C Lab 1

INTRODUCTION TO C

Basics of C

C was developed by Dennis Ritchie working at AT&T Bell Labs in the 1970’s.

C maps very easily to machine instructions (even allows inline assembly!)

Unlike Java or Python, the programmer is in charge of memory management. There is no garbage collector.

C is not Object Oriented (no inheritance or interfaces)

Terminology

Compiler—program that converts source code into object code (assembly/machine code)

- You will be using GCC to compile your source before you can run it

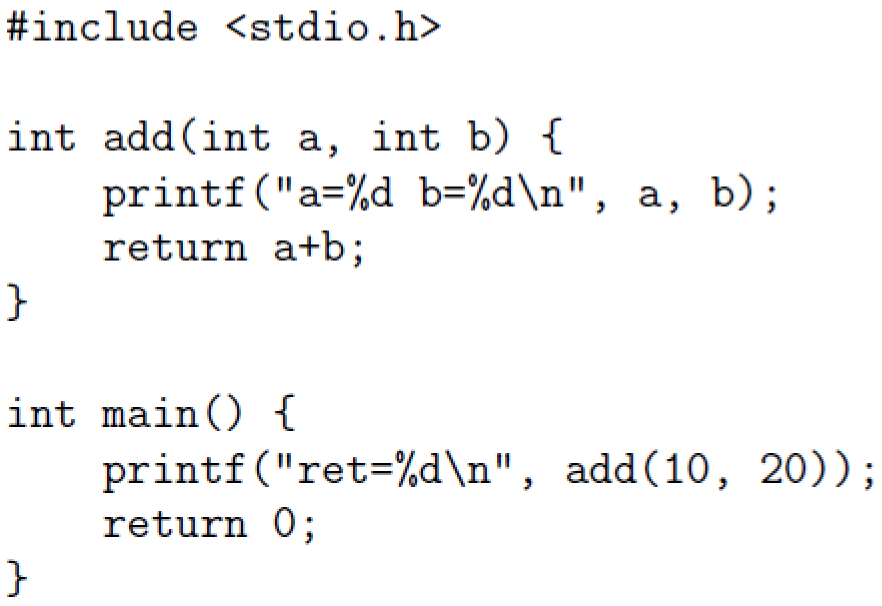
Debugger—Program that allows you to inspect a program while it is running.

- You will be using gdb
- A good alternative to print statements everywhere!

Header—A file of function and variable declarations

- If you want to use functions defined in other files, you must include a corresponding header (.h) file

Structure of a C Program



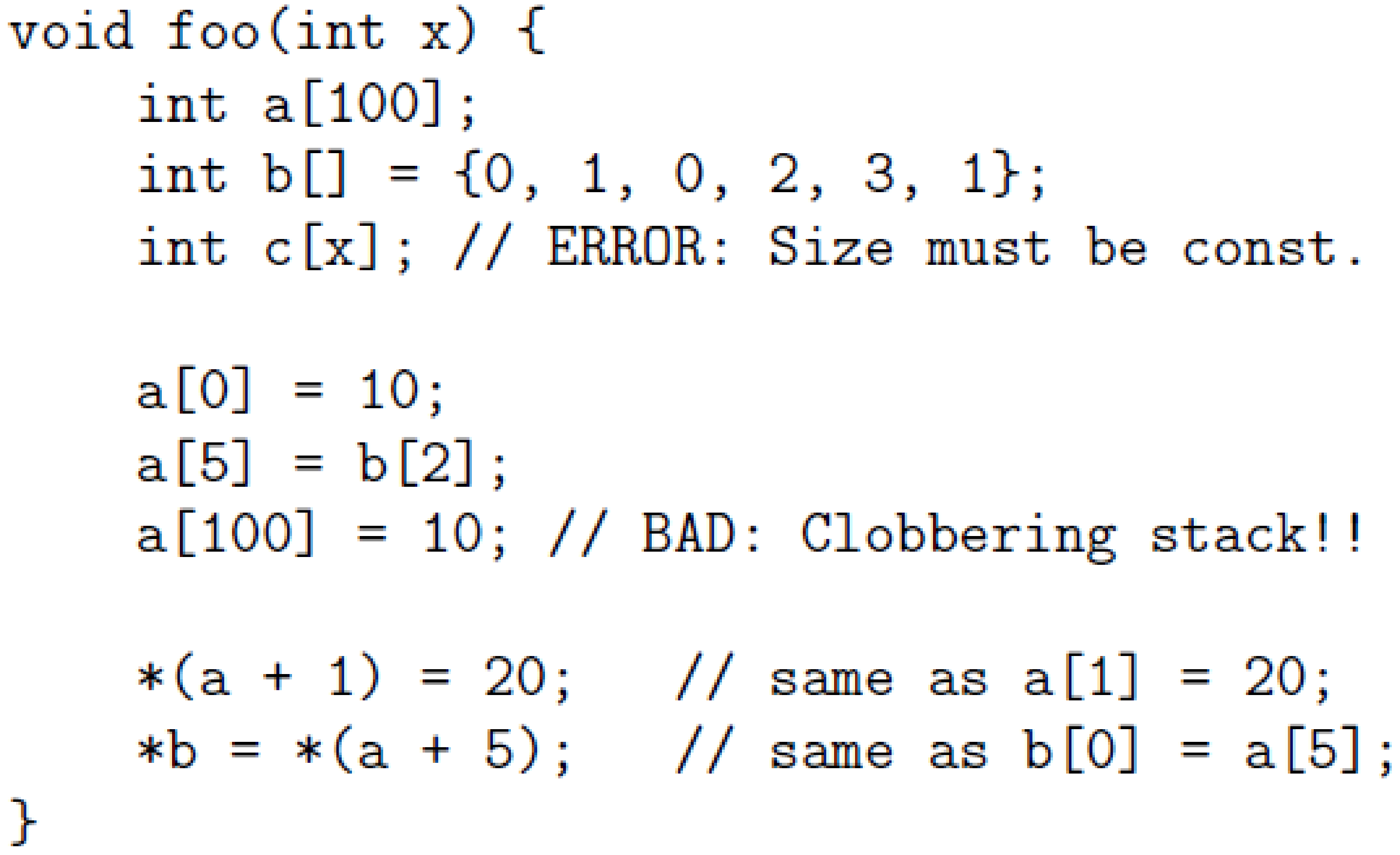
```c
#include <stdio.h>

int add(int a, int b) {
    printf("a=%d b=%d\n", a, b);
    return a+b;
}

int main() {
    printf("ret=%d\n", add(10, 20));
    return 0;
}
```

Which lines are preprocessor directives? Function declarations?

Basics: Arrays



```
void foo(int x) {
    int a[100];
    int b[] = {0, 1, 0, 2, 3, 1};
    int c[x]; // ERROR: Size must be const.

    a[0] = 10;
    a[5] = b[2];
    a[100] = 10; // BAD: Clobbering stack!!

    *(a + 1) = 20;   // same as a[1] = 20;
    *b = *(a + 5);   // same as b[0] = a[5];
}
```

Common Array Problems

C has **no** array-bound checks. You won’t even get a warning! At best you’ll get a segfault when you try to run it.

Do not use sizeof(array) or any pointer in general. It will return the size of the pointer, not the underlying memory size.

SEGFAULT

Strings in C…
are just null terminated char arrays

<string.h> has common string functions.

For example: “CS3410” is equivalent to:

Char str[] = {‘C’, ‘S’, ‘3’, ‘4’, ‘1’, ‘0’, ‘\0’}

Things to note:

Strlen(s) does not include the terminal character. Be careful when using memory operations (ie. Memcpy) which does include the character!

What are Pointers?

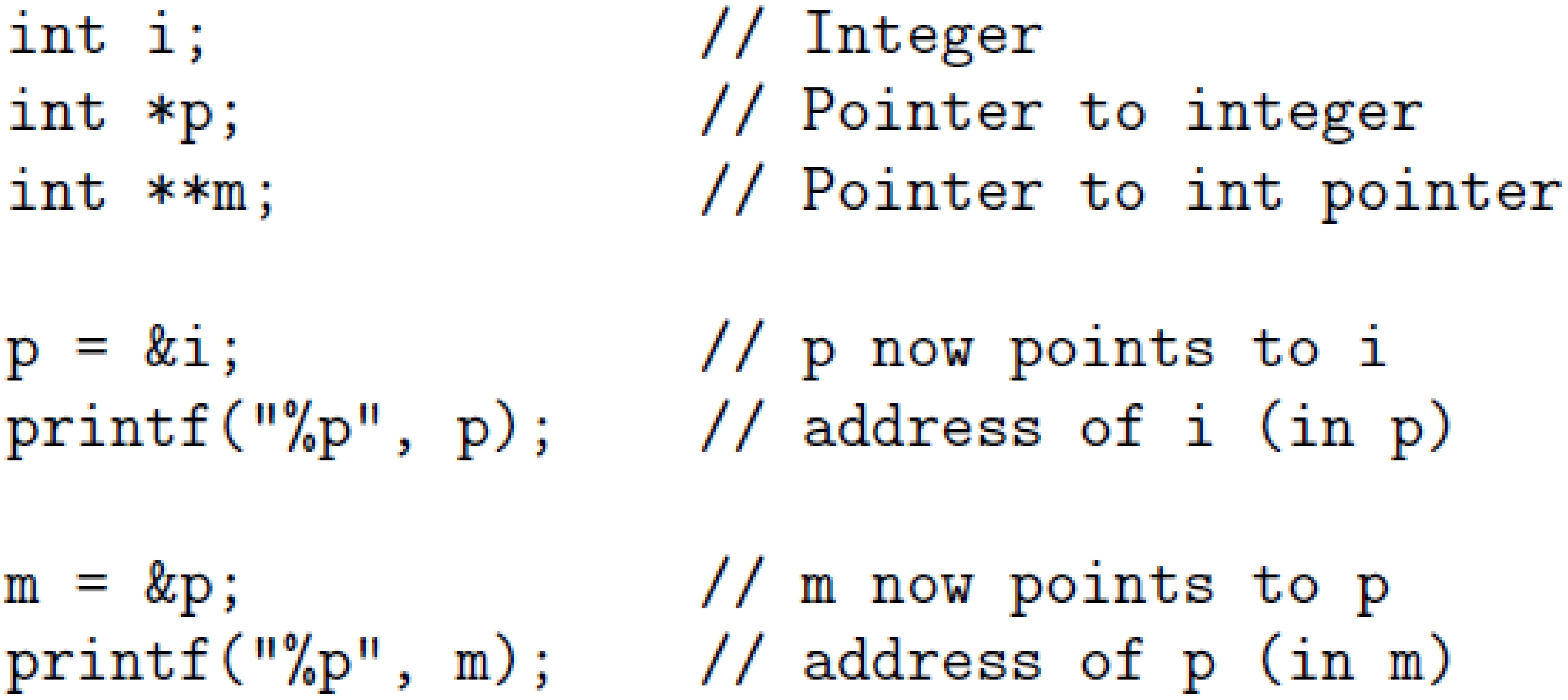
A pointer is an integer that represents a memory location either on the stack or on the heap.

The type of a pointer tells the compiler what kind of object to expect when it is dereferenced.

For example, a “double*” is an integer representing the memory location of a double.

A pointer does not actually create the data it points to. For the pointer to contain data, some other function must create the data it will point to. This is typically a call to malloc.

Pointers



```
int i;                  // Integer
int *p;                 // Pointer to integer
int **m;                // Pointer to int pointer

p = &i;                 // p now points to i
printf("%p", p);        // address of i (in p)

m = &p;                 // m now points to p
printf("%p", m);        // address of p (in m)
```

Heap

The heap persists across function calls. If you wish to return something that is not a primitive from a function, must use the heap.

In C, this is managed by the programmer through calls to (defined in stdlib.h):

- Malloc(size)
- Free(ptr)

You should always check the result of malloc since it can fail (you ran out of memory)

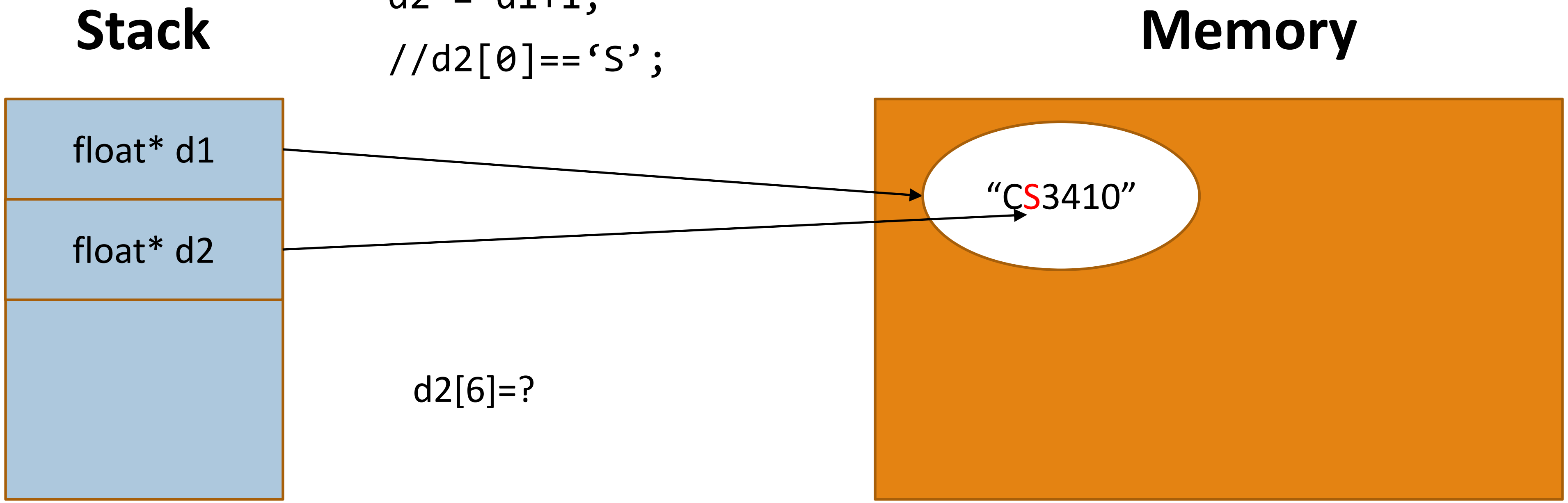
You **must** explicitly call free when the variable is no longer in use.

See cplusplus.com or your C reference manual for detailed information about library functions.

http://www.cplusplus.com/reference/cstdlib/malloc/

Pointers: A visual representation

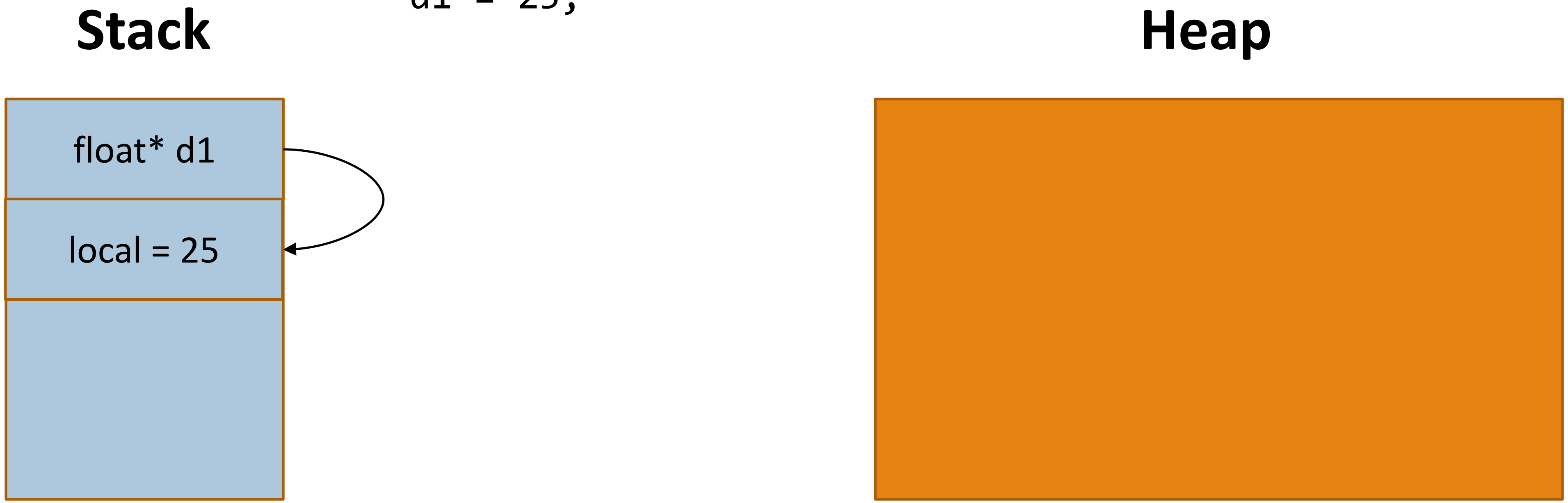
```
char* d1;
d1 = “CS3410”;
char* d2;
d2 = d1+1;
//d2[0]==‘S’;
```



d2[6]=?

Pointers to stack variables

```
float* d1;
float local = 42.42;
d1 = &local;
*d1 = 25;
```


Stack
float* d1
local = 25
Heap