GAME2001 Data Structures and Algorithms

Fall 2020

Week 6

Stacks and Queues I

Stacks and Queues

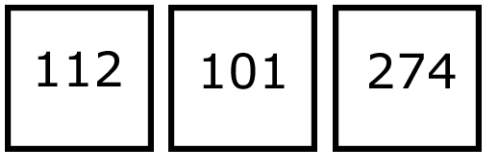
- adapters that are built on top of data structures we've previously seen such as arrays and linked lists
- created and used for the sole purpose of processing a task before being discarded
  - shorter life span than the array or the link list
- restricted-access structures
  - can only access one element at a time

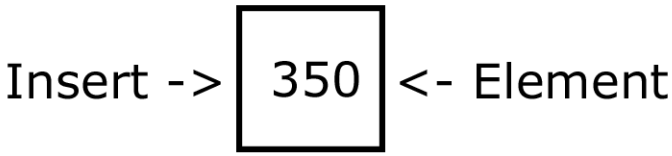
Stacks

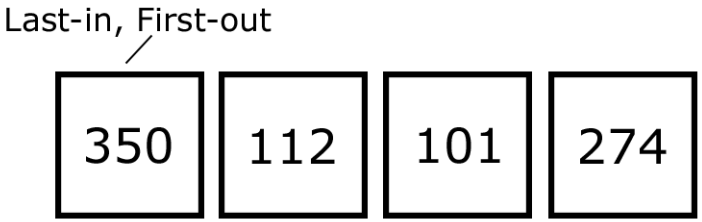
- last-in, first-out (LIFO) data structure
  - the last item inserted into the container is the first item removed from the container
- only one item can be inserted or removed at one time
- restrict the access to only one at a time
  - to access the elements deep within the container of a stack, items need to be removed from the top

Stacks

- no access by array indexes
- no access by searching
- no sorting
  - all could violate the LIFO order







Stacks

- main operations include
  - inserting items onto the stack
  - removing items from the stack
  - taking a peek at the top of the stack
    - returning the topmost element without popping it off of the stack

- Stack
  - useful tool to process an algorithm

Stacks

- Uses
  - any situation where LIFO access is desired
  - when implementing a function-call stack for a virtual machine of a game scripting system
  - when writing a compiler for parsing tokens in a game script
  - different areas in artificial intelligance

C++ array-based stack

```
template<class T>
class Stack
{
   public:
      Stack(int size, int growBy = 1) : m_size(0),
            m_top(-1), m_array(0), m_growSize(0)
      {
         if(size)
         {
            m_size = size;
            m_array = new T[m_size];

            assert(m_array != NULL);
            memset(m_array, 0, sizeof(T) * m_size);

            m_growSize = ((growBy > 0) ? growBy : 0);
         }
      }

      ~Stack()
      {
         if(m_array != NULL)
         {
            delete[] m_array;
            m_array = NULL;
         }
      }

      void push(T val)
      {
         assert(m_array != NULL);

         if(isFull())
         {
            Expand();
         }

         m_array[++m_top] = val;
      }

      void pop()
      {
         if(!isEmpty())
         {
            m_top--;
         }
      }

      const T& top()
      {
         assert(m_array != NULL);
         assert(m_top >= 0);

         return m_array[m_top];
      }
```

C++ array-based stack

```
57      int GetSize()     { return m_top + 1; }
58      int GetMaxSize() { return m_size; }
59      bool isEmpty()    { return (m_top == -1); }
60      bool isFull()     { return (m_top == m_size - 1); }
61
62   private:
63      bool Expand()
64      {
65         if(m_growSize <= 0)
66            return false;
67
68         assert(m_array != NULL);
69
70         T *temp = new T[m_size + m_growSize];
71         assert(temp != NULL);
72         memcpy(temp, m_array, sizeof(T) * m_size);
73         delete[] m_array;
74         m_array = temp;
75
76         m_size += m_growSize;
77
78         return true;
79      }
80
81   private:
82      T *m_array;
83      int m_top;
84      int m_size;
85      int m_growSize;
86};
```

C++ array-based stack example

```
1 #include <iostream>
2 #include "Stack.h"
3
4 using namespace std;
5
6 int main(int args, char **argc)
7 {
8     cout << "Stack Example" << endl;
9     cout << endl;
10
11    Stack<int> sList(5);
12
13    sList.push(101);
14    sList.push(201);
15    sList.push(301);
16    sList.push(401);
17    sList.push(501);
18    sList.pop();
19    sList.push(601);
20
21    cout << "Contents of the stack:";
22    while(sList.isEmpty() == false)
23    {
24        cout << " " << sList.top();
25        sList.pop();
26    }
27    cout << endl << endl;
28
29    return 1;
30 }
```

```
Stack Example

Contents of the stack: 601 401 301 201 101
```

UnorderedArray Stack

```
#include "Arrays.h"

template<class T>
class Stack
{
   public:
      Stack(int size, int growBy = 1)
      {
         assert(size > 0 && growBy >= 0);
         m_container = new UnorderedArray<T>(size, growBy);
         assert(m_container != NULL);
      }

      ~Stack()
      {
         if(m_container != NULL)
         {
            delete m_container;
            m_container = NULL;
         }
      }

      bool isEmpty()
      {
         assert(m_container != NULL);
         return (m_container->GetSize() == 0);
      }

      void push(T val)
      {
         assert(m_container != NULL);
         m_container->push(val);
      }

      void pop()
      {
         assert(m_container != NULL);
         m_container->pop();
      }

      const T& top()
      {
         assert(m_container != NULL);
         return (*m_container)[m_container->GetSize() - 1];
      }

      int GetSize()
      {
         assert(m_container != NULL);
         return m_container->GetSize();
      }

      int GetMaxSize()
      {
         assert(m_container != NULL);
         return m_container->GetMaxSize();
      }
   private:
      UnorderedArray<T> *m_container;
};
```

LinkList Stack

```
#include "DoublyLinkList.h"

template<class T>
class Stack
{
   public:
      Stack()  { }
      ~Stack() { }

      void push(T val)
      {
         m_container.Push(val);
      }

      void pop()
      {
         m_container.Pop();
      }

      const T& top()
      {
         LinkIterator<T> it;
         it = m_container.Last();
         return *it;
      }

      int GetSize()
      {
         return m_container.GetSize();
      }

      bool isEmpty()
      {
         return (m_container.GetSize() == 0);
      }

   private:
      LinkList<T> m_container;
      int m_size;
};
```

Character Matching With Stacks

- match tokens such as brackets, parentheses, and curly braces
  - useful in the implementation of a compiler, such as the ones found in game scripting systems

Character Matching With Stacks

```
#include <iostream>
#include "Stack.h"

using namespace std;

void PrintError(char ch, int index)
{
   cout << "   Error " << ch << " at " << index
        << "." << endl;
}

void ParseString(char *str, int size)
{
   if(str == NULL || size <= 0)
   {
      cout << "   Error with parameters!" << endl << endl;
      return;
   }

   Stack<char> sList(size);
   char ch = 0;
   int errors = 0;

   for(int i = 0; i < size; i++)
   {
      switch(str[i])
      {
         case '{':
         case '(':
         case '[':
            sList.push(str[i]);
            break;
         case '}':
         case ')':
         case ']':
            if(sList.isEmpty() == false)
            {
               ch = sList.top();

               if((ch != '{' && str[i] == '}') ||
                  (ch != '(' && str[i] == ')') ||
                  (ch != '[' && str[i] == ']'))
                  {
                     PrintError(ch, i + 1);
                     errors++;
                  }
               sList.pop();
            }
            break;
      }
   }

   if(sList.isEmpty() && errors == 0)
      cout << "   No Parsing Errors." << endl << endl;
   else if(sList.isEmpty() == false)
      cout << "   Unclosed Characters: " << sList.GetSize()
           << "." << endl << endl;
}
```

Character Matching With Stacks

```
int main(int args, char **argc)
{
   cout << "Character Matching with Stacks Example" << endl;
   cout << endl;

   char str[] = { '{', '(', 'a', '[', '5', ']', ')', '}' };
   int size = strlen(str);
   cout << "Parsing str." << endl;
   ParseString(str, size);

   char str2[] = { '{', ')', 'b', '[', '9', ']', ')', '}' };
   size = strlen(str2);
   cout << "Parsing str2." << endl;
   ParseString(str2, size);

   cout << endl;

   return 1;
}
```

```
Character Matching with Stacks Example

Parsing str.
   No Parsing Errors.

Parsing str2.
   Error { at 2.
```

STL Stack

- By default, STL stacks are implemented with a STL deque (double-ended queue)
  - can also be with the STL vector or STL list

Function	Description
`stack()` `stack(container)`	Constructors that will create a stack container; an optional second constructor that takes a container from which the stack is to be copied
`empty()`	Returns `true` if the container is empty or `false` if it is not
`pop()`	Removes the last item inserted into the list
`push(T val)`	Inserts the object `val` into the container
`top()`	Returns a reference to the element on the top of the stack
`size()`	Returns the number of elements in the container

STL Stack Example

```
1 #include <iostream>
2 #include <stack>
3 #include <vector>
4 #include <list>
5
6 using namespace std;
7
8 template<class T>
9 void DisplayStack(T &stack)
10 {
11    cout << "(Size - " << stack.size() << ") :";
12
13    while(stack.empty() == false)
14    {
15       cout << " " << stack.top();
16       stack.pop();
17    }
18
19    cout << "." << endl;
20 }
```

```
22 int main(int args, char **argc)
23 {
24    cout << "STL Stacks Example" << endl << endl;
25
26    stack<int> intStack;
27    stack<int, vector<int> > vecStack;
28    stack<int, list<int> > listStack;
29
30    for(int i = 0; i < 5; i++)
31    {
32       intStack.push(11 + i);
33       vecStack.push(22 + i);
34       listStack.push(33 + i);
35    }
36
37    // Display normal (deque) integer stack.
38    cout << "       Contents of the int stack ";
39    DisplayStack(intStack);
40
41    // Display vector integer stack.
42    cout << "Contents of the int vector stack ";
43    DisplayStack(vecStack);
44
45    // Display link list integer stack.
46    cout << "  Contents of the int list stack ";
47    DisplayStack(listStack);
48    cout << endl;
```

STL Stack Example

```
50    // Calling empty() to test if container is empty.
51    if(intStack.empty() == true)
52       cout << "The int stack is empty." << endl;
53    else
54       cout << "The int stack is NOT empty." << endl;
55
56    // Calling empty() to test if container is empty.
57    if(vecStack.empty() == true)
58       cout << "The vec int stack is empty." << endl;
59    else
60       cout << "The vec int stack is NOT empty." << endl;
61
62    // Calling empty() to test if container is empty.
63    if(listStack.empty() == true)
64       cout << "The list int stack is empty." << endl;
65    else
66       cout << "The list int stack is NOT empty." << endl;
67
68    cout << endl;
69
70    return 1;
71 }
```

```
STL Stacks Example

       Contents of the int stack (Size - 5) : 15 14 13 12 11.
Contents of the int vector stack (Size - 5) : 26 25 24 23 22.
  Contents of the int list stack (Size - 5) : 37 36 35 34 33.

The int stack is empty.
The vec int stack is empty.
The list int stack is empty.
```