valor dado de la indeterminada.

   Para el TAD *TPolinomio* construye dos implementaciones:

   – representando los polinomios mediante un vector de monomios ordenados por el grado; y
   – representando los polinomios mediante una lista enlazada de monomios ordenados por el grado.

3. Añade a la clase *TPilaDinamica*<*TElem*> dos nuevas operaciones:
   – *fondo*: devuelve el elemento del fondo de una pila no vacía.
   – *inversa*: devuelve una nueva pila con los elementos de la pila original apilados en orden inverso.

4. Las pilas formadas por números naturales del intervalo [0..N–1] (para cierto $N \geq 2$ fijado de antemano) se pueden representar por medio de números, entendiendo que un número natural P cuya representación en base N tenga "1" como dígito de mayor peso representa la pila formada por los restantes dígitos de la representación de P en base N (excepto el de mayor peso), siendo la cima el dígito de menor peso. Por ejemplo, si N = 10, el número 1073 representa la pila que contiene los números 0, 7, 3, con 0 en la base y 3 en la cima. Implementa el TAD *TPilaNat* con esta representación. Comprueba que esta representación permite implementar todas las operaciones de las pilas con coste $O(1)$.

5. Desarrolla un TAD genérico *TDosPilas*<*TElem*> que representa una pareja de pilas que pueden manejarse independientemente del modo usual.
   – Desarrolla una implementación estática de este TAD, representando la pareja de pilas con ayuda de un único vector de almacenamiento. Organiza la implementación de modo que las dos pilas crezcan en sentidos opuestos, cada una desde uno de los dos extremos del vector.
   – Desarrolla una segunda implementación utilizando la clase *TPilaDinamica*<*TElem*>.

6. Especifica e implementa un TAD genérico *TLista*<*TElem*> que represente a las listas de elementos con las siguientes operaciones:
   – *Nuevo*: crea una lista vacía.
   – *Cons*: añade un elemento al principio de la lista.
   – *ponDr*: añade un elemento al final de la lista.
   – *primero*: devuelve el primer elemento de la lista.
   – *resto*: elimina el primer elemento de la lista.
   – *ultimo*: devuelve el último elemento de la lista.
   – *inicio*: elimina el último elemento de la lista.
   – *concatena*: añade al final de una lista los elementos de otra.

– *esVacio*: determina si la lista está vacía.
– *numElem*: devuelve el número de elemento de la lista
– *elemEn*: devuelve el elemento que está en una cierta posición de la lista, identificada por su número de orden.

La lista se debe implementar como una lista enlazada.

7. Explica razonadamente qué mostrará este programa por pantalla.

```
typedef TPilaDinamica< TPilaDinamica<int> > TPilas;
typedef TPilaDinamica< TPilaDinamica<int>* > TPuntPilas;


TPilas pilas ( int max ) {
  TPilas res;
  TPilaDinamica<int> pila;

  for ( int i = 0; i < max; i++ ) {
    pila = TPilaDinamica<int>();
    for ( int j = 0; j < i; j++ )
      pila.apila(j);
    res.apila(pila);
  }

  return res;
}


int suma ( TPilas pilaDePilas ) {
  TPilaDinamica< int > pila;
  int res;

  res = 0;
  while ( ! pilaDePilas.esVacio() ) {
    pila = pilaDePilas.cima();
    pilaDePilas.desapila();
    while ( ! pila.esVacio() ) {
      res += pila.cima();
      pila.desapila();
    }
  }

  return res;
}
```

```
TPuntPilas pilas2 ( int max ) {
  TPuntPilas res;
  TPilaDinamica<int>* pila;

  for ( int i = 0; i < max; i++ ) {
    pila = new TPilaDinamica<int>();
    for ( int j = 0; j < i; j++ )
      pila->apila(j);
    res.apila(pila);
  }

  return res;
}


int suma2 ( TPuntPilas pilaDePilas )
{
  TPilaDinamica< int >* pila;
  int res;

  res = 0;
  while ( ! pilaDePilas.esVacio() ) {
    pila = pilaDePilas.cima();
    pilaDePilas.desapila();
    while ( ! pila->esVacio() ) {
      res += pila->cima();
      pila->desapila();
    }
  }

  return res;
}
```

```
int main(int argc, char* argv[])
{
  TPilas pilaDePilas;
  TPuntPilas pilaDePilas2;

  pilaDePilas = pilas( 4 );
  cout << suma( pilaDePilas ) + suma( pilaDePilas ) << endl;

  pilaDePilas2 = pilas2( 4 );
  cout << suma2( pilaDePilas2 ) + suma2( pilaDePilas2 ) << endl;
}
```