Stack
Topics

1) How does the computer actually handle..
   a) Calling a function?
   b) Passing arguments?
   c) Returning a value?
#include <iostream>
using namespace std;

int foo(int a, char b, float c) {
    int ans = a + b + c;
    return ans;
}

int main() {
    int x = 1;
    char y = 'A';
    x = foo(x, y, 3.14);
    return 0;
}

• How does this program:
  − Pass arguments to foo()?
  − Pass the return value back to main() from foo()?
  − Allocate local variables?

• Answer:..
Basics

- Computer's main memory is RAM:
  - Able to access any byte in memory..
- Each byte in memory has an address
- Each running program is given memory for:
  - Storing code (instructions)
    Code usually loaded by OS from disk.
  - Storing data (variables)
    Variables are..
Simple view of Memory

- Imagine memory as a very long row of bytes.

<table>
<thead>
<tr>
<th>Address</th>
<th>2047</th>
<th>2048</th>
<th>2049</th>
<th>2050</th>
<th>2051</th>
<th>2052</th>
<th>2053</th>
<th>2054</th>
<th>2055</th>
<th>2056</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![alpha]

- Declaring a variable..
  - Simplified idea: Variables declared sequentially in memory.
  - Find size of beta:..
  - Find location of beta:..

```c
int main() {
    char alpha = 'a';  // 1 byte
    int beta = 2;      // 4 bytes
    float gamma = 3.0; // 4 bytes
}
```

& is the operator
Stack Memory

- Memory can be view as a stack: Start at the..
- New variables allocated on top..
- Remove destroyed variables from top..

```
int main() {
    char alpha = 'a'; // 1 byte
    int beta = 2; // 4 bytes
    float gamma = 3.0; // 4 bytes
}
```
Function Calls

• Calling a function allocates a stack frame for the function:

```c
int foo(int a, char b, float c) {
    int ans = a + b + c;
    return ans;
}
```

```c
int main() {
    int x = 1;
    char y = 'A';
    x = foo(x, y, 3.14);
    ...
```
Function Execution

- Argument values..
- Function does work.
- Return value..
- Return value handled by calling code.

```c
int foo(int a, char b, float c) {
    int ans = a + b + c;
    return ans;
}

int main() {
    int x = 1;
    char y = 'A';
    x = foo(x, y, 3.14);
    ...
}
Function Completion

• When foo() finishes, ..

• Memory reused by the next function call.

```c
int foo(int a, char b, float c) {
    int ans = a + b + c;
    return ans;
}
```

```c
int main() {
    int x = 1;
    char y = 'A';
    x = foo(x, y, 3.14);
    ...
}
```
Stack Growth and Reuse

• Stack grows when one function calls another
  – Once on the stack, a variable is a fixed size.
  – (Trying to grow its size would grow the whole stack!)

• If main() calls foo() then bar(); bar();
  foo()'s stack frame has been popped
  bar()'s stack frame starts at same location
Review

- What is found in a function's stack frame?

- Explain the terms push and pop

- What will happen when executing `bar()`?
  ```c
  int bar() {
    return bar() + 1;
  }
  ```
Summary

- Stack used to store Stack Frames:
  - arguments, return value, and local variables.
- Entering a function pushes a stack frame, leaving pops it.
  - Stack space reused for next function call.