FRESNO STATE
Discovery. Diversity. Distinction.

Algorithms and Data Structures (CSci 115)

California State University Fresno
College of Science and Mathematics
Department of Computer Science
H. Cecotti

Learning Objectives

- Stack and queues
  - + associated functions
- When to use a stack, a queue?
- Where is it used?

Introduction

- Collection of data and the particular operations that are allowed on that data
  - list – add, delete, print
- Considered abstract because the operations you perform are separate from the underlying implementation
  - list – array or linked list
- Objects are perfectly suited
  - well-defined interface
  - implementation is hidden

Introduction

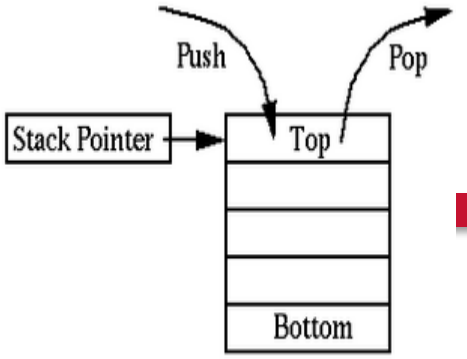
- Template in C++
  - Function template
  - Class template

```
// function template
#include <iostream>
using namespace std;

template <class T>
T GetMax(T a, T b) {
    T result;
    result = (a>b) ? a : b;
    return (result);
}

int main() {
    int i = 5, j = 6, k;
    long l = 10, m = 5, n;
    k = GetMax<int>(i, j);
    n = GetMax<long>(l, m);
    cout << k << endl;
    cout << n << endl;
    return 0;
}
```

```
// class templates
#include <iostream>
using namespace std;

template <class T>
class mypair {
    T a, b;
public:
    mypair(T first, T second)
    {
        a = first; b = second;
    }
    T getmax();
};

template <class T>
T mypair<T>::getmax()
{
    T retval;
    retval = a>b ? a : b;
    return retval;
}

int main() {
    mypair <int> myobject(100, 75);
    cout << myobject.getmax();
    return 0;
}
```

```
#include <iostream>
using std::cout;
using std::endl;

#include <iomanip>
using std::setw;

#include <typeinfo>

// define a clas array of type T
// the type is not know yet and will
// be defined by instantiation
// of object of class array<T> from main
template< typename T > class array {
private:
    int size;
    T *myarray;
public:
    // constructor with user pre-defined size
    array(int s) {
        size = s;
        myarray = new T[size];
    }
    // calss array member function to set element of myarray
    // with type T values
    void setArray(int elem, T val) {
        myarray[elem] = val;
    }

    // for loop to display all elements of an array
    void getArray() {
        for (int j = 0; j < size; j++) {
            // typeid will retriev a type for each value
            cout << setw(7) << j << setw(13) << myarray[j]
                << " type: " << typeid(myarray[j]).name() << endl;
        }
        cout << "----------------------------" << endl;
    }
};
```

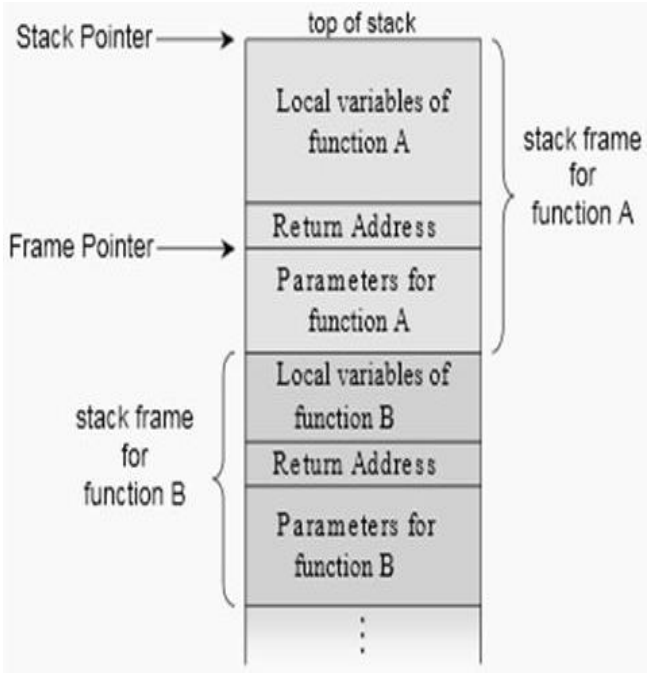
Stacks



A LIFO Stack

- Linear data structure
- Allows access to only one data item –
  - the last item inserted
- If you remove this item, you can access the next-to-last item inserted, and so on
- **Last-In-First-Out** (LIFO)
- Useful in many programming situations
  - to analyse arithmetic expressions
- It can also be used for algorithms applied to certain complex data structures,
  - traversing the nodes of a tree

Stacks- Real-world Examples



Thread Stack

Stack Terms

- Placing a data item on the top of the stack is called **pushing it**
  - **Push function**
- Removing a data item on the top of the stack is called **popping it**
  - **Pop**

Array representation of a Stack

Stack Class

- **`Stack`** class contains the following operations:
  - **`push`** – pushes an item onto the top of the stack
  - **`pop`** – removes an item from the top of the stack
  - **`peek`** – retrieves the top item of the stack without removing it
  - **`empty`**– returns true if the stack is empty
- The **`Stack`** class contains
  - methods corresponding to standard stack operations
  - a method to search for a particular object in the stack

Stack in C++

- STL (Standard Template Library)
  - empty()
    - Returns whether the stack is empty
  - size()
    - Returns the size of the stack
  - top()
    - Returns a reference to the top most element of the stack
  - push(g)
    - Adds the element ‘g’ at the top of the stack
  - pop()
    - Deletes the top most element of the stack

```cpp
#include <iostream>
#include <stack>

using namespace std;

void showstack(stack <int> gq)
{
    stack <int> g = gq;
    while (!g.empty())
    {
        cout << '\t' << g.top();
        g.pop();
    }
    cout << '\n';
}

int main()
{
    stack <int> gquiz;
    gquiz.push(10);
    gquiz.push(30);
    gquiz.push(20);
    gquiz.push(5);
    gquiz.push(1);

    cout << "The stack gquiz is : ";
    showstack(gquiz);

    cout << "\ngquiz.size() : " << gquiz.size();
    cout << "\ngquiz.top() : " << gquiz.top();


    cout << "\ngquiz.pop() : ";
    gquiz.pop();
    showstack(gquiz);

    return 0;
}
```

Stack Error Handling

- What happens if you try to push an item onto a stack that is already full?
- What happens if you try to pop an item from a stack that is empty?
- The responsibility for handling such errors is up to the class user. The user should always check to be sure that the stack is not full before inserting an item:

```
IF stack if full
    Output message
ELSE
    Add item to the stack
```

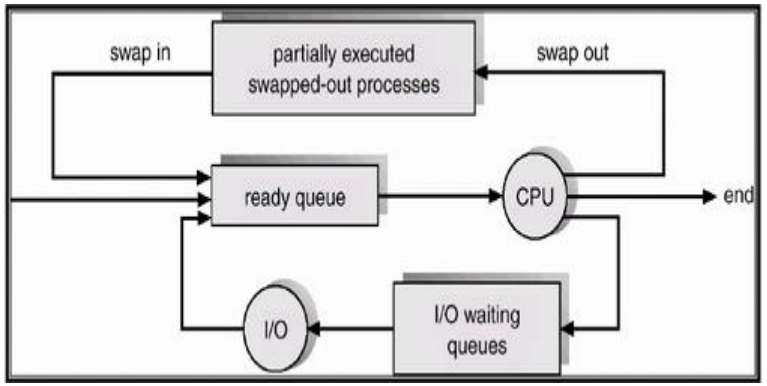
Stack Error Handling

- Many stack classes check for these errors internally
  - in the **`push()`** and **`pop()`** methods
- This is the preferred approach
  - A good solution for a stack class that discovers such errors is to throw an exception, which can then be caught and processed by the class user

Queues - Real-world Examples

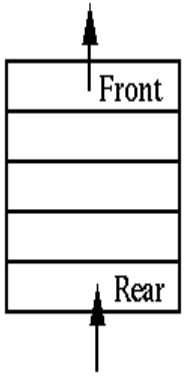
- Operating systems
  - Job queue
    - keeps all the processes in the system.
  - Ready queue
    - keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.
  - Device queues The processes which are blocked due to unavailability of an I/O device constitute this queue.

Operating System Queues



Queues

- Queues are used to model **real-world situations**
  - people waiting in line at a bank
  - airplanes waiting to take off
  - data packets waiting to be transmitted over the Internet



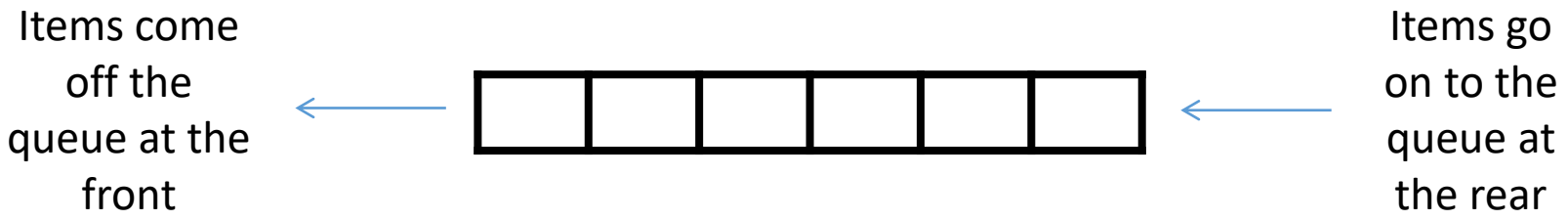
A FIFO queue

- There are various queues quietly doing their job in your computer's operating system:
  - a printer queue where print jobs wait for the printer to be available
  - a queue also stores keystrokes as you type at the keyboard

Queues

- Queues are linear data structures similar to lists

- Operate in a **First-In-First-Out (FIFO)** manner
  - The first person to join the line is the first person to reach the front of the line
  - The last person to line up is the last person to reach the front of the line



Queue Operations

- A queue data structure typically has the following operations:
  - **`insert`** – to add an item to the rear of the queue
    - sometimes referred to an **`enqueue`**
  - **`remove`** – to remove an item from the front of the queue
    - sometimes referred to an **`dequeue`**
  - **`empty`** – returns true if the queue is empty

Array representation of a Queue

Queues in C++

- STL (Standard Template Library)
  - empty()
    - Returns whether the queue is empty
  - size()
    - Returns the size of the queue
  - front()
    - Returns a reference to the first element of the queue
  - back()
    - Returns a reference to the last element of the queue
  - push(g)
    - Adds the element ‘g’ at the end of the queue
  - pop()
    - Deletes the first element of the queue

```
#include <iostream>
#include <queue>

using namespace std;

void showq(queue <int> gq)
{
    queue <int> g = gq;
    while (!g.empty())
    {
        cout << '\t' << g.front();
        g.pop();
    }
    cout << '\n';
}

int main()
{
    queue <int> gquiz;
    gquiz.push(10);
    gquiz.push(20);

    cout << "The queue gquiz is : ";
    showq(gquiz);

    cout << "\ngquiz.size() : " << gquiz.size();
    cout << "\ngquiz.front() : " << gquiz.front();
    cout << "\ngquiz.back() : " << gquiz.back();

    cout << "\ngquiz.pop() : ";
    gquiz.pop();
    showq(gquiz);

    return 0;
}
```

Priority Queue

- A priority queue
  - A more specialised data structure than a stack or a queue
- Like an ordinary queue, a priority queue has
  - a front
  - a rear
  - items are removed from the front
- In a priority queue, **however**,
  - Items are ordered by **key value** so that the item with the **lowest key** (or in some implementations the highest key) is **always at the front**
  - Items are inserted in the proper position to maintain the order – sorted list
  - The item with the lowest key has the highest priority

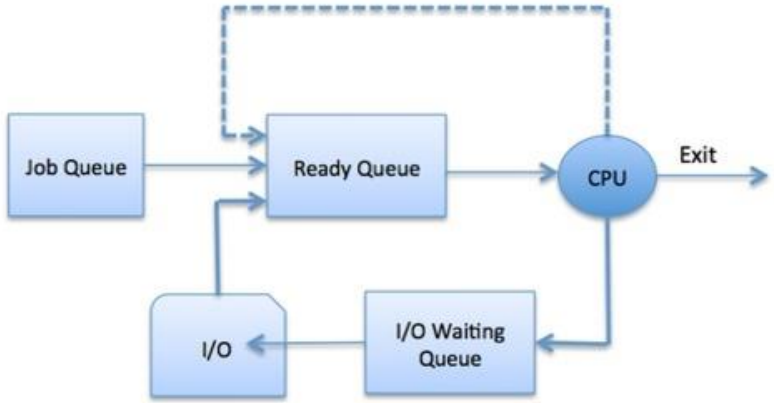
Priority Queue in C++

- STL (Standard Template Library)
  - empty()
    - Returns whether the queue is empty
  - size()
    - Returns the size of the queue
  - top()
    - Returns a reference to the top most element of the queue
  - push(g)
    - Adds the element ‘g’ at the end of the queue
  - pop()
    - Deletes the first element of the queue

```
#include <iostream>
#include <queue>

using namespace std;

void showpq(priority_queue <int> gq)
{
    priority_queue <int> g = gq;
    while (!g.empty())
    {
        cout << '\t' << g.top();
        g.pop();
    }
    cout << '\n';
}

int main()
{
    priority_queue <int> gquiz;
    gquiz.push(10);
    gquiz.push(30);
    gquiz.push(20);
    gquiz.push(5);
    gquiz.push(1);

    cout << "The priority queue gquiz is : ";
    showpq(gquiz);

    cout << "\ngquiz.size() : " << gquiz.size();
    cout << "\ngquiz.top() : " << gquiz.top();

    cout << "\ngquiz.pop() : ";
    gquiz.pop();
    showpq(gquiz);

    return 0;
}
```

Conclusion

- Stacks and Queues
  - Popular data structures to manage problems that are well suited for:
  - LIFO: Stacks
  - FIFO: Queues
- Application in Operating Systems (CSci144/CSci244)
  - Example: Process Scheduling



Questions ?

- Reading:
  - Chapter 10: Elementary Data Structures
    - Introduction to Algorithms 3rd Ed.