CSE 31 Computer Organization

Lecture 5 – C Pointers (cont.), C Strings

Announcement

- Labs
  - Lab 1 due this week (**no grace period** after due date)
    - Demo is REQUIRED to receive full credit
  - Lab 2 out this week
    - Due at 11:59pm on the same day of your next lab
    - You must demo your submission to your TA within 14 days
- Reading assignment
  - Chapter 4-6, 8.7 of K&R (C book)
  - Reading 01 (zyBooks 1.1 – 1.5) due 20-SEP
    - Complete Participation Activities in each section to receive grade towards Participation
    - IMPORTANT: Make sure to submit score to CatCourses by using the link provided on CatCourses

Announcement

- Homework assignment
  - Homework 01 (zyBooks 1.1 – 1.5) due 27-SEP
    - Complete *Challenge Activities* in each section to receive grade towards Homework
    - IMPORTANT: Make sure to submit score to CatCourses by using the link provided on CatCourses

Pointer Arithmetic (review)

- What is valid pointer arithmetic?
  - Add an integer to a pointer.
  - Subtract integer from pointer.
  - Subtract 2 pointers (in the same array).
  - Compare pointers (`<`, `<=`, `==`, `!=`, `>`, `>=`)
  - Compare pointer to `NULL` (indicates that the pointer points to nothing).
- Everything else is illegal since it makes no sense:
  - adding two pointers
  - multiplying pointers
  - subtract pointer from integer

Pointer Arithmetic Summary

- `x = *(p + 1)` ?
  - `x = *(p + 1);`
- `x = *p+1` ?
  - `x = (*p) + 1;`
- `x = (*p)++` ?
  - `x = *p; *p = *p + 1;`
- `x = *p++` ? `(*p++)` ? `*(p)++` ? `*(p++)` ?
  - `x = *p; p =  p + 1;`
- `x = *++p` ?
  - `p = p + 1; x = *p;`
- `x = ++*p` ?
  - `*p = *p + 1; x = *p;`
- Lesson?
  - Using nothing but the standard `*p++` , `(*p)++` causes more problems than it solves!

Pointers (1/4)

- Sometimes you want to have a function to increment a variable
- What gets printed?

```
void AddOne(int  x)
{     x =  x + 1;    }

int y = 5;
AddOne( y);
printf(“y = %d\n”, y);
```

y = 5

Pointers (2/4)

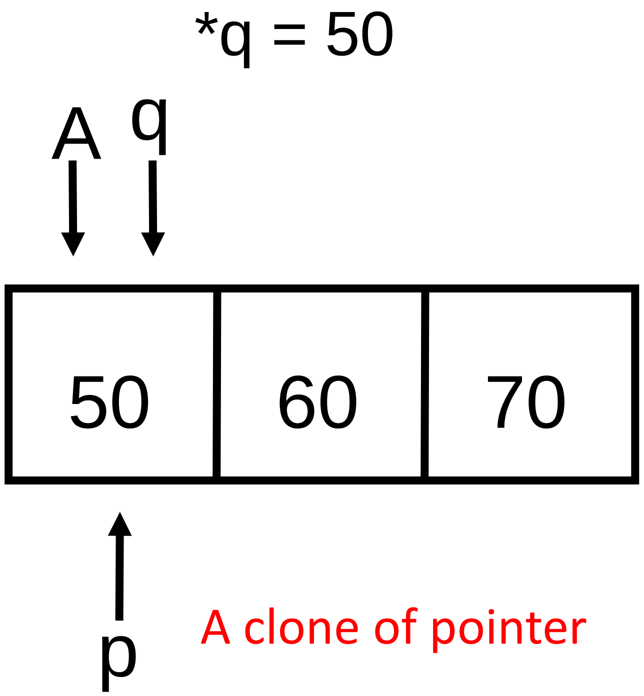
- Solved by passing in a pointer to our subroutine.
- Now what gets printed?

```
void AddOne(int *p)                      y = 6
{     *p = *p + 1;     }

int y = 5;
AddOne(&y);
printf("y = %d\n", y);
```

Pointers (3/4)

- But what if what you want changed is a pointer
- What gets printed?

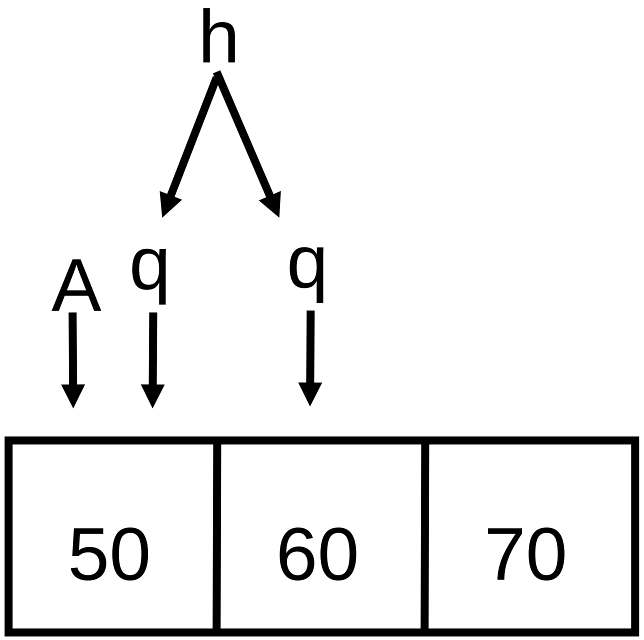
```
void IncrementPtr(int  *p)
{    p =  p + 1;    }

int A[3] = {50, 60, 70};
int *q = A;
IncrementPtr( q);
printf("*q = %d\n", *q);
```



Pointers (4/4)

- Solution! Pass a pointer to a pointer, declared as `**h`
- Now what gets printed?

```
void IncrementPtr(int **h)
{    *h = *h + 1;    }

int A[3] = {50, 60, 70};
int *q = A;
IncrementPtr(&q);
printf("*q = %d\n", *q);
```



```
*q = 60
```

Quiz:

How many of the following are invalid?

I. pointer + integer
II. integer + pointer
III. pointer + pointer
IV. pointer – integer
V. integer – pointer
VI. pointer – pointer
VII. compare pointer to pointer
VIII. compare pointer to integer
IX. compare pointer to 0
X. compare pointer to `NULL`

```
#invalid
   a) 1
   b) 2
   c) 3
   d) 4
   e) 5
```

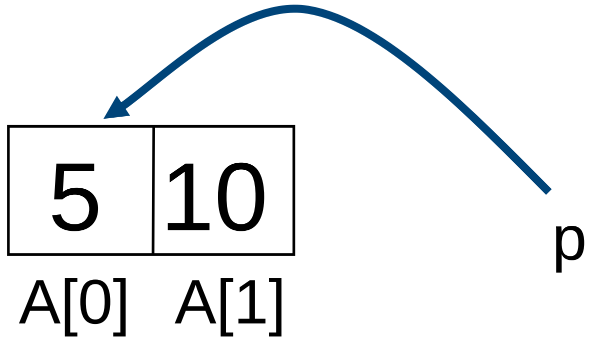
Quiz:

How many of the following are invalid?

I. pointer + integer
II. integer + pointer
III. pointer + pointer
IV. pointer – integer
V. integer – pointer
VI. pointer – pointer
VII. compare pointer to pointer
VIII. compare pointer to integer
IX. compare pointer to 0
X. compare pointer to `NULL`

```
#invalid
a) 1
b) 2
c) 3
d) 4
e) 5
```

Quiz:

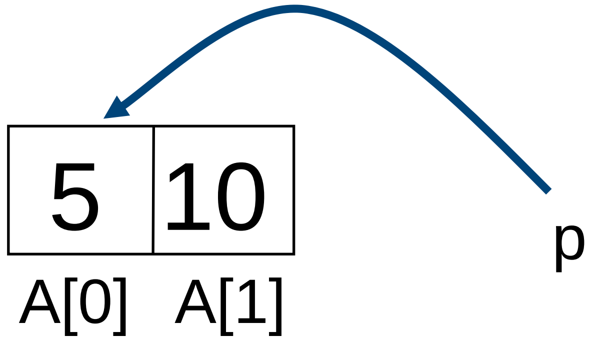


```
int main(void){
    int A[] = {5,10};
    int *p = A;

    printf("%p %d %d %d\n", p, *p, A[0], A[1]);
    p =  p + 1;
    printf("%p %d %d %d\n", p, *p, A[0], A[1]);
    *p = *p + 1;
    printf("%p %d %d %d\n", p, *p, A[0], A[1]);
}
```

- If the first `printf` outputs 100 5 5 10, what will the other two `printf` output?

- a) 101 10 5 10 then 101 11 5 11
  b) 104 10 5 10 then 104 11 5 11
  c) 101 <other> 5 10 then 101 <3-others>
  d) 104 <other> 5 10 then 104 <3-others>
  e) One of the two printfs causes an ERROR

Quiz:



```
int main(void){
    int A[] = {5,10};
    int *p = A;

    printf("%p %d %d %d\n", p, *p, A[0], A[1]);
    p =  p + 1;
    printf("%p %d %d %d\n", p, *p, A[0], A[1]);
    *p = *p + 1;
    printf("%p %d %d %d\n", p, *p, A[0], A[1]);
}
```

- If the first `printf` outputs 100 5 5 10, what will the other two `printf` output?

- a) 101 10 5 10 then 101 11 5 11
  b) 104 10 5 10 then 104 11 5 11
  c) 101 <other> 5 10 then 101 <3-others>
  d) 104 <other> 5 10 then 104 <3-others>
  e) One of the two printfs causes an ERROR

Pointers in C

- Why use pointers?
  - If we want to pass a huge struct or array, it’s easier / faster to pass a pointer than the whole thing.
  - In general, pointers allow cleaner, more compact code.
- So, what are the drawbacks?
  - Pointers are probably the single largest source of bugs in software, so be careful anytime you deal with them.
    - Dangling reference (premature free)
    - Memory leaks (tardy free)
- Make sure you know what you are doing!

Pointers Summary

- Pointers and arrays are virtually the same
- C knows how to increment pointers
- C is an efficient language, with little protection
  - Array bounds not checked
  - Variables not automatically initialized
- (Beware) The cost of efficiency is more overhead for the programmer.

C Strings

- A string in C is just an array of characters.

```
char string[] = "abc";
```

- How do you tell how long a string is?
  - Last character is followed by a 0 byte (null terminator)

```
int strlen(char s[])
{
        int n = 0;
        while (s[n] != 0)
        n++;
        return n;
}
```