CS240: Programming in C

Lecture 7: Multiple Arrays. Pointers and Arrays.

Multiple arrays

- int m[2][3]; /* 2 rows 3 cols*/
- int k;

- m [i][j] = k;
- m[i][j] = 2;

- int m[2][3] = {{1,1,1}, {1,1,1}};

Multiple arrays

- What if I want to store an array of names?
- char name[3][12];
- Let' s say that the names are "John", 'Dan' and 'Christopher'
- Can I store this in a more efficient way?

String and arrays

What does this mean?

char *s[3] = {”John", ”Dan", ”Christopher"};

It declares an array of pointers to char, and initializes each pointer with the address of the three constant strings

Pointers: reminder

- char c; is a declaration of a character
- char *ptr; is a declaration of an address that **points** to a character

ptr = &c;

&c means the address of c

*p = ‘m’ ;

*p means what is located at the address specified by p

Name of an array vs pointer

char a[10];

a is by convention also &a[0]

char *p;

Name of an array is not a variable

p = a; ALLOWED

p++;   ALLOWED

HOWEVER, it is not allowed

a = p;

a++;

int array[10];

int *p = &array[0];

p+i means the ith element in the array **regardless of the type stored** by the array

Let’s go back to the previous example

char *p = “John”;

Assigns the address of the string “John” and assigns to p the address of the constant string “John” . No string copy involved

char a[] = “John”;

This allocates the space for a to hold 5 characters (includes the ‘\0’).

Pointers of different types

```
#include <stdio.h>

int main() {
  int  *p_int = NULL;
  char *p_char = NULL;
  char c;

  p_char = &c;

  p_int = p_char; /* generates a warning */

  p_int  = (int*) p_char;

  return 0;
}
```

Pointers, operators, precedence

*++p;

++ applies before * , first the pointer is incremented, then dereferenced

Passing a multi-dimensional array to a function

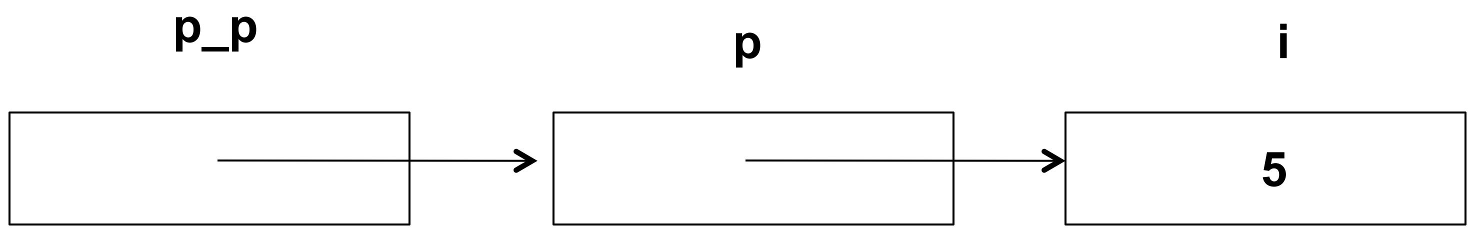
If a two-dimensional array is passed, the number of columns also needs to be passed, number of rows is irrelevant

```
int my_function(int matrix[12][31]);

int my_function(int matrix[][31]);
```

Pointer to pointer

```
int i = 5;
int *p = &i;
int **p_p = &p;
```



Think about it as *p_p is an int*, that is, p_p is a pointer to pointer to int

Pointer to pointer and arrays

char *s[3] = {”John", ”Dan",
  ”Christopher"};

s is a char **

char **p = s;

Passing arguments to programs

```
% cat file.txt
% ls -l
```

- These commands are frequently implemented as C programs
- Something like "-l" is usually called an option, which is still a command line argument
- How are the arguments passed to your C program?

Command line arguments

- A full prototype of the main function is:

```
int main(int argc, char **argv);
```

  - **`argc`** - number of command line arguments, including the program name
  - **`argv`** - an array of the arguments, each of which is a string (i.e., array of chars)
  - **`argc argv[0] argv[1] argv[2]`**

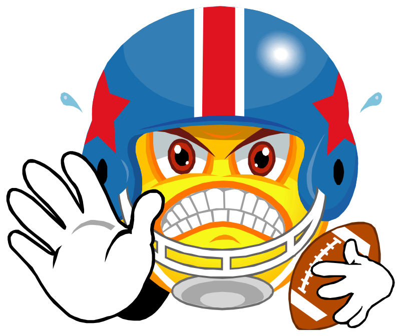
argv

- argv is char **
- First elements in argv is the name of the program.

```
for (i=0; i < argc; i++) {
  char *p = *(argv+i);
  printf("Argument %d : %s\n", i, p);
}
```

Exercise

- Write a small program where you free something twice and observe the behavior
- Write a small program where you don’t free the allocated memory and observe the behavior



Readings for This Lecture

K&R Chapter 5, up to 5.10