

CSCA48 Tutorial 3

Tabeeb Yeamin

January 30, 2020

Pointers!

- They are just a variable (like any other variable)
- This means they get their own locker in the memory model
- That locker will store a locker number for something else we want to access.

Pointer Usage

```
// store the address of x in pointer p  
p=&x;  
/*store the address of the first element  
  of stringy in pointer p*/  
p=&stringy[0];
```

Pointers!

- Must match pointer type to variable type

*/*Copy the contents of locker (p) into x
here (p) stands for a locker number, the actual
locker number stored in p.*/*

`x=*(p);`

*/*Copy the contents of locker (p+1) into x.
(p+1) is a locker number, equal to whatever
locker number is stored in p, plus one.*/*

`x=*(p+1);`

*/*Make the contents of locker (p+2) 5. Again,
(p+2) is a locker number, whatever locker
number is stored in p, plus 2.*/*

`*(p+2)=5;`

Equivalence between arrays and pointers

- The name of an array (by itself) is equivalent to the expression `&array[0]` - it gives us a pointer to the first entry in an array.
- Thus, if a function is declared as

```
int sum_array(int *p);
```

And we have an array

```
int array[10];
```

We can call the function as

```
sum_array(array);
```

or

```
sum_array(&array[0]);
```

These have identical meaning.

Modify the 'reverse' function

- In lecture, you did:

```
void reverse(char *input, char *output);
```

- takes pointers to an input string and an output string, then uses pointer indexing to make the output be the reverse of the input string

Mode the 'reverse' function

```
void reverse(char *input)
```

- reverses the string *in place*
- use pointer indexing only, and not create any temporary arrays within reverse()
- i.e. can't first reverse the string into another array and then copy the reversed one - have to swap things around as you go

```
char input[6] = "ABCDEF";  
reverse(input);  
printf("%s\n", input);
```

Poking Around (Time Permitting)

```
int main()
{
    char oneStr[1024]="I am string ONE!";
    char twoStr[1024]="I am string TWO!";
    char c='C';
    char d='*';
    char e='!';
    char f='@';
    int x=65;
    pokeAround(&twoStr[0]);
}
```

- implement: void pokeAround(char *p);
- print the characters stored in memory locations near p.
- use pointer+offset indexing to get this function to print out the text stored in oneStr, c,d,e,f, and x.

Poking Around (Time Permitting)

- HINT: we declare variables one after another, they usually end up in nearby locations in memory. You should use this idea somehow.