Basics of Pointers and Arrays

Key Points (excuse the pun)

1) How memory allocation works

2) How to create pointers to variables in memory in c

3) How to differentiate between an address in memory for a variable and its value (referencing / dereferencing)

4) How to create arrays and why arrays are really just pointers

Finding memory addresses in c

&x -> memory address of variable x (this is called referencing a variable)

```c
#include<stdio.h>
int main()
{
  int x=0;
  printf("value of x = %d,", x);
  // following code computes the address of x through &x, and recasts this as a typeless "pointer" so c can print (void *)
  printf("address that x is stored in is %p\n", (void *) &x);
  return 0;
}
```


Output of code:

```
value of x = 0,address that x is stored in is 0x7ffff6f81d54
```

Finding memory addresses in c

&x -> memory address of variable x (this is called referencing a variable)

```c
#include<stdio.h>
int main()
{
  int x=0;
  printf("value of x = %d,", x);
  // following code computes the address of x through &x, and recasts this as a typeless "pointer" so c can print (void *)
  printf("address that x is stored in is %p\n", (void *) &x);
  return 0;
}
```


Output of code:

```
value of x = 0,address that x is stored in is 0x7ffff6f81d54
```

Pointers

- A pointer stores a memory address.
- The type of the pointer is used by the compiler to see how much memory is stored there.

Pointers

- A pointer stores a memory address.
- The type of the pointer is used by the compiler to see how much memory is stored there.

Pointers

- A pointer stores a memory address.
- The type of the pointer is used by the compiler to see how much memory is stored there.

Pointers

- A pointer stores a memory address.
- The type of the pointer is used by the compiler to see how much memory is stored there.

Pointers

- A pointer stores a memory address.
- The type of the pointer is used by the compiler to see how much memory is stored there.

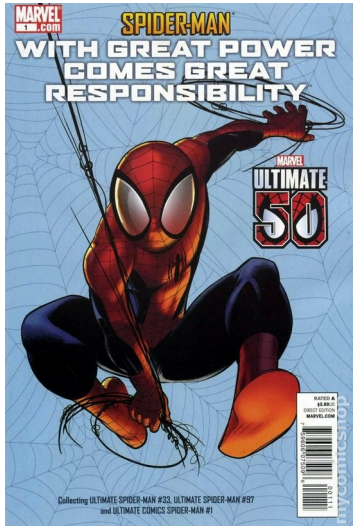
- In c the character * denotes an integer that points to an address.
- This is what we call “dereferencing” a “pointer”

Pointers

- A pointer stores a memory address.
- The type of the pointer is used by the compiler to see how much memory is stored there.

- In c the character * denotes an integer that points to an address.
- This is what we call “dereferencing” a “pointer”

- **Note that c DOES NOT enforce safety with pointers.**
  - **You can do many twisted and perverse things with pointers**



Arrays

- Int a[4] -> array of 4 integers. char c[4] -> array of 4 characters.

```c
#include<stdio.h>
int main()
{
  //initialize a 1D array with 4 entries
  int a[4]={4,3,2,1};
  for (int i=0; i<4; i++)
  {
    printf("a[%d]=%d,",i,a[i]);
  }
  printf("\n");

  //initialize a 2D array with 2x4 = 8 entries
  int b[2][4]={{1,2,3,4},{5,6,7,8}};
  for (int i=0; i<2; i++)
    {
      for (int j=0; j<4;j++)
        {
          printf("b[%d,%d]=%d,", i,j,b[i][j]);
        }
      printf("\n");
    }
}
```



```
a[0]=4,a[1]=3,a[2]=2,a[3]=1,
b[0,0]=1,b[0,1]=2,b[0,2]=3,b[0,3]=4,
b[1,0]=5,b[1,1]=6,b[1,2]=7,b[1,3]=8,
```

Arrays

- Int a[4] -> array of 4 integers. char c[4] -> array of 4 characters.

- An array is actually really a pointer to a block of memory.

Arrays

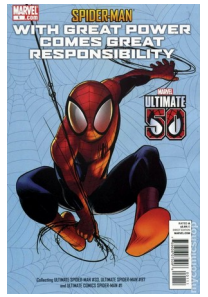
- Int a[4] -> array of 4 integers. char c[4] -> array of 4 characters.

- An array is actually really a pointer to a block of memory.

C does not prevent you from accessing a[5]

(because the address is a valid part of mem)

You may get a segmentation fault if the memory is protected



Demo

- Now onto the most dangerous part of the lecture: the live demo!

Stack frame

- The stack frame keeps track of all the memory that has been allocated and tracks the address of each variable.


```c
#include<stdio.h>
int main()
{
  int x = 5;
  int y = 9;
  printf("5 * 9 = %d\n",x*y);
  return 0;
}
```

Stack frame

- The stack frame keeps track of all the memory that has been allocated and tracks the address of each variable.


```c
#include<stdio.h>
int main()
{
  int x = 5;
  int y = 9;
  printf("5 * 9 = %d\n",x*y);
  return 0;
}
```

Stack frame

- The stack frame keeps track of all the memory that has been allocated and tracks the address of each variable.


```c
#include<stdio.h>
int main()
{
  int x = 5;
  int y = 9;
  printf("5 * 9 = %d\n",x*y);
  return 0;
}
```

Stack frame

- The stack frame keeps track of all the memory that has been allocated and tracks the address of each variable.


```c
#include<stdio.h>
int main()
{
  int x = 5;
  int y = 9;
  printf("5 * 9 = %d\n",x*y);
  return 0;
}
```