Lecture 10: More on Pointers

Pointer Variable : A variable whose only allowable value is an address

How to declare and use pointer variables:

```
int *y;     /* y is a pointer to an integer*/
int i=5;      /*  i is an integer */
y = &i;
*y = 7;
printf("%d", i);
```

Summary:

`&i` = address of variable i

`*y` = value stored in the location pointed to by y

```
int i=4, j=5;
float x =2.3;
int *intptr;
float *floatptr;
intptr = &i;
floatptr =&x;
```

Some common errors:

```
&i = intptr;
```

```
intptr = floatptr;
```

```
j = *i;
```

Pointer Arithmetic

What does the following C code do?

```
int a[100];
int *y;

y=&a[0];

for(i=0; i<100; i++)
    {
     *y = i;
      y = y + 1;
    }
```

Why pointers are important

Reason 1. They allow functions to "share" variables

Example 1: `swap()` function from last time

Example 2: `printf()` *v/s* `scanf()`

```
int i;
printf("input a number");
scanf("%d", &i);
printf("%d", i);
```

Reason 2: Pointers allow programs to adapt to varying data size at run-time.

The nonadaptive program:

```
int i, numbers[10];
printf("Enter 10 integers");
for (i=0; i<10; i++)
   scanf("%d", &numbers[i]);
```

The adaptive program:

```
int i, n, *numbers;
printf("How many integers will you input?");
scanf("%d", &n);
numbers = calloc(n, sizeof(int));

for (i=0; i<n; i++)
    scanf("%d", numbers + i);
```

Reason 3: They allow us to define dynamic data structures (lists, trees, queues, stacks..)

Will do next time