

CSE1301  
Computer Programming  
Lecture 16  
*Pointers 2*

1

## Topics

- Review of pointer basics
- More on how to dereference a pointer
- More on function passing by reference
- More pointer examples

2

## Recall

- A pointer *points* to another variable
- A pointer contains the *address* of another variable
- The **\*** operator is used to declare a pointer
  - eg `int* xPtr;`
- The **&** operator gives you the address of a variable
- The **\*** operator *dereferences* a pointer - gives you the value the pointer points to

3

## Recall

- A pointer is declared to point to a specific data type
- Any data type can have a pointer pointing to it
- Like any other variable, the type of a pointer is fixed
  - So a variable that is declared to be a **char\*** will *always* point to variables of type **char**

4

## More on Dereferencing

- Pointers point to other variables
- We need to go *through* the pointer and find out the value of the variable it points to
- We do this by *dereferencing* the pointer, using the **\*** operator
- But what is actually happening when we dereference?

5

## Algorithm for Dereferencing

- To *dereference* a pointer, use `*xPtr`, which means:
  1. **Go to xPtr**
  2. **Take the value you find there, and use it as an address**
  3. **Go to that address**
  4. **Return the value you find there**

6

### Pointer examples

```

char ch;
ch = 'A';

```

0x2000

ch:

'A'

  
  

```

char* chPtr=NULL;
chPtr=&ch;

```

0x2004

chPtr:

0x2000

7

### Pointer examples

```

char ch='A';
char init='B';
char* cPtr1=NULL;
char* cPtr2=NULL;
cPtr1=&ch;
cPtr2=&init;
*cPtr1 is ??
*cPtr2 is ??
cPtr1 is ??
cPtr2 is ??

```

0x2000

'A'

ch

0x2001

'B'

init

0x2002

0x2000

cPtr1

0x2003

0x2001

cPtr2

8

### Pointer examples

```

char ch='A';
char init='B';
char* cPtr1=NULL;
char* cPtr2=NULL;
cPtr1=&ch;
cPtr2=&init;
*cPtr1 is 'A'
*cPtr2 is ??
cPtr1 is ??
cPtr2 is ??

```

0x2000

'A'

ch

0x2001

'B'

init

0x2002

0x2000

cPtr1

0x2003

0x2001

cPtr2

9

### Pointer examples

```

char ch='A';
char init='B';
char* cPtr1=NULL;
char* cPtr2=NULL;
cPtr1=&ch;
cPtr2=&init;
*cPtr1 is 'A'
*cPtr2 is 'B'
cPtr1 is ??
cPtr2 is ??

```

0x2000

'A'

ch

0x2001

'B'

init

0x2002

0x2000

cPtr1

0x2003

0x2001

cPtr2

10

### Pointer examples

```

char ch='A';
char init='B';
char* cPtr1=NULL;
char* cPtr2=NULL;
cPtr1=&ch;
cPtr2=&init;
*cPtr1 is 'A'
*cPtr2 is 'B'
cPtr1 is 0x2000
cPtr2 is ??

```

0x2000

'A'

ch

0x2001

'B'

init

0x2002

0x2000

cPtr1

0x2003

0x2001

cPtr2

11

### Pointer examples

```

char ch='A';
char init='B';
char* cPtr1=NULL;
char* cPtr2=NULL;
cPtr1=&ch;
cPtr2=&init;
*cPtr1 is 'A'
*cPtr2 is 'B'
cPtr1 is 0x2000
cPtr2 is 0x2001

```

0x2000

'A'

ch

0x2001

'B'

init

0x2002

0x2000

cPtr1

0x2003

0x2001

cPtr2

12

### Pointer examples

```

char ch='A';
char init='B';
char* cPtr1=NULL;
char* cPtr2=NULL;
cPtr1=&ch;
cPtr2=&init;
...
cPtr1=&init;
*cPtr1 is ??
    
```

0x2000	'A'	ch
0x2001	'B'	init
0x2002	0x2000	cPtr1
0x2003	0x2001	cPtr2

13

### Assigning to pointers

- Pointers are just normal variables, that happen to have the type “address of <some type>”
- Pointers can be assigned to the address of any variable (of the right type)
- The value of a pointer can change, just like the value of any other variable
- The value of a pointer can be manipulated, just like the value of any other variable

14

### Pointer examples

```

char ch='A';
char init='B';
char* cPtr1=NULL;
char* cPtr2=NULL;
cPtr1=&ch;
cPtr2=&init;
*cPtr2='\.';
    
```

0x2000	'A'	ch
0x2001	'B'	init
0x2002	0x2000	cPtr1
0x2003	0x2001	cPtr2

15

### Pointer examples

```

char ch='A';
char init='B';
char* cPtr1=NULL;
char* cPtr2=NULL;
cPtr1=&ch;
cPtr2=&init;
*cPtr2='\.';
cPtr2 is ??
    
```

0x2000	'A'	ch
0x2001	'.'	init
0x2002	0x2000	cPtr1
0x2003	0x2001	cPtr2

16

### Pointer examples

```

char ch='A';
char init='B';
char* cPtr1=NULL;
char* cPtr2=NULL;
cPtr1=&ch;
cPtr2=&init;
*cPtr2='\.';
cPtr2 is 0x2001
(same as before - why would it change?)
    
```

0x2000	'A'	ch
0x2001	'.'	init
0x2002	0x2000	cPtr1
0x2003	0x2001	cPtr2

17

### Pointers as Parameters

- Recall that parameters are normally passed as copies
- **f(x)** takes a copy of the value of **x** and passes it to **f**
- This is called *passing by value*
- When you pass the address of a variable, you tell the function where to find the *actual* variable, not just a copy of it

18

### Pointers as Parameters (cont)

- Passing the address means the function can go and look at the variable, and change its value if it wants to.
- This is called *passing by reference*
- If you pass by reference, you can change the variable
- If you pass by value, you can only change the copy - this has no effect on the original variable

19

### Advantages of passing by reference

- Efficient
  - because you are not wasting space by making extra copies of variables every time you call a function
- Another way to return information from a function
  - How do you return more than one value from a function? Using parameters passed by reference!

20

### Disadvantages of passing by reference

- Harder to keep track of where (and how) a variable changes
  - Now changes could happen anywhere in a program, not just in the function a variable was *born* in (is local to)
- Functions are less *neat*
  - a function that returns a single value is mathematically neat, one that changes other values is messier to define precisely

21

### More pointer examples

```
int i=0;
int* myPtr=NULL;
int x=3;
myPtr=&x; /* set myPtr to point to x */
*myPtr=34; /* set x to be 34, using myPtr */
myPtr=&i; /* set myPtr to point to i */
printf("%d",*myPtr); /* print i using myPtr */
printf("%p",myPtr); /* print the address of i */
```

22

### More pointer examples

```
float x=5.4,y=78.25;
float* xPtr=NULL;
float* yPtr=NULL;
xPtr=&x; /* set xPtr to point to x */
yPtr=&y; /* set yPtr to point to y */
*xPtr=*yPtr; /* put the value of y in x using pointers */
*yPtr=45.0 /* put 45.0 in y using yPtr */
```

23