Today's Plan

Upcoming:

- A1 posted!
- Labs next week

Last time:

- C++ Review

Today's topics:

- From last time:
  - Function Calls: Pass by Value/Reference
- C++ Demos
- Stacks
- How Function Calls Work with the Stack

Demo: Counting Long Lines

Let's look at a program `line_check.cpp` that counts and prints the number of lines in a text file that have more than 100 characters.

Demo: Counting Words

wc is the Linux "word count" utility:

```
> wc austenPride.txt
 13427 124580 704158 austenPride.txt
```

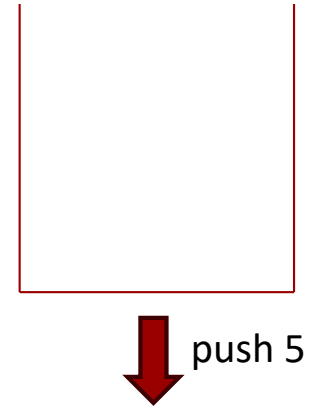
Let's look at a possible implementation for this.

How function calls work

- Knowing how function calls work in C++ is helpful
- It lets you better understand program flow, memory management, and local/global variables
- It also helps in understanding recursive functions

How function calls work: Stacks

an empty stack



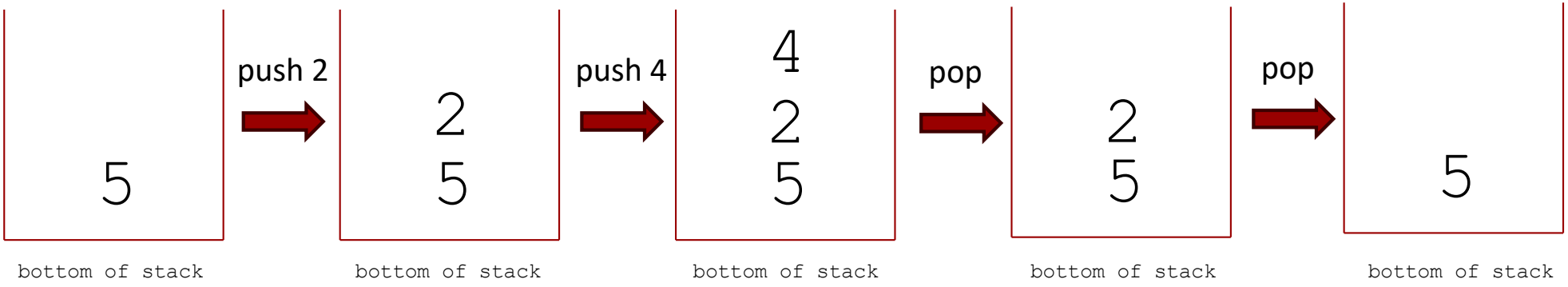
push 5

A **stack** is a simple data structure that supports these operations:

- **is_empty()** returns true if the stack is empty, and false otherwise
- **push x** puts **x** on the top of the stack
- **pop** removes the top element of the stack
- **peek** returns a copy of the top element of the stack.

pushing onto a full stack causes an error: **stack overflow**

popping/peeking an empty stack causes an error (**stack underflow**)

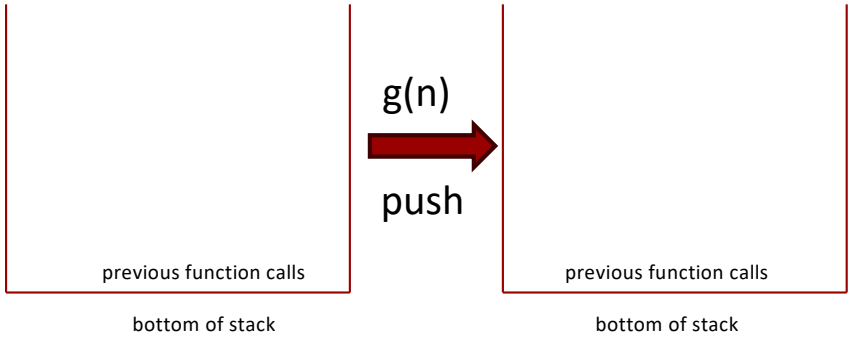


How function calls work

- For a running C++ program, C++ designates part of memory as the programs **call stack**
- **Every time a function is called**, the function, its parameters, and return address are **pushed** onto the call stack
  - Everything that gets pushed is refer to as a **stack frame**
  - Local variables are also stored in the functions stack frame
- **Every time a function exits**, the stack frame on top of the stack is **popped** and the return-value of the function call is put there (conceptually)
  - Local variables are deleted by this pop

```
int g(int a) {
    int n = a + 2;
    return 3*n;
}
```

```
int n = 5;
g(n); // push onto call stack
cout << "done!";
```

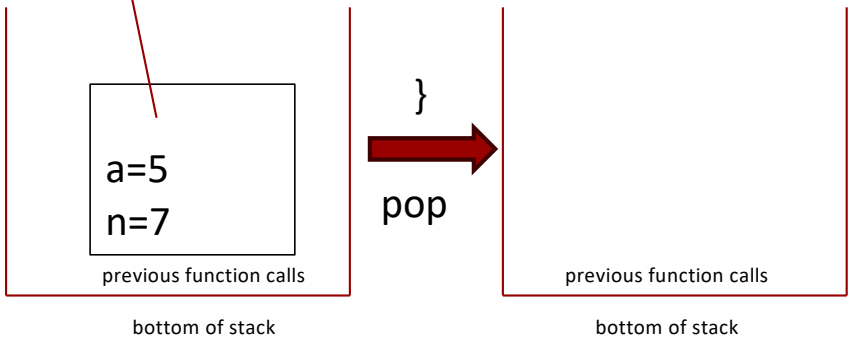


How function calls work

- For a running C++ program, C++ designates part of memory as the programs **call stack**
- **Every time a function is called**, the function, its parameters, and return address are **pushed** onto the call stack
  - Everything that gets pushed is refer to as a **stack frame**
  - Local variables are also stored in the functions stack frame
- **Every time a function exits**, the stack frame on top of the stack is **popped** and the return-value of the function call is put there (conceptually)
  - Local variables are deleted by this pop

```
int g(int a) {
    int n = a + 2;
    return 3*n;
}
```

```
int n = 5;
g(n); // push onto call stack
cout << "done!";
```

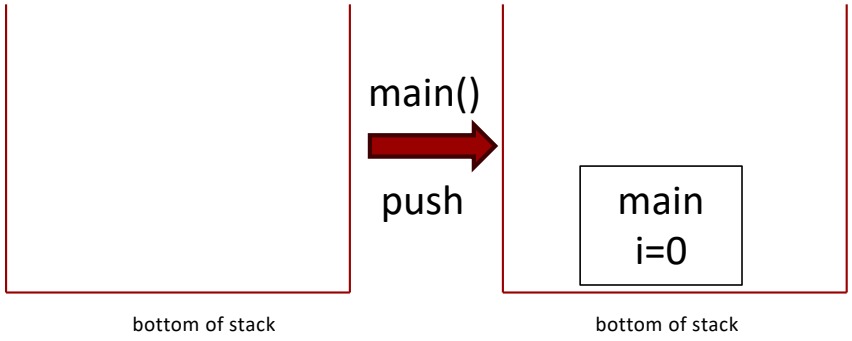


How function calls work

- For a running C++ program, C++ designates part of memory as the programs **call stack**
- **Every time a function is called**, the function, its parameters, and return address are **pushed** onto the call stack
  - Everything that gets pushed is refer to as a **stack frame**
  - Local variables are also stored in the functions stack frame
- **Every time a function exits**, the stack frame on top of the stack is **popped** and the return-value of the function call is put there (conceptually)
  - Local variables are deleted by this pop

```
int f() {
   int i = 7;
   i++;
   return i;
}
```

```
int main() {
   int i = 0;
   cout << f();
}
```

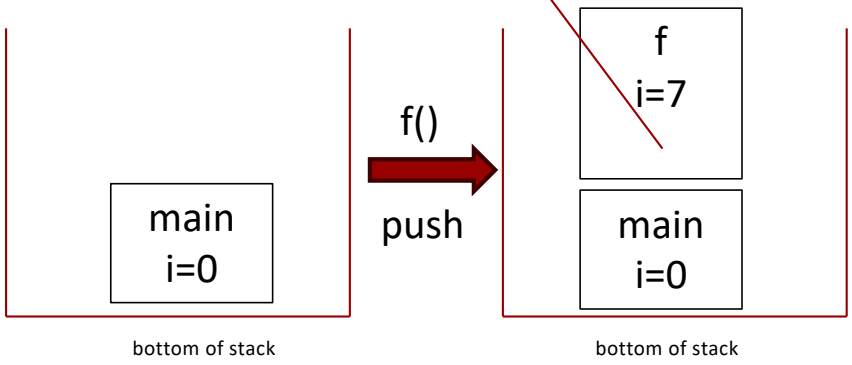


How function calls work

- For a running C++ program, C++ designates part of memory as the programs **call stack**
- **Every time a function is called**, the function, its parameters, and return address are **pushed** onto the call stack
  - Everything that gets pushed is refer to as a **stack frame**
  - Local variables are also stored in the functions stack frame
- **Every time a function exits**, the stack frame on top of the stack is **popped** and the return-value of the function call is put there (conceptually)
  - Local variables are deleted by this pop

```
int f() {
   int i = 7;
   i++;
   return i;
}
```

```
int main() {
   int i = 0;
   cout << f();
}
```

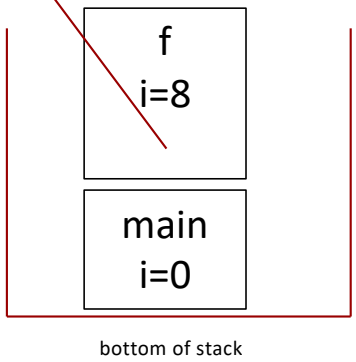


How function calls work

- For a running C++ program, C++ designates part of memory as the programs **call stack**
- **Every time a function is called**, the function, its parameters, and return address are **pushed** onto the call stack
  - Everything that gets pushed is refer to as a **stack frame**
  - Local variables are also stored in the functions stack frame
- **Every time a function exits**, the stack frame on top of the stack is **popped** and the return-value of the function call is put there (conceptually)
  - Local variables are deleted by this pop

```
int f() {
   int i = 7;
   i++;
   return i;
}
```

```
int main() {
   int i = 0;
   cout << f();
}
```

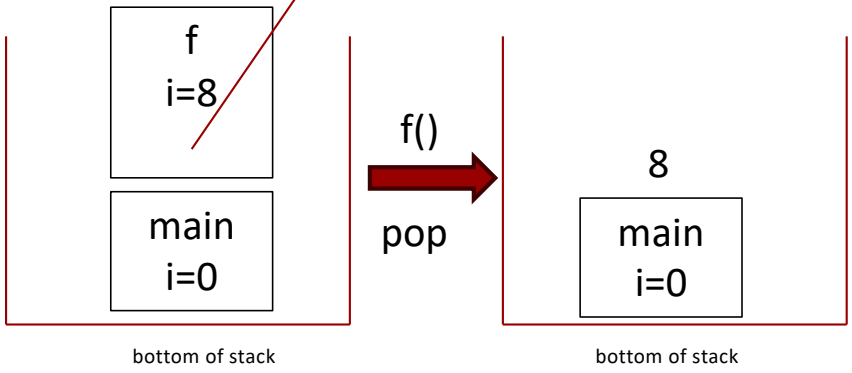


How function calls work

- For a running C++ program, C++ designates part of memory as the programs **call stack**
- **Every time a function is called**, the function, its parameters, and return address are **pushed** onto the call stack
  - Everything that gets pushed is refer to as a **stack frame**
  - Local variables are also stored in the functions stack frame
- **Every time a function exits**, the stack frame on top of the stack is **popped** and the return-value of the function call is put there (conceptually)
  - Local variables are deleted by this pop

```
int f() {
   int i = 7;
   i++;
   return i;
}
```

```
int main() {
   int i = 0;
   cout << f();
}
```

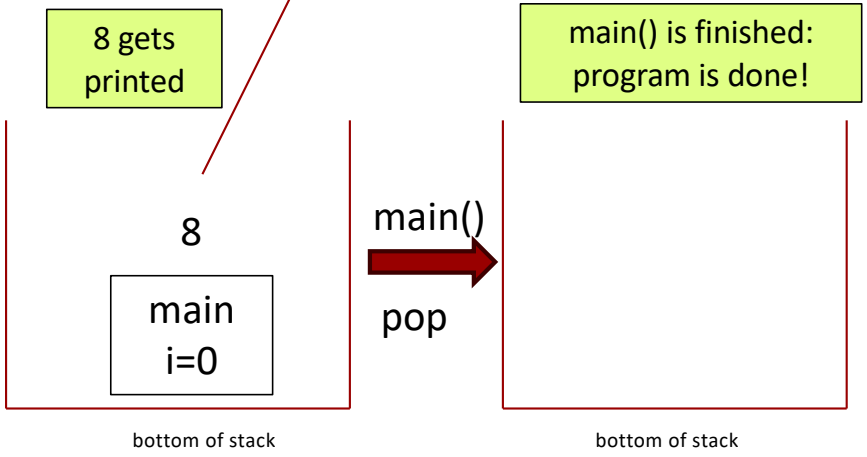


How function calls work

- For a running C++ program, C++ designates part of memory as the programs **call stack**
- **Every time a function is called**, the function, its parameters, and return address are **pushed** onto the call stack
  - Everything that gets pushed is refer to as a **stack frame**
  - Local variables are also stored in the functions stack frame
- **Every time a function exits**, the stack frame on top of the stack is **popped** and the return-value of the function call is put there (conceptually)
  - Local variables are deleted by this pop

```
int f() {
   int i = 7;
   i++;
   return i;
}
```

```
int main() {
   int i = 0;
   cout << f();
}
```



How function calls work

- Stacks are easy to implement efficiently
  - use an array and an index variable to keep track of the top of the stack
- Function calls are generally quite efficient
  - C++ compilers can use **function inlining to** replace short functions calls with the contents of their body, thus avoiding the overhead of pushing/popping the call stack
- It's possible to call so many functions without returning that you get a **stack overflow** error
  - Your program crashes because it's out of memory
  - This is a significant issue in systems with limited memory, e.g. embedded systems
- Works with **recursive functions**, e.g. functions that call themselves