Lecture05: Pointers and Arrays
10/15/2012

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?
  - Sound abstract …

2

Memory

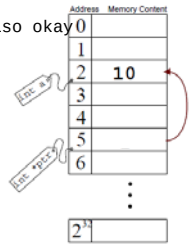
- Variables are stored in computer memory
- Think of memory as a very large array
  - Like an one-column excel sheet, each entry/location is a byte.
  - Every location in memory has an address
  - An address is an integer, just like an array index
- In C, a memory address is called a pointer
  - C lets 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;
```

pa → 5 a

```
printf(“%d\n”, *pa);

printf(“%p\n”, pa); // ?
```

5

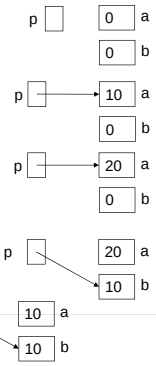
Example

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

a = 10;
p = &a;
printf(“%d\n”, *&a);

*p = 20;          // a = ? b = ?

p = &b;

*p = 10;          // a = ? b = ?
a = *p;           // a = ? b = ?
```

p → 20 a; 0 b

p → 0 b

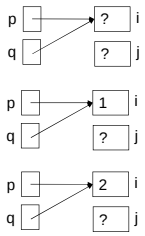
p → 10 a; 10 b



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);
```

- Do you understand?

8

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)
```

9

What are pointers good for?

Passing Pointers to Functions

```
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);
}
```

10

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);
}
```

11

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
```

12

Use `const` to Protect Arguments

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

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

13

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;
}
```

14

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);

}
```

15

Pointers Are (Very) Dangerous

What's wrong with the code?

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

16

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

17

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);
}
```

18

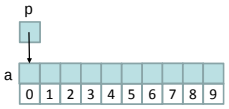
Pointers and Arrays

- 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

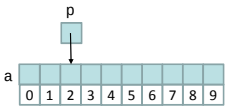
19

Adding an integer to a Pointer

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

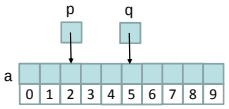



```
p = &a[2];
```




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

```
p += 6;
```



20

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);
}
```

21

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!!
```

22

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) { ... }

23

Dynamically Allocating Arrays

- malloc: Allocate contiguous memory dynamically
  - int* p = (int*) malloc(n * sizeof(int));
  - An array of size n
- free: De-allocate the memory
  - free(p);
- Make sure malloc and free are paired!

24

In-Class Exercise 5-2

Modify the `max_min( )` function (Exercise 5-1) so that it uses a pointer instead of an integer to keep track of the current position in the array. That is, instead of

```
int i;
for (i=0; i<9; i++)…
```

you ought write

```
int *pi;
…
```

25

Strings

- There is no string type in C!
- Instead, strings are implemented as arrays of characters:
  - `char*` or `char []`
- Enclosed in double-quotes
  - Terminated by NULL character (`'\0'`)
  - `"Hello"`
- printf format: %s
- same as
  - `char str[] = {'H', 'e', 'l', 'l', 'o', '\0'}`

26

Strings

- <string.h> has functions for manipulating null-terminated strings, e.g.
  - `int strlen(char* s):` returns length of s
  - `char* strcpy(char* s1, char* s2):` copies s2 into s1 (Check if s1 has enough space.)
  - `int strcmp(char* s1, char* s2)`: compares s1 and s2 (return 0 if they are the same, !=0 if they are different)
  - `char* strcat(char* s1, char* s2)`: appends s2 to s1 (Check if s1 has enough space.)
- Your assignments will implement these four basic string functions (of course, without using the <string.h> library).

27

Example – strcmp()

```
#include <stdio.h>
#include <string.h>
int main()
{
   char szKey[] = "apple";
   char szInput[80];
   for (;;)
   {
       printf("Guess my favorite fruit?\n");
       scanf("%s", szInput);
       if (strcmp (szKey,szInput) == 0) break;
   }
   printf("Correct answer!");
   return 0;
}
```

28

Example – strcpy()

```
#include <stdio.h>
#include <string.h>

int main ()
{
   char str1[]="Sample string";
   char str2[40];
   char str3[40];
   strcpy (str2,str1);
   strcpy (str3,"copy successful");
   printf ("str1: %s\nstr2: %s\nstr3: %s\n",
           str1,str2,str3);
   return 0;
}
```

29

Example – strcat()

```
#include <stdio.h>
#include <string.h>

int main ()
{
   char str[80];
   strcpy (str,"these ");
   strcat (str,"strings ");
   strcat (str,"are ");
   strcat (str,"concatenated.");
   printf("%s", str);
   return 0;
}
```

30

Example

```
int strlen(char* s)
{
      int n = 0;
      while (*s != '\0')
      {
            s++;
            n++;
      }
      return n;
}

char* p = "hello, world";
strlen(p);
strlen(p + 7);
```

31