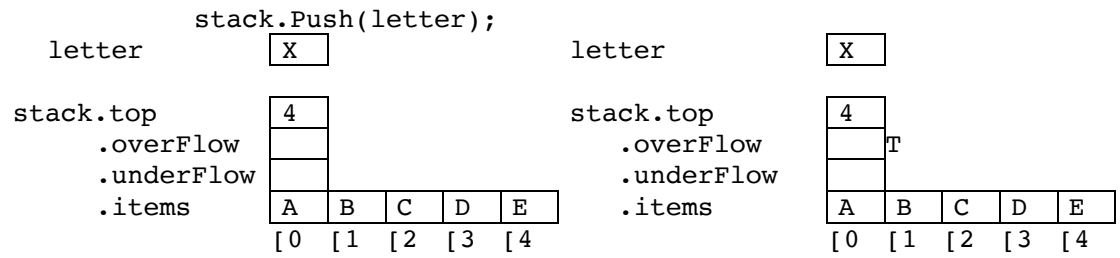
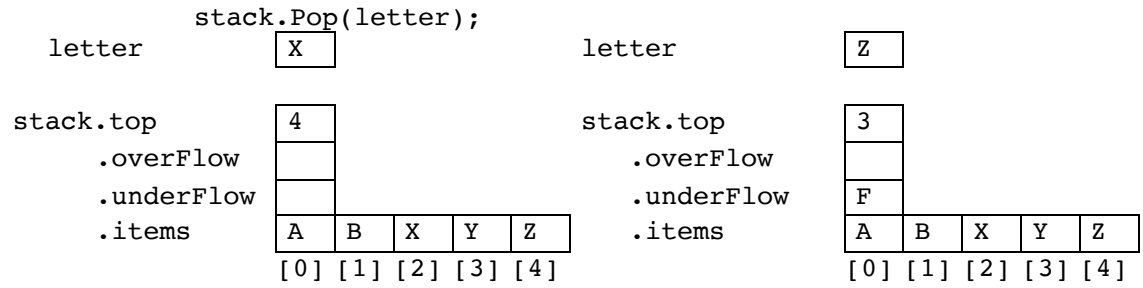
Stack ADT:

- Ex. 5



- Ex. 7



- Ex. 10

The answer is (d) for the first statement; the parameter to Push may be an expression.

The answer is (a) for the second statement; the parameter to Pop is a reference parameter.

- Ex. 15

```
void ReplaceItem(StackType& stack, ItemType oldItem, ItemType
newItem)
{
  StackType tempStack;
  ItemType tempItem;
  while (!stack.IsEmpty())
  {
    stack.Pop(tempItem);
    if (tempItem.ComparedTo(oldItem) == EQUAL)
      tempStack.Push(newItem);
    else
      tempStack.Push(tempItem);
  }
  // restore stack
  while (!tempStack.IsEmpty())
  {
    tempStack.Pop(tempItem);
```

```
    stack.Push(tempItem);
  }
}
```

- Ex. 16

```
// Represent candy container as a stack PezJar.
// Eat yellow candies
WHILE PezJar is not empty
    PezJar.Pop(candy)
    IF candy != yellow candy
        tempJar.Push(candy)

// Restore candy container
WHILE tempJar is not empty
    tempJar.Pop(candy)
    PezJar.Push(candy)
```

- Ex. 18

```
bool Identical(StackType stack1, StackType stack2)
{
  StackType<ItemType> tempStack;
  ItemType item1;
  ItemType item2;
  bool same = true;

  while (!stack1.IsEmpty() && !stack2.IsEmtpy() && same)
  {
    stack1.Pop(item1);
    stack2.Pop(item2);
    if (item1.ComparedTo(item2) == EQUAL)
      tempStack.Push(item1);
    else
    {
      same = false;
      stack1.Push(item1);
      stack2.Push(item2);
    }
  }
  same = same && stack1.IsEmpty() && stack2.IsEmpty();
  // restore stacks
  while (!tempStack.IsEmpty())
  {
    tempStack.Pop(item);
    stack1.Push(item);
    stack2.Purh(item);
  }
  return same;
}
```

Queue ADT:

- Ex. 22

(a)

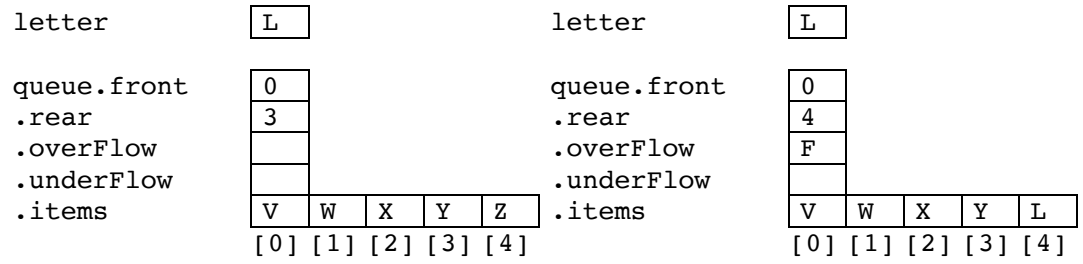
```
1 0 4 (on separate lines)
5 16 0 3 (on separate lines)
```

(b)

```
6 4 6 0 (on separate lines)
6 5 0 (on separate lines)
```

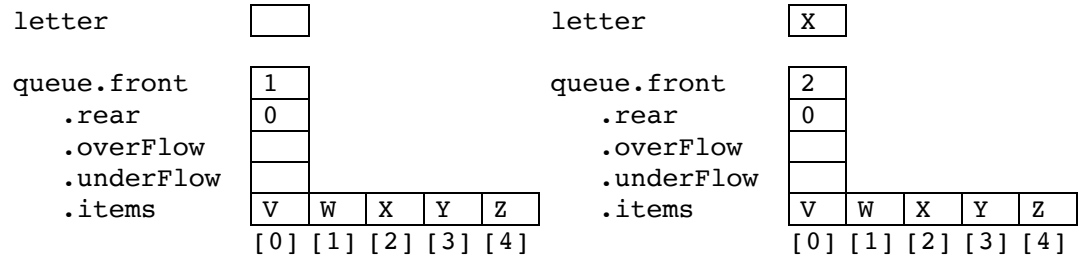
- Ex. 26

```
queue.Enqueue(letter);
```



- Ex. 29

```
queue.Dequeue(letter);
```



- Ex. 33

(a) The function as client code, using operations from the `QueType` class.

```
{
  ItemType item;
  ItemType first;
  QueType tempQ;

  while (!queue.IsEmpty())
  {
    queue.Dequeue(item);
    tempQ.Enqueue(item);
  }
  tempQ.Dequeue(first);
  queue.Enqueue(first);
  while (!tempQ.IsEmpty())
  {
```

```
    tempQ.Dequeue(item);
    queue.Enqueue(item);
  }
  return first;
}
```

(b) The function as a new member function of the `QueType` class.

```
ItemType Front();          // prototype
ItemType QueType::Front()
{
  return items[(front + 1) % maxQue];
}
```

- Ex. 34

```
{
  QueType<ItemType> tempQ;
  ItemType item;

  while (!queue.IsEmpty())
  {
    queue.Dequeue(item);
    if (item == oldItem)
      tempQ.Enqueue(newItem);
    else
      tempQ.Enqueue(item);
  }
  while (!tempQ.IsEmpty())
  {
    tempQ.Dequeue(item);
    queue.Enqueue(item);
  }
}
```

- Ex. 40

The `MakeEmpty` operation is a logical operation that sets the structure to empty. A class constructor is a C++ construct that is implicitly called when an object of the type is defined. `MakeEmpty` is under the control of the client program and can be applied any number of times.