Chapter 3
Stacks and Queues

授課老師：詹寶珠教授

助教：黃詰琳
Templates in C++ (X)

• Template function in C++ makes it easier to reuse classes and functions.
• A template can be viewed as a variable that can be instantiated to any data type, irrespective of whether this data type is a fundamental C++ type or a user-defined type.
Selection Sort Template(X)

Template <class KeyType>
void sort(KeyType *a, int n)
// sort the n KeyTypes a[0] to a[n-1] into nondecreasing order
{
    for (int i = 0; i < n; i++)
    {
        int j = i;
        // find smallest KeyType in a[i] to a[n-1]
        for (int k = i+1; k < n; k++)
            if (a[k] < a[j]) { j = k;}
        // interchange
        KeyType temp = a[i]; a[i] = a[j]; a[j] = temp;
    }
}

float farray[100];
int intarray[200];

............
sort(farray, 100);
sort(intarray, 200);
Template (Cont.) (X)

• Can we use the sort template for the Rectangle class?
• Well, not directly. We’ll need to use operator overloading to implement “>” for Rectangle class.
Stack

- What is a stack? A stack is an ordered list in which insertions and deletions are made at one end called the top. It is also called a Last-In-First-Out (LIFO) list.
Stack (Cont.)

- Given a stack $S = (a_0, \ldots, a_{n-1})$, $a_0$ is the bottom element, $a_{n-1}$ is the top element, and $a_i$ is on top of element $a_{i-1}$, $0 < i < n$. 

![Diagram of stack operations: Push (Add) and Pop (Delete)]
System Stack

• Whenever a function is invoked, the program creates a structure, referred to as an activation record or a stack frame, and places it on top of the system stack.
Template <class KeyType>
class Stack
{
    // objects: A finite ordered list with zero or more elements
    public:
        Stack (int MaxStackSize = DefaultSize);
        // Create an empty stack whose maximum size is MaxStackSize
        Boolean IsFull();
        // if number of elements in the stack is equal to the maximum size
        // of the stack, return TRUE(1) else return FALSE(0)
        void Add(const KeyType& item);
        // if IsFull(), then StackFull(); else insert item into the top of the stack.
        Boolean IsEmpty();
        // if number of elements in the stack is 0, return TRUE(1) else return FALSE(0)
        KeyType* Delete(KeyType& );
        // if IsEmpty(), then StackEmpty() and return 0;
        // else remove and return a pointer to the top element of the stack.
};
Implementation of Stack by Array

Array index: 0, 1, 2, 3, ..., n-1

Array: a₀, a₁, a₂, ..., aₙ₋₁

Diagram shows the structure of an array representing a stack.
Queue

- A queue is an ordered list in which all insertions take place at one end and all deletions take place at the opposite end. It is also known as First-In-First-Out (FIFO) lists.
Template <class KeyType>
class Queue
{
    // objects: A finite ordered list with zero or more elements
    public:
        Queue(int MaxQueueSize = DefaultSize);
        // Create an empty queue whose maximum size is MaxQueueSize
        Boolean IsFull();
        // if number of elements in the queue is equal to the maximum size of
        // the queue, return TRUE(1); otherwise, return FALSE(0)
        void Add(const KeyType& item);
        // if IsFull(), then QueueFull(); else insert item at rear of the queue
        Boolean IsEmpty();
        // if number of elements in the queue is equal to 0, return TRUE(1)
        // else return FALSE(0)
        KeyType* Delete(KeyType&);
        // if IsEmpty(), then QueueEmpty() and return 0;
        // else remove the item at the front of the queue and return a pointer to it
};
Queue Manipulation Issue

- It’s intuitive to use array for implementing a queue. However, queue manipulations (add and/or delete) will require elements in the array to move. In the worse case, the complexity is of \( O(\text{MaxSize}) \).
Shifting Elements in Queue

- Front and rear arrows indicate shifting elements in a queue.
- The queue's structure changes as elements are added or removed at the front and rear.
Circular Queue

• To resolve the issue of moving elements in the queue, circular queue assigns next element to q[0] when rear == MaxSize – 1.
• Pointer front will always point one position counterclockwise from the first element in the queue.
• Queue is empty when front == rear. But it is also true when queue is full. This will be a problem.
Circular Queue (Cont.)

front = 0; rear = 4

front = n-4; rear = 0
Circular Queue (Cont.)

• To resolve the issue when front == rear on whether the queue is full or empty, one way is to use only MaxSize – 1 elements in the queue at any time.

• Each time when adding an item to the queue, newrear is calculated before adding the item. If newrear == front, then the queue is full.

• Another way to resolve the issue is using a flag to keep track of last operation. The drawback of the method is it tends to slow down Add and Delete function.
Subtyping and Inheritance in C++ (X)

- Inheritance is used to express subtype relationships between two ADTs.
- If B inherits from A, then B IS-A A. Also, A is more general than B.
  - VW Beetle IS-A Car; Eagle IS-A Bird
Inheritance (X)

- A derived class inherits all the non-private members (data and functions) of the base class.
- Inherited members from public inheritance have the same level of access in the derived class as they did in the base class.
- The derived class can reuse the implementation of a function in the base class or implement its own function, with the exception of constructor and destructor.
Class Inheritance Example (X)

class Bag
{
public:
    Bag (int MaxSize = DefaultSize); // constructor
    virtual ~Bag();               // destructor
    virtual void Add(int);        // insert element into bag
    virtual int* Delete (int&);   // delete element from bag
    virtual Boolean IsFull();     // return TRUE if the bag is full; FALSE otherwise
    virtual Boolean IsEmpty();    // return TRUE if the bag is empty; FALSE otherwise

protected:
    virtual void Full();          // action when bag is full
    virtual void Empty();         // action when bag is empty
    int *array;                   // size of array
    int MaxSize;                  // highest position in array that contains an element
    int top;                     // highest position in array that contains an element
}
whips  

Class Inheritance Example (Cont.)

(X)

class Stack : public Bag
{
public:
    Stack(int MaxSize = DefaultSize); // constructor
    ~Stack(); // destructor
    int* Delete(int&); // delete element from stack
};

Stack::Stack(int MaxStackSize) : Bag(MaxStackSize) {} // Constructor for Stack calls constructor for Bag

Stack::~Stack() {} // Destructor for Bag is automatically called when Stack is destroyed. This ensures that array is deleted.

int* Stack::Delete(int& x)
{
    if (IsEmpty()) {Empty(); return 0;}
    x = array[top--];
    return &x;
}
Class Inheritance Example (Cont.) (X)

Bag b(3);       // uses Bag constructor to create array of size 3
Stack s(3);     // uses Stack constructor to create array of size 3

b.Add(1); b.Add(2); b.Add(3);
// use Bag::Add. Bag::Add calls functions Bag::IsFull and Bag::Full

s.Add(1); s.Add(2); s.Add(3);
// Stack::Add not defined, so use Bag::Add. Bag::Add calls Bag::IsFull
// and Bag::Full because these have not been redefined in Stack

int x;
b.Delete(x);    // uses Bag::Delete, which calls Bag::IsEmpty and Bag::Empty
s.Delete(x);
// uses Stack::Delete, which calls Bag::IsEmpty and Bag::Empty because these
// have not been redefined in Stack.
The Maze Problem
Allowable Moves

X

N

[i-1][i]

[i-1][j]

[i-1][j+1]

NE

[i][j]

N

NW

[i][j-1]

[i][j]

[i][j+1]

E

[i+1][i]

[i+1][j]

[i+1][j+1]

SE

[i+1][j-1]

SW
The Maze Problem (Cont.)

• Stack is used in solving the maze problem for storing the coordinates and direction.
• Use of another \((m+2)\times(p+2)\) array to mark any position that has been visited before.
The Maze Problem (Cont.)

- Surround boundary by 1
- Following directions \{N, NE, E, SE, S, SW, W, NW\}
- Use constant move[i].a and move[i].b to record the next movement,
  - e.g. move[2].a=1, move[2].b=0, meaning moving East by add the position(1,0)
- Use another array[m+2][p+2] to flag by 1 of the positions which have been visited.
  Initial ← 0
The Maze Problem (Cont.)

<設起始，將目前位置及方向放到stack中>

when the stack is not empty
{
    pop stack;
    設定行走方向;
    while (d<8)
    {
        設定下一點之位置 (h, g)，若已達終點，return;
        if (此點為 0，且尚未經過)
        {
            記下回去的路，(push stack)
            向前走 (i, j <-- h, g);
        }
        else
        {
            d++; // 下一方向
        }
    }
}
Evaluation Expression in C++

- When evaluating operations of the same priorities, it follows the direction from left to right.

<table>
<thead>
<tr>
<th>Priority</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Unary minus, !</td>
</tr>
<tr>
<td>2</td>
<td>*, /, %</td>
</tr>
<tr>
<td>3</td>
<td>+, -</td>
</tr>
<tr>
<td>4</td>
<td>&lt;, &lt;=, &gt;=, &gt;</td>
</tr>
<tr>
<td>5</td>
<td>==, !=</td>
</tr>
<tr>
<td>6</td>
<td>&amp;&amp;</td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>
Postfix Notation

Expressions are converted into Postfix notation before compiler can accept and process them.

\[ X = \frac{A}{B} - C + D \times E - A \times C \]

Infix: \( A/B-C+D\times E-A\times C \)
Postfix: \( AB/C-DE*+AC*- \)

Postfix operation:
1: When encounter operand, push it into stack
2: When encounter operator, then pop out the last two operands and perform the operation on the two operands.
   Push the result back to stack.
\[
\begin{align*}
A / B - C + D \times E - A \times C & \quad AB / C - D \times E + A \times C - \\
\text{Operation} & \quad \text{Postfix} \\
T_1 = A / B & \quad T_1 C - D \times E + A \times C - \\
T_2 = T_1 - C & \quad T_2 D \times E + A \times C - \\
T_3 = D \times E & \quad T_2 T_3 + A \times C - \\
T_4 = T_2 + T_3 & \quad T_4 A \times C - \\
T_5 = A \times C & \quad T_4 T_5 - \\
T_6 = T_4 - T_5 & \quad T_6
\end{align*}
\]
### Postfix Notation (cont.)

**Example:** $A/B-C+D\times E-A\times C$

<table>
<thead>
<tr>
<th>Stack</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>AB</td>
</tr>
<tr>
<td>/</td>
<td>AB/C</td>
</tr>
<tr>
<td>-</td>
<td>AB/C-D</td>
</tr>
<tr>
<td>+</td>
<td>AB/C-D</td>
</tr>
<tr>
<td>+</td>
<td>AB/C-DE</td>
</tr>
<tr>
<td>+</td>
<td>AB/C-DE*+A</td>
</tr>
<tr>
<td>-</td>
<td>AB/C-DE*+AC</td>
</tr>
<tr>
<td>-</td>
<td>AB/C-DE*+AC*-</td>
</tr>
</tbody>
</table>
Infix to Postfix

(1) If it is an operand, then output the operand

(2) If it is an operator

if (priority) this operator \( \leq \) operator in stack
output the operator in stack
push this operator

Note:
‘(’ : 若未在stack中，則當成最大priority，表示一定只會放入；
若已在stack中，則設成最小priority，表示一定不會被取出，
直到‘)’出現
## Infix to Postfix

**Example:** \( A/D + E - (B - D \times C + E \times F - E) \)

<table>
<thead>
<tr>
<th>State</th>
<th>Stack</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A )</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>( A/D )</td>
<td>/AD</td>
<td></td>
</tr>
<tr>
<td>( A/D+ )</td>
<td>+AD/</td>
<td></td>
</tr>
<tr>
<td>( A/D+E )</td>
<td>+AD/E</td>
<td></td>
</tr>
<tr>
<td>( A/D+E- ) (</td>
<td>- (AD/E+</td>
<td></td>
</tr>
<tr>
<td>( A/D+E- (B )</td>
<td>-(AD/E+B</td>
<td></td>
</tr>
<tr>
<td>( A/D+E- (B- )</td>
<td>-(AD/E+B-</td>
<td></td>
</tr>
<tr>
<td>( A/D+E- (B-D )</td>
<td>-(AD/E+BD</td>
<td></td>
</tr>
<tr>
<td>( A/D+E- (B-D* )</td>
<td>-(AD/E+BD*</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State</th>
<th>Stack</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A/D+E- (B-D\times C) )</td>
<td>-(AD/E+BDC*</td>
<td></td>
</tr>
<tr>
<td>( A/D+E- (B-D\times C+E) )</td>
<td>-(AD/E+BDC*E</td>
<td></td>
</tr>
<tr>
<td>( A/D+E- (B-D\times C+E*F) )</td>
<td>-(AD/E+BDC<em>E</em>F</td>
<td></td>
</tr>
<tr>
<td>( A/D+E- (B-D\times C+E*F-E) )</td>
<td>-(AD/E+BDC<em>E</em>F-E</td>
<td></td>
</tr>
</tbody>
</table>

Infix: \( A/D + E - (B - D \times C + E \times F - E) \) \( \Rightarrow \) Postfix: \( AD/E + BDC* - EF* + E - \)
Multiple Stacks and Queues

- **Two stacks case:**

  Stack A → Stack B

  0 1 2 3 4

  \[0\ 1\ 2\ 3\ 4\ m-4\ m-3\ m-2\ m-1\]

- **More than two stacks case**

  0 \(\lfloor m/n\rfloor - 1\) \(2\lfloor m/n\rfloor - 1\)

  \[0\ \lfloor m/n\rfloor - 1\ 2\lfloor m/n\rfloor - 1\ m-1\]


  \[t[0] t[1] t[2] t[n]\]