BAKI ALİ NEFT MƏKTƏBİ
BAKU HIGHER OIL SCHOOL

Programming and Computer Applications-2

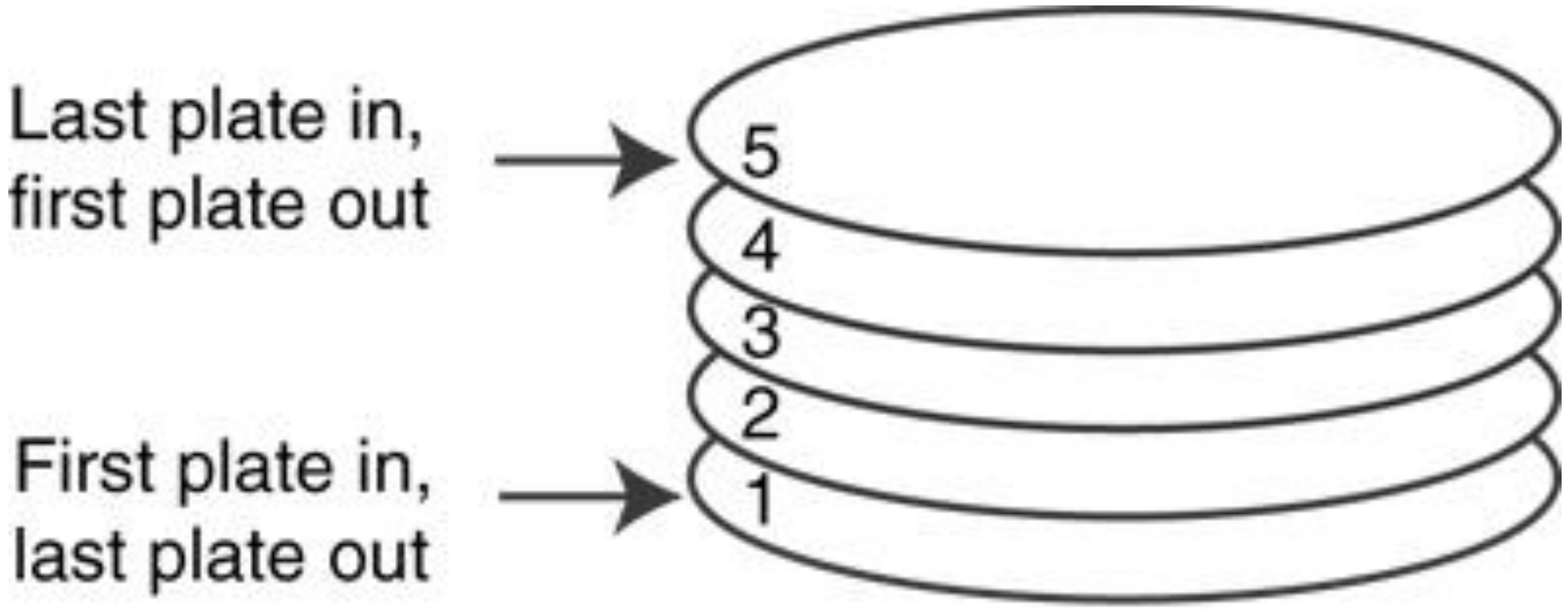
Stacks

Instructor : PhD, Associate Professor Leyla Muradkhanli

Introduction

- **Stack**: a LIFO (last in, first out) data structure
- Examples:
  - plates in a cafeteria
  - return addresses for function calls
- Implementation:
  - static: fixed size, implemented as array
  - dynamic: variable size, implemented as linked list

A LIFO Structure

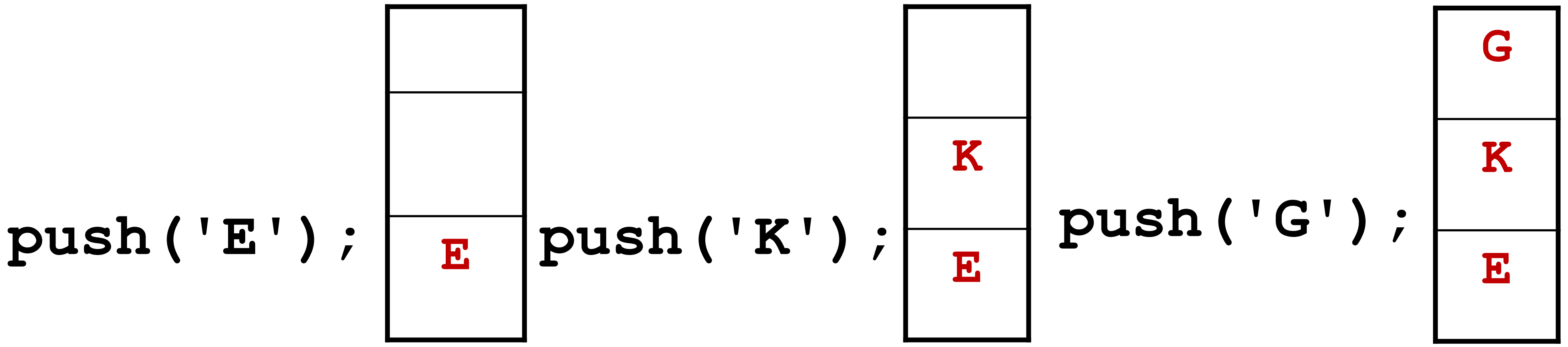


Stack Operations and Functions

- Operations:
  - **push**: add a value onto the top of the stack
  - **pop**: remove a value from the top of the stack
- Functions:
  - **`isFull`**: **`true`** if the stack is currently full, *i.e.*, has no more space to hold additional elements
  - **isEmpty**: **`true`** if the stack currently contains no elements

Stack Operations - Example

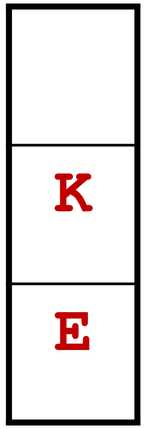
- A stack that can hold **`char`** values:



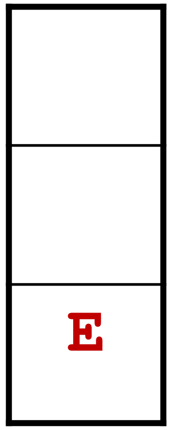
Stack Operations - Example

- A stack that can hold **`char`** values:

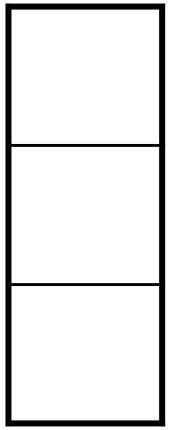
`pop();`
(remove `G`)



`pop();`
(remove `K`)



`pop();`
(remove `E`)



Dynamic Stacks

- Grow and shrink as necessary
- Can't ever be full as long as memory is available
- Implemented as a linked list

The STL `stack` Container

- Stack template can be implemented as a **`vector`**, a **linked list**, or a **`deque`**
- Implements **`push`**, **`pop`**, and **`empty`** member functions
- Implements other member functions:
  - **`size`**: number of elements on the stack
  - **`top`**: reference to element on top of the stack

Defining a `stack`

- Defining a stack of **`char`s**, named **`cstack`**, implemented using a **`vector`**:

```
stack< char, vector<char> > cstack;
```

- implemented using a **list**:

```
stack< char, list<char> > cstack;
```

- implemented using a **`deque`**:

```
stack< char > cstack;
```

- Spaces are required between consecutive >>, << symbols

Stack member functions

emplace

Constructs and inserts new element at the top of stack.

empty

Tests whether stack is empty or not.

operator =

Assigns new contents to the stack by replacing old ones.

pop

Removes top element from the stack.

Stack member functions

push

Inserts new element at the top of the stack.

size

Returns the total number of elements present in the stack.

swap

Exchanges the contents of stack with contents of another stack.

top

Returns a reference to the topmost element of the stack.

```
#include <iostream>
#include <stack>
using namespace std;
int main()
{
   // Define an STL stack
   stack< int> myStack;
   // Push values onto the stack.
   for (int i = 1; i <=10; i++)
   {
      cout << "Pushing " << i << endl;
      myStack.push(i);
   }
   // Display the size of the stack.
   cout << "The size of the stack is ";
   cout << myStack.size() << endl;
```

```cpp
// Pop the values of the stack.
    for (int i = 1; i <=10; i++)
    {
        cout << "Popping " << myStack.top() << endl;
        myStack.pop();
    }
    return 0;
}
```

```
Pushing 1
Pushing 2
Pushing 3
Pushing 4
Pushing 5
Pushing 6
Pushing 7
Pushing 8
Pushing 9
Pushing 10
The size of the stack is 10
Popping 10
Popping 9
Popping 8
Popping 7
Popping 6
Popping 5
Popping 4
Popping 3
Popping 2
Popping 1
```

stack::emplace() function

```cpp
#include <iostream>
#include <stack>
using namespace std;
int main() {
   stack<int> s;
   for (int i = 1; i <=5; ++i)
      s.emplace(i);
   while (!s.empty()) {
      cout << s.top() << endl;
      s.pop();
   }
   return 0;
}
```

```
Stack contents are
5
4
3
2
1
```

stack::empty() function

```cpp
#include <iostream>
#include <stack>
using namespace std;
int main() {
   stack<int> s;
   if (s.empty())
      cout << "Stack is empty." << endl;
   s.emplace(1);
   if (!s.empty())
      cout << "Stack is not empty." << endl;
   return 0;
}
```

```
Stack is empty.
Stack is not empty.
```

stack::operator= function (copy version)

```
#include <iostream>
#include <stack>
using namespace std;
int main() {
   stack<int> s1;
   stack<int> s2;
   for (int i = 1; i <=5; ++i)
      s1.push(i);
   s2 = s1;
   cout << "Contents of stack s2" << endl;
   while (!s2.empty()) {
      cout << s2.top() << endl;
      s2.pop();
   }
return 0;
}
```

```
Contents of stack s2
5
4
3
2
1
```

stack::operator= function (move version)

```
#include <iostream>
#include <stack>
using namespace std;
int main() {
   stack<int> s1;
   stack<int> s2;
   for (int i = 1; i <=5; ++i)
      s1.push(i);
   cout << "Size of stack s1 before move operation = " << s1.size()
<< endl;
```

stack::operator= function (move version)

```
s2 = move(s1);
cout << "Size of stack s1 after move operation = " << s1.size() << endl;
cout << "Contents of stack s2" << endl;
   while (!s2.empty()) {
      cout << s2.top() << endl;
      s2.pop();
   }
   return 0;
}
```

```
Size of stack s1 before move operation = 5
Size of stack s1 after move operation = 0
Contents of stack s2
5
4
3
2
1
```

stack::pop() function

```
#include <iostream>
#include <stack>
using namespace std;
int main() {
   stack<int> s;
   for (int i = 1; i<=5; ++i)
      s.emplace(i);
   while (!s.empty()) {
      cout << s.top() << endl;
      s.pop();
   }
return 0;
}
```

```
Stack contents are
5
4
3
2
1
```

stack::push() function (copy version)

```
#include <iostream>
#include <stack>
using namespace std;
int main() {
   stack<int> s;
   for (int i = 1; i <= 5; ++i)
      s.push(i);
   cout << "Stack contents are" << endl;
   while (!s.empty()) {
      cout << s.top() << endl;
      s.pop();
   }
return 0;
}
```

```
Stack contents are
5
4
3
2
1
```

stack::push() function (move version)

```
#include <iostream>
#include <stack>
using namespace std;
int main() {
   stack<int> s1;
   stack<int> s2;
   for (int i = 1; i <=5; ++i)
      s1.push(i);
   while (!s1.empty()) {
      s2.push(move(s1.top()));
      s1.pop();
   }
```

stack::push() function (move version)

```
cout << "Stack contents are" << endl;
   while (!s2.empty()) {
      cout << s2.top() << endl;
      s2.pop();
   }
   return 0;
}
```

```
Stack contents are
1
2
3
4
5
```

stack::size() function

```
#include <iostream>
#include <stack>
using namespace std;
int main() {
   stack<int> s;
   cout << "Initial size of stack = " << s.size() << endl;
   for (int i = 1; i <=5; ++i)
      s.push(i);
   cout << "Stack size after push operation = " << s.size() << endl;
   return 0;
}
```

```
Initial size of stack = 0
Stack size after push operation = 5
```

stack::swap() function

```cpp
#include <iostream>
#include <stack>
using namespace std;
int main() {
   stack<int> s1;
   stack<int> s2;
   for (int i = 1; i <=5; ++i)
      s1.push(i);
   for (int i = 0; i < 3; ++i)
      s2.push(100 + i);
   s1.swap(s2);
   cout << "Contents of stack s1 after swap operation" << endl;
```

stack::swap() function

```
while (!s1.empty()) {
        cout << s1.top() << endl;
        s1.pop();
    }
    cout << endl;
    cout << "Contents of stack s2 after swap operation" << endl;
    while (!s2.empty()) {
        cout << s2.top() << endl;
        s2.pop();
    }
    return 0;
}
```

```
Contents of stack s1 after swap operation
102
101
100
Contents of stack s2 after swap operation
5
4
3
2
1
```

stack::top() function

```cpp
#include <iostream>
#include <stack>
using namespace std;
int main() {
   stack<int> s;
   for (int i = 1; i <=5; ++i)
      s.emplace(i);
   while (!s.empty()) {
      cout << s.top() << endl;
      s.pop();
   }
   return 0;
}
```

Stack contents are

```
5
4
3
2
1
```