

Intro to Data Structs: Queues and Stacks

Wagner Truppel
Lecturer, Dept. of Computer
Science & Engineering
UC Riverside

wagner@cs.ucr.edu
<http://www.cs.ucr.edu/~wagner>

<http://www.cs.ucr.edu/cs12>

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 4 1

Today's Topics

- Brief review of last lecture
- Brief exploration of **Queues**
- Not-so-brief exploration of **Stacks**
- More on **memory layout**
- **Activation Frames**
- Prelude to **Recursion**

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 4 2

Brief review of Data Structures

- Data Structures
 - ◆ Are "containers" for the data that your program manipulates
 - ◆ Allow you to group together pieces of data that are logically related
 - ◆ Have standard operations defined on them
 - ◆ May be implemented in different ways
 - ◆ Different implementations may give you different speeds for the standard operations
 - ◆ You'll learn more in CS 14 and CS 141

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 4 3

Arrays

- The simplest example of a data structure
- Fast **random access**
- **Insertion and Deletion**
 - ◆ slow: $O(n)$
 - ◆ Examples: library books, movie line
- **Search**
 - ◆ slow if not sorted [**linear search**, $O(n)$]
 - ◆ fast if sorted [**binary search**, $O(\log_2 n)$]
- Incidentally...
 - ◆ Sorting n items almost always takes at least $O(n \log_2 n)$ steps

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 4 4

Queues

- Another simple data structure
- Also called a **FIFO** data structure
 - ◆ First-In, First-Out
- Examples
 - ◆ bank line
 - ◆ printer job scheduling
 - ◆ process scheduling
- Operations
 - ◆ **Enqueue(x)**
 - ◆ **Dequeue()**
- Operations affect **both** ends, and **only** the ends
- Queues are dynamic data structures

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 4 5

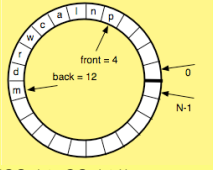
Queues

- Another simple data structure
- Also called a **FIFO** data structure
- Examples: bank line, process scheduling
- Operations: **Enqueue(x)** and **Dequeue()**
- Dumb implementation: simple array
- Why dumb?

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 4 6

Queues

- Better implementation: *circular array*
 - ◆ Two auxiliary ints: *front* and *back*
 - ◆ Update *front* and *back* as needed
 - ◆ Must avoid collisions
- There are other, even better, implementations (CS 14, CS 141)
- enqueue() and dequeue() are a little complicated... won't show you here



©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 4 7

Stacks

- Yet another relatively simple data structure
- Also called a *LIFO* data structure
 - ◆ Last-In, First-Out
- Examples
 - ◆ cafeteria trays
 - ◆ getting dressed/undressed
 - ◆ function calls
 - ◆ recursion
- Operations: **Push(x)** and **Pop()**
- Operations affect **only one end**
- Push(x) and Pop() are fast
- Insertion and Deletion are slow
- Search is slow
- Stacks are also dynamic data structures

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 4 8

Stacks

- Several implementations (CS 14, CS 141)
- Simplest implementation uses a static array to support the stack
- Need:
 - ◆ An array (duh!): *stackA*
 - ◆ An integer: *int top*

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 4 9

Stacks: example

```

...
push ('A');
push ('B');
push ('D');
pop ();
push ('C');
push ('D');
push ('E');
pop ();
pop ();
...
                
```

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 4 10

Stack Operations

```

bool isEmpty()
{ return (top == -1); }

bool isFull()
{ return (top == (ARRAY_SIZE - 1)); }

void push(type x)
{ stackA[++top] = x; }

type pop()
{ return stackA[top--]; }
                
```

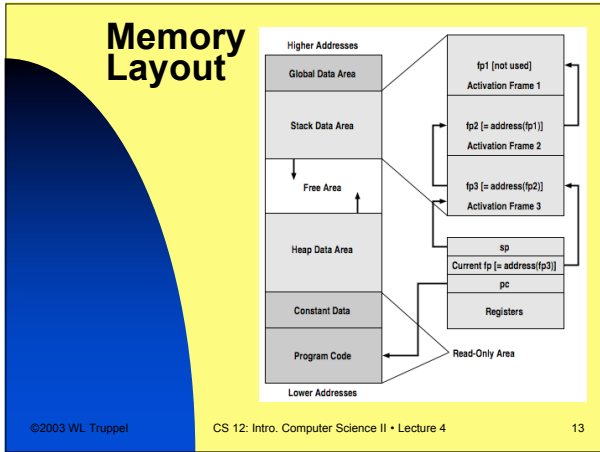
▪ Here, **type** is an arbitrary type, such as *int*, *char*, *float*, or whatever type your stack is supposed to store

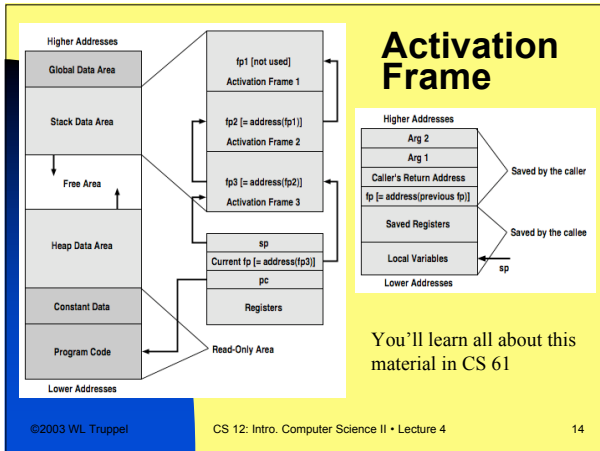
©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 4 11

More memory layout

- Typically, the computer's OS allocates 5 kinds of memory regions for your program
 - ◆ **Read-only area:**
 - * code and constants
 - ◆ **Global/static area:**
 - * global variables
 - ◆ **Stack/local area:**
 - * grows/shrinks as needed when the program enters/exits code blocks and fns
 - ◆ **Heap/dynamic area:**
 - * used to store dynamic arrays and objects
 - ◆ **Registers:** used for fast access and for some specialized activities

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 4 12





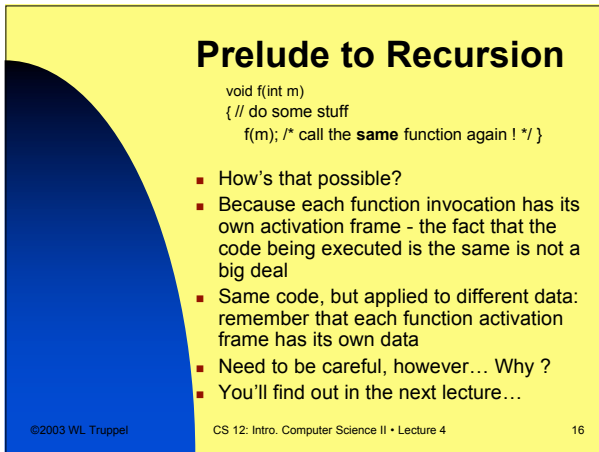
Prelude to Recursion

- You're used to a function calling another function:


```
void f(int m)
{ // do some stuff
  g(m); /* call another function */ }
```
- Can a function call itself ?


```
void f(int m)
{ // do some stuff
  f(m); /* call the same function again ! */ }
```
- Yes! That's called a **recursive call** and the function is called a **recursive function**.

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 4 15



Prelude to Recursion

```
void f(int m)
{ // do some stuff
  f(m); /* call the same function again ! */ }
```

- How's that possible?
- Because each function invocation has its own activation frame - the fact that the code being executed is the same is not a big deal
- Same code, but applied to different data: remember that each function activation frame has its own data
- Need to be careful, however... Why ?
- You'll find out in the next lecture...

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 4 16
