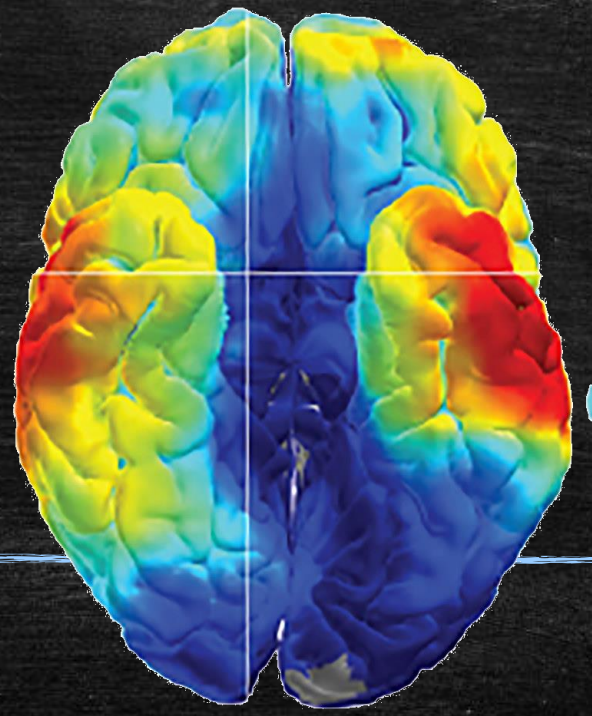
The Shell & Dynamic Memory



CS 150 – C++ Programming I
Lecture 23

Command-line Arguments

- Arguments passed from the shell to the `main()` function
  - Always supplied as an array of C-strings
- To collect them, use one of these versions of `main()`

```
int main(int argc, char * argv[])
int main(int argc, char ** argv)
```

  - *argc*: number of strings on command line
    - including the executing program name
  - *argv* is an array of C-strings *char**

Processing the Command-line

- You can use `argc` to check the number of arguments passed
  - You can use `argv[0]` and `cerr` to print a "usage" message

```
if (argc == 1)   // NO argument passed
{
    cerr << "usage: "
         << argv[0] << " arg" << endl;
    return EXIT_FAILURE;
}
```

  - Use `exit()` or `return` if no argument is passed

Processing All Arguments

- You can process all arguments by using a loop

```
cout << argv[0] << endl;
for (int i = 1; i < argc; i++)
    cout << "#" << i << "->"
         << argv[i] << endl;
```

- argc doesn't count redirection parameters

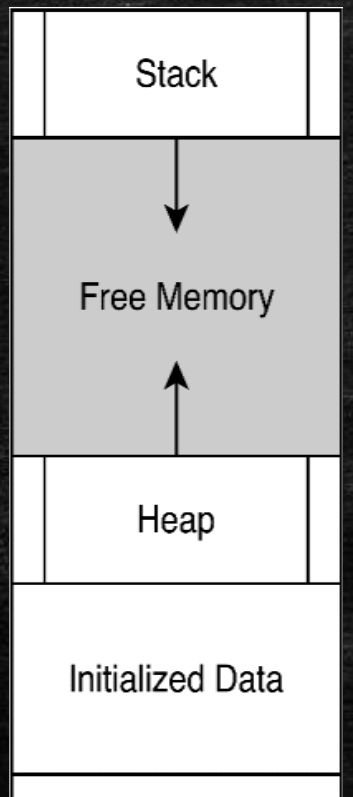
```
./ex                    // argc == 1
./ex arg1 arg2          // argc == 3
./ex arg1 > out.txt     // argc == 2
```

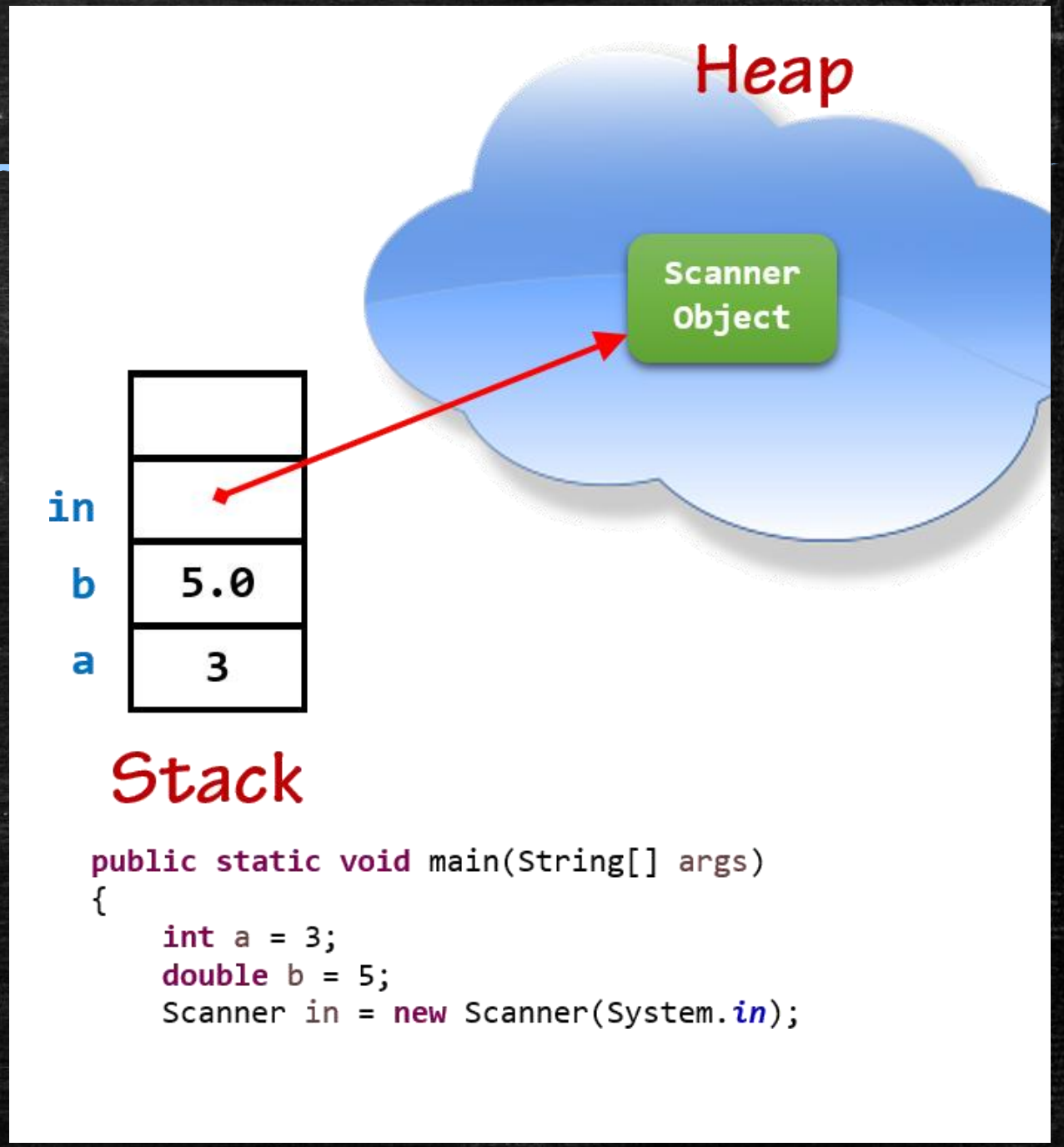
Command Lines-Try it Yourself

- Write the program *frp* (*file replace*)
  - The purpose of the program is to open a file and print it, replacing every instance a given word with another
- Command line arguments
  - The name of the file
  - The word to search for
  - The word to use as the replacement (optional)
  - See error messages and sample dialog in the handout
- Test your program with *make test*

Meet the Heap

- Like Java and Pascal, C++ places variables on the heap using the ***new*** operator






```
public static void main(String[] args)
{
    int a = 3;
    double b = 5;
    Scanner in = new Scanner(System.in);
```

Dynamic Memory Overview

- Variables are allocated on the heap using new
  - The result is an address, stored in a pointer
  - `int *p1 = new int{3};  // initialized int`
  - `int *ia = new int[3](); // array`
- Unlike Java, C++ has manual memory management
  - You are responsible for returning the memory to the system
  - `delete p1;  // single object`
  - `delete[] ia;  // array on heap`

Introducing Dynamic Variables

- Exercise: request some memory on the heap
  - Allocate memory on the heap with the new operator
- Store the result in a pointer, not a regular variable
  - `int *p1 = new int;  //uninitialized`
  - `int *p2 = new int{};  // default initialized`
  - `int *p3 = new int(3);  // direct initialized`
  - `int *p4 = new int[3];  //uninitialized array`
  - `int *p5 = new int[3]{};  //default init`
  - `int *p6 = new int[3]{1,2,3}; //initialized`
  - `Employee *p7 = new Employee{"Bob", 200};`

Initializing and Freeing Memory

- Notice that we don't see any errors! Is everything OK?
  - NO. Unlike Java/C#, you must return any memory allocated on the heap to the operating system
  - Do `make grind` and then `make check` to see errors
- Use `delete` or `delete[]` to return memory
  - `delete` the pointer to free the heap object
- Three pitfalls
  - 1. Only delete pointers allocated by `new`
  - 2. Never delete a pointer twice
  - 3. Never access an element after it is freed