Lecture06: Pointers
4/1/2013

Slides modified from Yin Lou, Cornell CS2022: Introduction to C

1

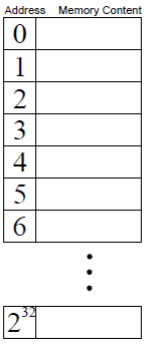
Pointers

- A pointer is a variable that contains the (memory) address of another variable
  - What is a memory address?

2

Memory

- Variables are stored in computer memory
- Think of memory as a very large array
  - Like an one-column excel sheet, each entry/location is a byte.
  - Every byte/location in memory has an **address**
  - An address is an integer, just like an array index
- In C, a **memory address is called a pointer**
  - C lets you manipulate memory locations directly (store, add, subtract,etc.)



3

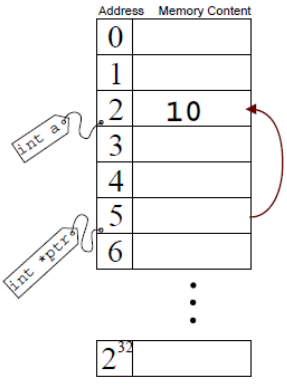
Pointer Declaration & Usage

- Pointer Declaration

```
int *p1;   // int* p1 also okay
float *p2;
unsigned int *p3;
char *p4;
void *p5;
```

- Pointer Usage

```
int a = 10;
int *ptr;
ptr = &a;
```

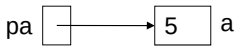


4

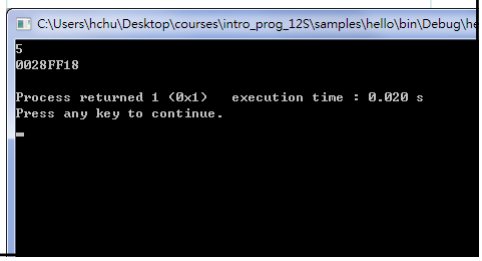
Two Operators for Pointers

- & (“address of”) operator
  - Returns the address of its argument (i.e., the row number in excel)
  - Said another way: returns a pointer to its argument
  - The argument must be a variable name
- * (“dereference”) operator
  - Returns the value stored at a given memory address
  - The argument must be a pointer

```
int a = 5;
int *pa = &a;
```

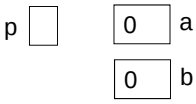


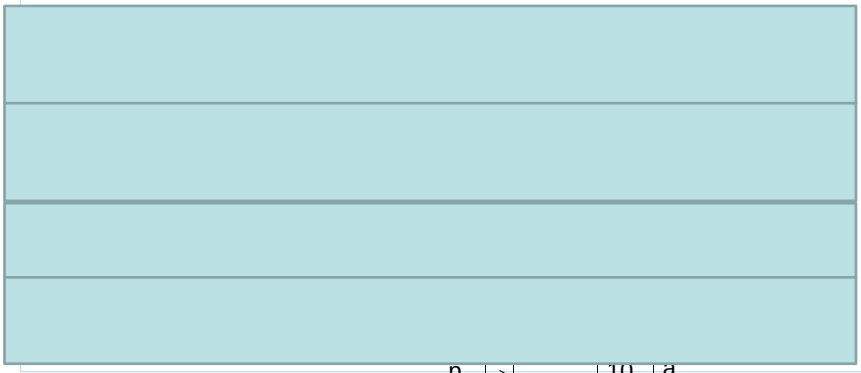

```
printf(“%d\n”, *pa);
printf(“%p\n”, pa); // ?
```

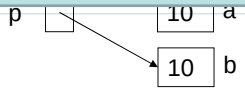


Example

```
int a = 0;
int b = 0;
int *p;
```







6

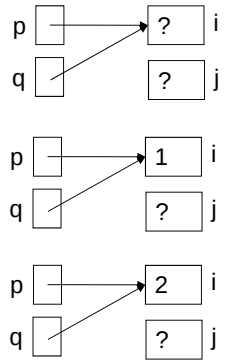
More example

```
int i, j, *p, *q;

p = &i;
q = p;

*p = 1;

*q = 2;
```



7

Recall the & in scanf( )?

```
int i;
scanf("%d\n", &i);
```

- Pass the pointer to i, so that scanf() can modify its value.

8

What are pointers good for?

Passing pointers to functions so that functions can change the values pointed by these pointers

```
void swap(int* c, int* d)
{
   int t = *c;
   *c= *d;
   *d = t;
}

void main()
{
   int a = 5, b = 3;
   printf("Before swap: a = %d b = %d\n", a, b);
   swap(&a, &b);
   printf("After swap: a = %d b = %d\n", a, b);
}
```

9

What are pointers good for?

Multiple Return Values

```
void initialize(int *a, char *b)
{
   *a = 10;
   *b = 'x';
}

void main()
{
   int a, b;
   initialize(&a, &b);
}
```

10

In-Class Exercise 5-1

Write a function that finds the largest and smallest elements in an array. This function has the following prototype:

```
void max_min(int a[], int n, int *max, int *min);
```

A call to `max_min` might be as follows:

```
max_min(b, N, &big, &small);
```

Also write a complete program that reads 10 integers into an array and passes the array to `max_min`, and prints the results as follows:

```
Enter 10 numbers: 34 82 49 102 7 94 23 11 50 31
Largest: 102
Smallest: 7
```

11

Exercise 5.1 Solution

```
#include <stdio.h>

void max_min(int a[], int n, int *max, int *min)  // max = &big; min = &small;
{
    int i;
    *max = a[0];
    *min = a[0];

    for (i=1; i<n; i++)
    {
        if (*max < a[i])
            *max = a[i];
        if (*min > a[i])
            *min = a[i];
    }
}

int main()
{
    int b[10];
    int i, big, small;

    printf("Input 10 numbers:");
    for (i=0; i<10; i++)
        scanf("%d", &b[i]);

    max_min(b, 10, &big, &small);
    printf("Largest: %d \nSmallest: %d", big, small);
    return 1;
}
```

12

Example – pointer to pointer

```
int i;            // Integer i
int *p;           // Pointer to integer
int **m;          // Pointer to int pointer

p = &i;           // p now points to i
printf("%p", p);  // Prints the address of i (in p)

m = &p;           // m now points to p
printf("%p", m);  // Prints the address of p (in m)
```

13

Use `const` to Protect Arguments

```
/* f cannot change the value of p */

void f(const int *p)
{
   *p = 0;    /* ERROR! */
}
```

14

Pointers as Return Values

```
/* return pointer to whichever integer is larger */
int *max(int *a, int *b)
{
   if (*a > *b)
       return a;
   else
       return b;
}
```

15

Pointers Are (Very) Dangerous

What's wrong with the code?

```
void main()
{
   int *p;
   *p = 10;
   printf("%d", *p);
}
```

```
int *f()
{
   int i;
   return (&i);

}
```

16

Pointers Are (Very) Dangerous

What's wrong with the code?

```
void main()
{
   char x = 'a';
   char *p = &x;
   p++;
   printf("%c\n", *p);
}
```

17

Recall: Arrays

- To declare an array, use [], e.g.:
  - int a[5]; // Creates an array with 5 integer elements
- The size of an array can't be changed
- The number between the brackets must be a constant
- You can give initial values for array elements, e.g:
  - int a[5] = {3, 7, -1, 4, 6};
  - A better way: int a[] = {3, 7, -1, 4, 6}; // Let the compiler calculate the size

18

Recall: Arrays

- Array indices in C are zero-based, e.g. a[0], a[1], ..., a[4]

```
void main()
{
	int a[] = {3, 7, -1, 4, 6};
	int i;
	double average = 0;

	// compute the average of values in an array
	for (i = 0; i < 5; ++i)
	{
		average += a[i];
	}
	average /= 5;  // average = average / 5;
	printf("Average = %.2f\n", average);
}
```

19

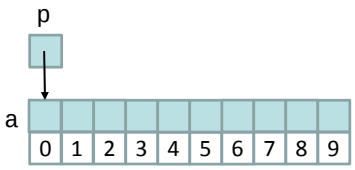
Pointers and Arrays

- Pointers and arrays are closely related
  - An array variable is actually just a pointer to the first element in the array
- You can access array elements using array notation or pointers
  - `a[0]` is the same as `*a`
  - `a[1]` is the same as `*(a + 1)`
  - `a[2]` is the same as `*(a + 2)`
- You can add or subtract pointers like integers

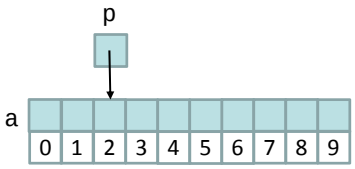
20

Adding an integer to a Pointer

```
int a[10];
int *p, *q, i;
p = a;
```

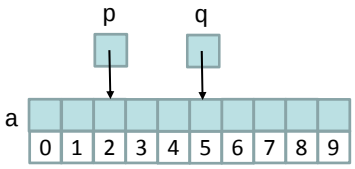



```
p = &a[2];
```




```
q = p + 3;
```

```
p += 6;
```



21

Pointers and Arrays

- Access array elements using pointers

```
void main()
{
	int a[] = {3, 7, -1, 4, 6};
	int i;
	double average = 0;

	// compute the average of values in an array
	for (i = 0; i < 5; ++i)
	{
		average += *(a + i);
	}
	average /= 5;
	printf("Average= %.2f\n", average);
}
```

22

Pointers and Arrays

- If pa points to a particular element of an array, (pa + 1) always points to the next element, (pa + i) points i elements after pa and (pa - i) points i elements before.
- The only difference between an array name (a) and a pointer (pa):
  - A pointer is a variable, so pa = a and pa++ are legal
  - An array name is not a variable, so a = pa and a++ are illegal

```
int a[10];
int *pa;
pa = a; pa++;     // okay
a = pa; a++;      // wrong!!
```

23

Pointers, Arrays and Functions

- It's possible to pass part of an array to a function, by pass a pointer to the beginning of the sub-array.
  - f(&a[2])
  - f(a + 2)
- Within f, the parameter declaration can read
  - f(int arr[]) { ... }
  - f(int* arr) { ... }

24

In-Class Exercise 5-2

Complete the following program that prints the characters in a reverse order.

```
char s[10] = "abcde";
char* cptr;
for (cptr = &s...)…
{
        printf("%c", *cptr);
}
```

25

Dynamically Allocating Arrays

- malloc: Allocate contiguous memory dynamically

```
int n;
if (…)
    n =10;
else
    n = 5;
int* p = (int*) malloc(n * sizeof(int));

int n = 5;
int p[n];
```

  - An array of size n
- free: De-allocate the memory
  - `free(p);`
- Make sure malloc and free are paired!

26

In-Class Exercise 5-3

Modify the code in Exercise 5-1 to ask for how many numbers. Use `malloc()` and `free()` to allocate the size of the array.

```
Enter N: 10
Enter 10 numbers: 34 82 49 102 7 94 23 11 50 31
Largest: 102
Smallest: 7
```

27

Exercise 5.3 Solution

```
#include <stdio.h>

void max_min(int a[], int n, int *max, int *min)  // max = &big; min = &small;
{
    int i;
    *max = a[0];
    *min = a[0];

    for (i=1; i<n; i++)
    {
        if (*max < a[i])
            *max = a[i];
        if (*min > a[i])
            *min = a[i];
    }
}

int main()
{
    int* b;
    int i, big, small;

    b = (int *)malloc(10*sizeof(int));
    printf("Input 10 numbers:");
    for (i=0; i<10; i++)
        scanf("%d", &b[i]);

    max_min(b, 10, &big, &small);
    printf("Largest: %d \nSmallest: %d", big, small);
    free(b);
    return 1;
}
```

28