CS2351
Data Structures

Lecture 7:
A Brief Review of Pointers in C
About this lecture

- **Pointer** is a useful object that allows us to access different places in our memory.
- We will review the basic use of pointer.
- **Usage:** Many data structures are dynamic, and their shapes change from time to time.
  - The use of pointers allows us to change the shapes, in a very flexible way.
Example: A Dynamic List

• Suppose we have some people, who are waiting in a line to buy Disneyland tickets.

But from time to time, these people may bring in friends to line after them ...
Example: A Dynamic List

• To maintain the ordering of the people in the line, we can obviously use an array

• However, there will be problems ...
  - “Insertion after” requires $O(n)$ time in the worst case

• Later, we will study dynamic list
  - “Insertion after” can be done in $O(1)$ time
What are Pointers?

• Consider an array $A$ with 10 integers


• Also, we can get or modify the content of an entry (Ex: $y = A[3]$; $A[9] = 113$)
What are Pointers?

• In fact, our memory is just a long array

• Like a normal array, each entry has a location (or an address), and contains space for storing data

• To access an entry in our memory, we can use a pointer to specify its location
What are Pointers?

- In C, we declare a pointer using the following syntax:

  ```
  int *ptr;
  ```

- The above line declares a variable `ptr`, which is used to point at a location in the memory for storing an integer.

- Similarly, we can also do something like:

  ```
  char *cptr;
  ```
Pointers in Action

• Once we have declared a pointer, we can do something like:

\[ \text{ptr} = 0; \]

• The above line tells \text{ptr} to point to the location 0 in the memory

• This doesn’t seem very useful, since there is no particular reason why we want to access location 0 in the memory …
Pointers in Action

• As mentioned, our memory is an array
• Each variable that we declare occupies a certain location in the memory
• Ex: When we declare

```c
int a;
```

then a certain part of memory will be used by `a`
Pointers in Action

• In C, the location of a in the memory can be obtained by &a.

• Then we can write something like:

\[
\text{ptr} = \&a; \\
\]

which tells ptr to point to the location of a in the memory.
Pointers in Action

• In C, when a pointer `ptr` points to a location in the memory, we can get the value stored in that location by `*ptr`

```c
int a, b, *ptr;
ptr = &a;
a = 5;
printf("value pointed by p: %d\n", *ptr);
a = 8;
printf("value pointed by p: %d\n", *ptr);
```

In C, `*p` is called dereferencing of a pointer `ptr`
Pointers in Action

• In C, we can also get or modify the content in the location pointed by a pointer $\textit{ptr}$

• The syntax is as follows:

```c
b = *ptr ;
*ptr = 15 ;
```

• The first line changes the content of $b$ to the content pointed by $\textit{ptr}$

• The second line changes the content pointed by $\textit{ptr}$ to be 15
Pointers in Action

• What will happen in the following code?

```c
int a, *ptr;
ptr = &a;
a = 5;
printf("value pointed by p: %d\n", *ptr);
*ptr = 15;
printf("value pointed by p: %d\n", *ptr);
printf("value stored by a: %d\n", a);
```
What will happen in the following code?

```c
int a, b, *ptr;
a = 5;  b = 3;
ptr = &a;  *ptr = 21;
ptr = &b;  *ptr = 15;
printf("value stored by a: %d\n", a);
printf("value stored by b: %d\n", b);
```
Remarks

• Although *ptr usually refers the content of the location pointed by ptr, an exception is during declaration.

• The statement:

\[
\text{int } \ast\text{ptr} = 0; \\
\]

is exactly the same as

\[
\begin{align*}
\text{int } \ast\text{ptr} \\
\text{ptr} = 0;
\end{align*}
\]
Address of Variable

- In C, each variable has a location in the memory for storing its content.
- It is true even for a pointer variable!!
- What will happen?

```c
int a, *ptr = 0;
printf("the value of ptr: %x\n", ptr);
printf("address of ptr: %x\n", &ptr);
ptr = &a;
printf("the value of ptr: %x\n", ptr);
printf("address of ptr: %x\n", &ptr);
```
Address of Variable

• Each entry in an array also has an address
• What will happen in the following code?

```c
int a[10] ;
printf("address of a[0]: %x\n", &(a[0]));
printf("address of a[1]: %x\n", &(a[1]));
```

• In fact, the array name is a “constant pointer” to the location of its first entry

```c
printf("the value of a: %x\n", a);
```
Pointer Arithmetic

• The entries of an array in C occupies contiguous locations in the memory
• When a pointer points to a certain entry in an array, we can increment the pointer to point to the next entry

```c
int a[10], *ptr;
ptr = a; // same as ptr = &(a[0]);
ptr++; // ptr now points at a[1]
ptr = ptr + 1; // ptr now points at a[2]
```
In fact, we can do more:

```c
int a[10], *ptr;
ptr = a;       // same as ptr = &a[0];
ptr = ptr + 3; // ptr now points at a[3]
printf("a[7] = %d\n", *(ptr + 4));  
// note: ptr still points at a[3]
```

Similarly, we can decrement a pointer to point back to the previous entry.
Remarks

• When we add 1 to \texttt{ptr}, the actual value stored \texttt{ptr} may not be increased by 1
  - Reason: this operation is for a change in the memory location, and the change depends on the type of thing pointed by \texttt{ptr}

```c
int a[10], *ptr = a;
printf("value of ptr: %x\n", ptr);
printf("value of ptr + 1: %x\n", ptr + 1);
```

Note: In a 32-bit machine, the change is 4, since each integer occupies 4 bytes in the memory
Segmentation Fault

- A pointer allows us to access freely any location in the memory
- However, some part of the memory is forbidden (ex: it may be running our OS)
- When we try to touch the content in a forbidden area, segmentation fault occurs

```c
int *ptr = 0;
printf("value of ptr: %x\n", ptr);
printf("value pointed by ptr: %d\n", *ptr);
```
Casting and Bus Error

• In C, we are allowed to perform “casting” to view a variable as a different type from its declared type

```c
char a;
int b;
a = 'A';
b = (int) a; // casting a as int type
printf("value of b: %d\n", b);
```
Casting and Bus Error

• We can also cast pointers

```c
char a[4]; int *ptr;
ptr = (int *) a;
printf("value pointed by ptr: %d\n", *ptr);
```

• The above is like:

```
|   |   |   | A |
---|---|---|---|
|   |   |   |   |
```

... 0 0 0 A 0 0 0 ...
Casting and Bus Error

• However, we need to be very careful ...
• What will happen in the following code?

```c
char a[4]; int *ptr;
a[0] = 0, a[1] = 0, a[2] = 0, a[3] = 65;
ptr = (int *) &a[1];
printf(“value pointed by ptr: %d\n”, *ptr);
```

• A bus error occurs, because we try to dereference an integer pointer at a location that is impossible for storing an integer.