CSE 30: Computer Organization and Systems Programming

Lecture 6:
C Data types: Pointers

Diba Mirza
University of California, San Diego
### C Integer Data Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>4 bytes</td>
<td>(word size of machine)</td>
</tr>
<tr>
<td>short</td>
<td>2 bytes</td>
<td></td>
</tr>
<tr>
<td>long</td>
<td>4 bytes</td>
<td>(at least 4 bytes)</td>
</tr>
<tr>
<td>long long</td>
<td>8 bytes</td>
<td></td>
</tr>
<tr>
<td>unsigned</td>
<td>4 bytes</td>
<td></td>
</tr>
</tbody>
</table>

2's complement

Unsigned a;
Unsigned short b;
(inside a function)
automatic → stack

Address of b is the lower 2 bytes of the address of the two bytes.

b = 0x00
b = 0x01 0x00


32-bit machine 4 bytes
64-bit " 8 bytes

Word size
Other datatypes

cchar c='a'; /* 1 byte */

• By default, may be signed or unsigned
• Always enforce using the appropriate keyword
  unsigned char c;
  signed char c;

float f; /* 4 bytes */
double d; /* 8 bytes */
Accessing value, Lvalue and Rvalue

```
int x, y; /* What is the value of x? */
y = 10;
x = 20;
y = x > y ? x : y;
```

- `x` is an lvalue because it is an identifier.
- `x` is not initialized, so it has a junk value.
- `y` is initialized to 10, so it is an rvalue.
- `x > y` compares the lvalue of `x` to the rvalue of `y`.
- The ternary operator `?` is used to assign the result of the comparison to `y`.

```
3 == a; // x 20 > 10 ? 20 : 10;
    
    y = 20;
```
Pointers

- **Pointer**: A variable that contains the **address** of a variable
- **Declaration**: `type * pointer_name;`

```c
int y = 5;
int *x;  // x is intended to store the memory location of an integer
```

**How do we initialize a pointer?**

```c
int *x = 10;  // Compiler error
int *x = NULL;  // in C
```

**Why pointers?**
Accessing location

To access the location/address, use the address operator ‘&’

```c
int *x = NULL;  // Set x to NULL
t
int y = 20;     // Set y to 20

x = &y;  // Get address of y
```

Diagram:
- `int *x = NULL;`
- `int y = 20;`
- `x = &y;` (Get address of y)
- `102` (Address of y)
- `20` (Value of y)
Pointers

• How to make a pointer point to something.

```c
int *x;
int y;
y = 3;
x = &y;
```
Pointers

• How to make a pointer point to something?

```c
int *x, y;
y = 3;
x = &y;
```

`sizeof(x) = 4`

*x points to y*
• Short hand diagram for the following scenario

\[
\begin{array}{c}
\text{x} & \text{120} & \text{y} & \text{3} \\
\text{102} & & & \\
\end{array}
\]

(Don't forget that \text{x} is a variable just like any other)
To change the value of a variable using pointers: use dereference * operator to left of pointer name

```c
int y=3, *x;
x = &y;
*x = 5;
```
Two ways of changing the value of any variable

Why this is useful will be clear when we discuss functions and pointers

Change the value of y directly:

Change the value of y indirectly (via pointer x):

\[ y = 5; \]

\[ \text{int} \ z = \* x; \]

\[ \* x = 5; \]
Q: Which of the following pointer diagrams best represents the outcome of the above code?

A. [Diagram A]

B. [Diagram B]

C. Neither, the code is incorrect
Q: This code gives a warning at compile time. Why?

```c
char *p;
int y;
p = &y;
```

A. The pointer ‘p’ is made to point to a variable of incompatible type

B. *p does not contain a valid value because y was not initialized
Q: What happens when we run the following code?

```c
int *p;
*p = 5;
```

A.  

```
5
```

B.  

```
5
```

C. Compile time error

D. Runtime error (p was never made to point to a valid memory location)
Segmentation faults (aka segfault)

• Indicates that your program has crashed!
• What caused the crash?
  – Segfaults occur if your program is trying to read or write an illegal memory location.
Q: What is the output of this code?

```c
int *p, x = 5;
p = &x;
printf("%d",(*p)++);
```

A. The value pointed to by p, which is 5
B. The value pointed to by p plus one, which is 6
C. Undefined
D. Compiler error
E. Segmentation fault
Two important facts about Pointers

1) A pointer can only point to one type – (basic or derived) such as int, char, a struct, another pointer, etc

2) After declaring a pointer: `int *ptr;`
`ptr` doesn’t actually point to anything yet. We can either:

- make it point to something that already exists, or
- allocate room in memory for something new that it will point to
- Null check before dereferencing

```c
if (ptr) {
    // Only enter here if ptr is not NULL
    ?
}
```