Pilhas e Filas

Algoritmos e Estruturas de Dados

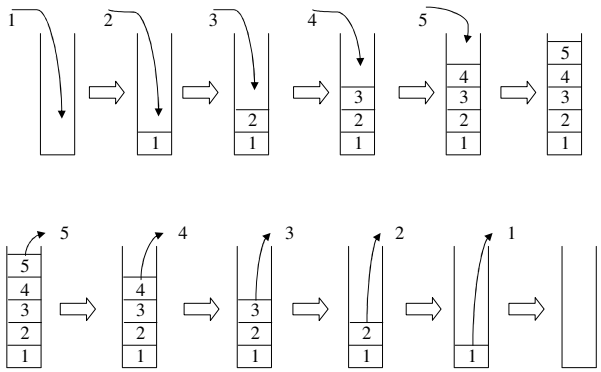
2005/2006

Pilhas

- Pilha
  - estrutura de dados linear em que a inserção e a remoção de elementos de uma sequência se faz pela mesma extremidade, designada por **topo** da pilha
  - uma pilha pode ser considerada como uma *restrição* de lista
  - porque é uma estrutura de dados mais simples que a lista, é possível obter implementações mais eficazes
  - o conceito de iterador **não se aplica** a esta estrutura de dados
  - a pilha é uma estrutura do tipo LIFO *(Last-In-First-Out)*
  - Operações mais usuais:
    - criar uma pilha vazia
    - adicionar/remover um elemento a uma pilha
    - verificar qual o último elemento colocado na pilha

Pilhas

- Operações com pilhas



Pilhas: implementação baseada em listas

- Declaração da classe ***LStack*** em C++ (secção pública)

```
template <class T> class LStack {
public:
   LStack();
   LStack(const LStack & stk);
   ~LStack();

   bool isEmpty() const;
   const T & top() const;
   void makeEmpty();
   void pop();
   void push(const T &x);
   T topAndPop();
   const LStack & operator =(const LStack & stk);
   // ...
};
```

Pilhas: implementação baseada em listas

- Declaração da classe ***LStack*** em C++ (secção privada)

```
template <class T> class LStack {
// ...
private:
   class ListNode {      // classe privada
   public:
      T element;
      ListNode *next;

      ListNode(const T & elem, ListNode *n = 0) : element(elem), next(n)
      { };
   };

   ListNode *topOfStack;
};
```

Pilhas: implementação baseada em listas

- classe ***LStack*** : construtor, destrutor, makeEmpty(), isEmpty()

```
template <class T> LStack<T>:: LStack()
{ topOfStack = 0; }

template <class T> LStack<T>:: ~LStack()
{ makeEmpty(); }

template <class T> void LStack<T>:: makeEmpty() {
   ListNode *no = topOfStack;
   while ( no != 0 ) {
      ListNode *aposNo = no->next;
      delete no;
      no = aposNo;
   }

template <class T> bool LStack<T>:: isEmpty() const
{ return topOfStack == 0; }
```

Pilhas: implementação baseada em listas

- classe ***LStack*** : top(), push()

```
template <class T>
const T & LStack<T>:: top() const
{
   if ( isEmpty() ) throw Underflow();
   return topOfStack->element;
}

template <class T>
void LStack<T>:: push(const T & x)
{
   topOfStack = new ListNode(x, topOfStack);
}
```

Pilhas: implementação baseada em listas

- classe ***LStack*** : pop() e topAndPop()

```
template <class T>
void LStack<T>:: pop()
{
   if ( isEmpty() ) throw Underflow();
   ListNode *oldTop = topOfStack;
   topOfStack = topOfStack->next;
   delete oldTop;
}

template <class T>
T & LStack<T>:: topAndPop()
{
   T topItem = top();
   pop();
   return topItem;
}
```

Pilhas: implementação baseada em vectores

- Declaração da classe ***VStack*** em C++

```
template <class T> class VStack {
public:
   explicit VStack(int capacity = 100);

   bool isEmpty() const;
   const T & top() const;
   void makeEmpty();
   void pop();
   void push(const T &x);

private:
   vector<T> theArray;
   int topOfStack;
};
```

Pilhas: implementação baseada em vectores

- classe ***VStack*** : isEmpty(), makeEmpty() e top()

```
template <class T>
bool VStack<T>:: isEmpty() const {
   return topOfStack == -1;
}

template <class T>
void VStack<T>:: makeEmpty() {
   topOfStack = -1;
}

template <class T>
const T & VStack<T>:: top() const {
   if ( isEmpty() ) throw Underflow();
   return theArray[topOfStack];
}
```

Pilhas: implementação baseada em vectores

- classe ***VStack*** : push(), pop() e topAndPop()

```
template <class T> void VStack<T>:: push(const T & x) {
   theArray.push_back();
   topOfStack++;
}

template <class T> void VStack<T>:: pop() {
   if ( isEmpty() ) throw Underflow();
   theArray.pop_back();
   topOfStack --;
}

template <class T> T & LStack<T>:: topAndPop() {
   if ( isEmpty() ) throw Underflow();
   T elem = theArray[topOfStack--];
   theArray.pop_back();
   return elem;
}
```

Pilhas (Standard Template Library - STL)

- class ***stack***

- Alguns métodos:
  - stack & operator =(const stack &)
  - bool empty() const
  - T size() const
  - T & top()
  - void push(const T &)
  - void pop()
  - bool operator ==(const stack &, const stack &)
  - bool operator <(const stack &, const stack &)

(http://www.sgi.com/tech/stl/stack.html)

Filas

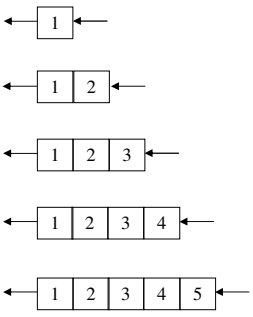
- Fila
  - estrutura de dados linear em que a inserção e a remoção de elementos de uma sequência se faz por extremidades opostas, geralmente designadas por **cabeça** e **cauda** da fila
  - uma fila pode ser considerada como uma *restrição* de lista
  - porque é uma estrutura de dados mais simples que a lista, é possível obter implementações mais eficazes
  - o conceito de iterador **não se aplica** a esta estrutura de dados
  - a pilha é uma estrutura do tipo FIFO *(First-In-First-Out)*

  - Operações mais usuais:
    - criar uma fila vazia
    - adicionar/remover um elemento a uma fila
    - verificar qual o elemento da cabeça da fila (elemento mais antigo)

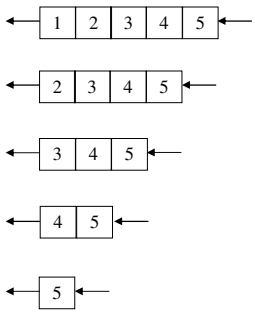
Filas

- Operações com filas

Inserir: 1, 2, 3, 4, 5

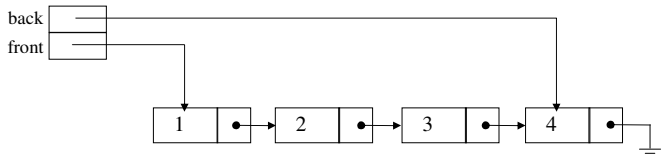


Remover: 1, 2, 3, 4, 5



Filas: implementação baseada em listas

- LQueue<int>



Filas: implementação baseada em listas

- Declaração da classe ***LQueue*** em C++ (secção pública)

```
template <class T> class LQueue {
public:
   LQueue();
   LQueue(const LQueue & q);
   ~LQueue();

   bool isEmpty() const;
   const T & getFront() const;
   void makeEmpty();
   void dequeue();
   void enqueue(const T &x);

   const LQueue & operator =(const LQueue & q);
   // ...
};
```

Filas: implementação baseada em listas

- Declaração da classe ***LQueue*** em C++ (secção privada)

```
template <class T> class LQueue {
// ...
private:
   class ListNode {      // classe privada
   public:
      T element;
      ListNode *next;

      ListNode(const T & elem, ListNode *n = 0) : element(elem), next(n)
      { };
   };

   ListNode *front, *back;
};
```

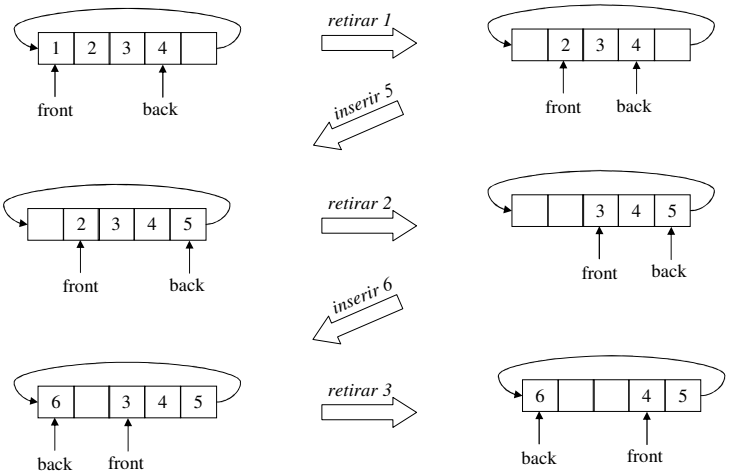
Filas: implementação baseada em listas

- classe ***LQueue*** : enqueue(), dequeue()

```
template <class T> void LQueue<T>:: enqueue(const T & x)
{
    ListNode *oldBack = back;
   back = new ListNode(x,0);
   if (oldBack) oldBack->next = back;
   if (!front) front = back;
}

template <class T> void LQueue<T>:: dequeue()
{
   if ( isEmpty() ) throw Underflow();
   ListNode *oldFront = front;
   front = front->next;
   delete oldFront;
   if (!front) back = 0;
}
```

Filas: implementação baseada em vectores



Filas: implementação baseada em vectores

- Declaração da classe ***CQueue*** em C++

```
template <class T> class CQueue {
public:
   explicit CQueue(int capacity = 100);
   bool isEmpty() const;
   const T & getFront() const;
   void makeEmpty();
   void dequeue();
   void enqueue(const T &x);

private:
   vector<T> theArray;
   bool isFull() const;
   int currentSize;
   int front, back;
   void increment(int & x);
};
```

Nota: não se permite a extensão do vector

Filas: implementação baseada em vectores

- classe ***CQueue*** : enqueue(), dequeue()

```
template <class T> void CQueue<T>:: enqueue(const T & x) {
   if ( isFull() ) throw Underflow();
   increment(back);
   theArray[back] = x; currentSize++;
}

template <class T> void CQueue<T>:: dequeue() {
   if ( isEmpty() ) throw Underflow();
   currentSize --;
   increment(front);
}

template <class T> void CQueue<T>:: increment(int & x)
{ if ( ++x == theArray.size() ) x=0; }
```

Filas (Standard Template Library - STL)

- class ***queue***

- Alguns métodos:
  - queue & operator =(const queue &)
  - bool empty() const
  - T size() const
  - T & front()
  - T & back()
  - void push(const T &)
  - void pop()
  - bool operator ==(const queue &, const queue &)
  - bool operator <(const queue &, const queue &)

(http://www.sgi.com/tech/stl/queue.html)

Aplicações de pilhas

Notação RPN *(Reverse Polish Notation)*

- expressões matemáticas em que os operadores surgem a seguir aos operandos (notação pósfixa)
- vantagem: não requer parêntesis nem regras de precedência

Notação infixa

- os operadores binários surgem entre os operandos

Notação infixa: 2 * (4 + 5) / 3
Notação RPN: 2 4 5 + * 3 /

Aplicações de pilhas

Avaliação de expressões RPN : *algoritmo*

1. Processar sequencialmente os elementos da expressão.
   Para cada elemento:
   - 1.1 Se o elemento for um número (operando), colocá-lo na pilha
   - 1.2. Se o elemento for um operador
     - 1.2.1. Retirar os dois elementos do topo da pilha
     - 1.2.2. Processar os elementos de acordo com o operador
     - 1.2.3. Colocar o resultado na pilha
2. Retirar o (único) elemento da pilha. É o resultado

Aplicação de pilhas: avaliação de expressões

```
class Elemento
{
public:
  float valor;
  char op;
  Elemento(float v=0, char o='?'): valor(v), op(o) {};
};

float calcOp(float v1, float v2, char op)
{
   switch(op) {
      case '+' : return v1+v2;
      case '-' : return v1-v2;
      case '*' : return v1*v2;
      case '/' : return v1/v2;
      default: throw OperacaoInvalida();
  }
}
```

Aplicação de pilhas: avaliação de expressões

```
// uso de pilha implementada através da classe LStack
float avaliaRPN(string expressao)   // simplificação: números de 1 dígito apenas
{
   LStack<Elemento> pilha;
   for ( int i=0; i<expressao.length(); i++ ) {
         if ( expressao[i]>='0' &&  expressao[i]<='9')  // e numero
               pilha.push(Elemento(expressao[i]-48,'?'));
         else {
               float res = calcOp(pilha.topAndPop().valor,
                     pilha.topAndPop().valor, expressao[i]);
               pilha.push(Elemento(res,'?'));
         }
   }
   return pilha.topAndPop().valor;
}
```

Aplicação de pilhas: avaliação de expressões

```
// uso de pilha implementada através da classe stack (STL)
        float avaliaRPN(string expressao)   // simplificação: números de 1 dígito apenas
        {
           stack<Elemento> pilha;
           for ( int i=0; i<expressao.length(); i++ ) {
                   if ( expressao[i]>='0' &&  expressao[i]<='9')  // e numero
                           pilha.push(Elemento(expressao[i]-48,'?'));
                   else {
                           float num1 = pilha.top().valor; pilha.pop();
                           float num2 = pilha.top().valor; pilha.pop();
                           float res = calcOp(num1, num2, expressao[i]);
                           pilha.push(Elemento(res,'?'));
                   }
           }
           float res = pilha.top().valor;  pilha.pop();
           return res;
        }
```