Ch 9. Pointers
Part 1

CS 2308
Fall 2011

Jill Seaman

Lecture 4

A Quote

A pointer is a variable that contains the address of a variable. Pointers are much used in C, partly because they are sometimes the only way to express a computation, and partly because they usually lead to more compact and efficient code than can be obtained in other ways. Pointers and arrays are closely related; this chapter also explores this relationship and shows how to exploit it.

Pointers have been lumped with the `goto` statement as a marvelous way to create impossible-to-understand programs. This is certainly true when they are used carelessly, and it is easy to create pointers that point somewhere unexpected. With discipline, however, pointers can also be used to achieve clarity and simplicity. This is the aspect that we will try to illustrate.

From: "The C Programming Language (2nd ed.)", Brian W. Kernighan and Dennis M.Ritchie, Englewood Cliffs, NJ: Prentice Hall. 1988. p. 93.

The Address Operator

- Consider main memory to be a sequence of consecutive cells (1 byte per cell).
- The cells are numbered. The number of a cell is its address.
- Each variable is allocated a sequence of cells, large enough to hold a value of its data type.
- The address operator (&) returns the address of a variable.

```
int x = 99;
cout << x << endl;
cout << &x << endl;

Output:
99
0xbffffb0c
```

- Addresses in C/C++ are displayed in hexadecimal. [`bffffb0c` = 3,221,224,204]

Pointer Variables

- Pointer variables are
  - variables that contain addresses
  - of other variables
  - of a certain, specified datatype.
- An asterisk is used to define a pointer variable

```
int *ptr;
```

- “ptr is a pointer to an int”
- The type of the variable pointed to is used in operations over pointers.

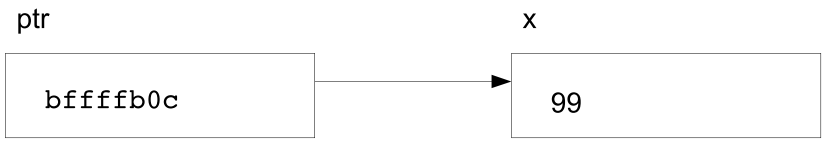
Using Pointer Variables

- ptr gets the address of x:

```
int x = 99;
int *ptr;

ptr = &x;
cout << x << endl;
cout << ptr << endl;

Output:
99
0xbffffb0c
```

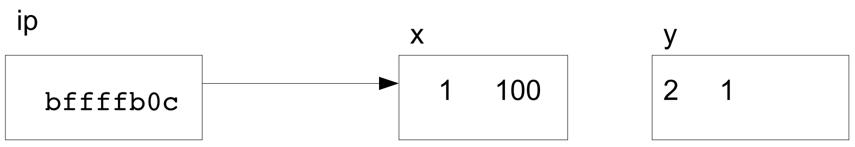


Dereferencing Operator: *

- The unary operator * is the *indirection* or *dereferencing* operator.
- `*ptr` is an alias for the variable that ptr points to.

```
int x = 1;
int y = 2;
int *ip;

ip = &x;      // ip points to x
y = *ip;      // y gets value of var ip points to
*ip = 100;    // the variable ip points to gets 100
```



Dereferencing Operator

- Another example

```
int x = 25, y = 50, z = 75;
int *ptr;

ptr = &x;
*ptr = *ptr + 100;

ptr = &y;
*ptr = *ptr + 100;

ptr = &z;
*ptr = *ptr + 100;

cout << x << " " << y << " " << z << endl;
```

Pointers and Arrays

- An array variable is a pointer to its first element.

```
int numbers[] = {10, 20, 30, 40, 50};

cout << "first: " << numbers[0] << endl;
cout << "first: " << *numbers << endl;

cout << &(numbers[0]) << endl;
cout << numbers << endl;

Output:
first: 10
first: 10
0xbffffafc
0xbffffafc
```

Pointer Arithmetic

- When you add a value to a pointer, you are actually adding that value times the size of the data type being referenced by the pointer.

```
int numbers[] = {10, 20, 30, 40, 50};

// sizeof(int) is 4.
// Let us assume numbers is equal to 0xbffff400
// Then numbers+1 is really 0xbffffb00 + 4, or 0xbffffb04
// And numbers+2 is really  0xbffffb00 + 8, or 0xbffffb08
// And numbers+3 is really  0xbffffb00 + 12, or 0xbffffb0c
```

Pointer Arithmetic

- Note unary * has higher precedence than +

```
int numbers[] = {10, 20, 30, 40, 50};

cout << “second: ” << numbers[1] << endl;
cout << “second: ” << *(numbers+1) << endl;

cout << "size: " << sizeof(int) << endl;
cout << numbers << endl;
cout << numbers+1 << endl;

Output:
second: 20
second: 20
size: 4
0xbffffafc
0xbffffb00
```

- Note: array[index] is equivalent to *(array + index)

Pointers and Arrays

- pointer operations can be used with array variables.

```
int array[10];
cin >> *(array+3);
```

- subscript operations can be used with pointers.

```
int array[] = {1,2,3};
int *ptr = array;
cout << ptr[2];
```

- You cannot change the value of the array variable.

```
double totals[20];
double *dptr;
dptr = totals;    //ok
totals = dptr;    //not ok, totals is a const
```