Computer Programming Using C
COP 3275 - Summer 2017

Lecture 16: Pointers
Pointer Variables

• The first step in understanding pointers is through visualizing what they represent at the machine level.

• The main memory of your computer is divided into *bytes* (8 bits, each is either 0 or 1).
• Each byte in the memory has a unique address, thus memory of n bytes in memory has address value that ranges from 0 to n-1.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>01010011</td>
</tr>
<tr>
<td>1</td>
<td>01100101</td>
</tr>
<tr>
<td>2</td>
<td>01100001</td>
</tr>
<tr>
<td></td>
<td>.</td>
</tr>
<tr>
<td></td>
<td>.</td>
</tr>
<tr>
<td>n-1</td>
<td>01001001</td>
</tr>
</tbody>
</table>
• Any executable program consists of both machine instructions (your C program) and variables (data in your program).

• Each variable in a program occupies one or more bytes of memory, where the address of the first byte is said to be the address of the variable.

• If a variable i occupies two bytes (address 2000 and 2001), thus the address of the variable i is 2000.
• Although such addresses are represented by numbers, their range may differ from the range of an integer variable (so we can’t store such address in ordinary integer variable).

• Addresses can be stored in special variable type named **pointer variables**.

• When we store the address of a variable i in the pointer variable X, we say that X “points to” i.
Declaring Pointer Variables

• The pointer variable is declared as an ordinary variable and the name is preceded by an asterisk:

```c
int *p;
```

• \( p \) is a pointer variable capable of pointing to objects of type integer (\( \text{int} \)).

• The term \textit{object} is more general than \textit{variable}, since the pointer may point to any part of memory (that doesn't belong to a variable).
In C, the pointer variable points only to objects of its particular type (the *referenced type*):

```c
int *p;     /* points only to integers */
double *q;  /* points only to doubles */
char *r;    /* points only to characters */
```

There are no restrictions on what the referenced type may be (in some cases, a pointer can point to another pointer).
The Address and Indirection Operators

- C provides a pair of operators for use with pointers.
  - To find the address of a variable, we use the \& (address) operator (e.g., \&x is the address of variable x in the memory).
  - To gain access to the object that a pointer points to, we use the * (indirection) operator. (e.g., *p is the object to which p currently points to).
Declaring a pointer variable that can point to an integer variable.

```c
int *p;
```

Pointer `p` is not initialized and is pointing nowhere.

```c
int i;
p = &i;
```

`p` is now pointing to the address of `i` in memory, without affecting the value of `i`.

This can still be done that way:

```c
int j;
int *q = &j;
```
The Indirection Operator

• Once a pointer variable points to an object, we can use the * (indirection) operator to access what’s stored in the object.

• If \( p \) points to \( i \), we can print the value of \( i \) as follows:

\[
\text{printf}("\%d\n", \*p);
\]

• Applying & to a variable produces a pointer to the variable. Applying * to the pointer takes us back to the original variable:

\[
j = \*\&i; \quad /* \text{same as } j = i; */
\]
As \( p \) now points to \( i \), then \( *p \) has the same value as \( i \) and *Changing the value of \( *p \) changes the value of \( i \).*
```c
int i, *p;
p = &i;

i = 1;

printf("%d\n", i);    /* prints 1 */
printf("%d\n", *p);    /* prints 1 */

*p = 2;

printf("%d\n", i);    /* prints 2 */
printf("%d\n", *p);    /* prints 2 */
```
• Applying the indirection operator to an uninitialized pointer variable causes undefined behavior:

```c
int *p;
printf("%d", *p);    /* garbage */
```

• Assigning a value to *p is particularly dangerous, p might point somewhere in memory and modifying it can cause big problems:

```c
int *p;
*p = 1;                /* WRONG */
```