Lecture05: Arrays & Recursion
10/5/2012

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

1

Slow Down A Bit - Review

- Basic Types: {`short, int, long, long long, float, double, long double, char`}
- Function: `{ double area(double radius)`

```
{
            return(3.1415 * radius * radius);
};
```

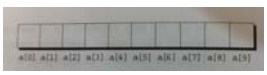
- Loop: {`for (i=1; i<=5; i++) …, while … }`
- Condition: {`if, else if, switch, case, … }`
- More depth: Recursion
- New: Arrays

2

Arrays

- Each variable can store only one value. What if you want a variable to store many values?
- To declare an array, use []:

```
int a[10]; // create an array with 10 integer elements
double b[5]; // create an array with 5 double elements
```



- Common to use a constant (MACRO) to define the length of an array

```
#define N 10
int a[N];
```

3

More on Arrays

- The size of an array can't be changed after declaration
- 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
```

- Sizeof()

```
char a[] = {'a', 'b', 'c'};
for (i=0; i<sizeof(a); i++)
    a[i] = 0;
```

4

Access Array elements

- 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);
}
```

5

More Array Examples

- What do the following code do?

```
for (i = 0; i < N; i++)
       a[i] = 0;

for (i = 0; i < N; i++)
       scanf("%d", &a[i]);

for (i = 0; i < N; i++)
       sum += a[i];

int a[10], i;
for (i = 0; i <= 10; i++)
       a[i] = 0;
```

6

In-Class Exercise 4-1

Write a program that asks the user to enter 5 integers (store these 5 integers in an array), then writes the numbers in reverse order:

```
Enter 5 integers: 34 82 49 102 7
In reverse order: 7 102 49 82 34
```

7

In-Class Exercise 4-2

Check whether any of the digits in a number appear more than once. Print out the repeated numbers. Use an array, called `digit_seen[10]`, initialized to 0. If seeing 2, set `digit_seen[2]` to 1.

```
Enter an integer: 282128
Repeated digit: 28

Enter an integer: 12345
No repeated digit
```

8

Multidimensional Arrays

```
int m[2][3] =     {1, 1, 1, 0, 0, 0};
int m[2][3] =     {{1, 1, 1}, {0, 0, 0}};


m[2][1] = 0;
```

9

Recursion

- A function is recursive if it calls itself.

```
int fact(int n)
{
   if (n <= 1)
   {
       return 1;
   }
   else
   {
       return n * fact(n -
   1);
   }
}
```

```
i = fact(3);
```

fact(3) calls fact(2)
  fact(2) calls fact(1)
    fact(1) returns 1 to fact(2)
  fact(2) returns 2*1 = 2 to fact(3)
fact(3) returns 3*2=6.

```
fact = 0;
for (i=n; i>=1; i++)
     fact = fact*i;
```

10

Why Recursion?

- It comes naturally with a common CS problem solving approach called **Divide-and-Conquer**.
  - What is divide-and-conquer?
  - Divide a large (complex) problem into small (simple) subproblems ; then solve the subproblems.
- Classical example: sorting

11

Sorting algorithm

- Enter 10 numbers into an array. Sort the array in an increasing order and print the array.

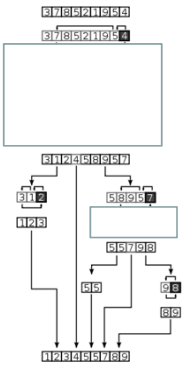
```
Enter no more than 10 numbers to be sorted: 9 16 47 82 4
66 12 3 25 51
In sorted order: 3 4 9 12 16 25 47 51 66 82
```

- How to solve this problem using divide and conquer? Hint: by dividing a large list into smaller sublists.

12

Quicksort algorithm

- Pick an element (i.e., the last one), called a pivot (基準), from the list
- Reorder the list such that elements with values < pivot comes before the pivot, values >= pivot comes after the pivot
  - Divide: two (unordered) sub-lists: a "lesser" sublist, a "greater" sublist
- Sort the two sublists (**recursively**) until the sublists have length <= 1



13

Quicksort code

pass an array into a function can change its value (will explain this later)

```
int split(int a[], int low, int
   high);  // return the
   position of the pivot after
   the split

void quicksort(int a[], int
   low, int high)
{
    int pivot;
    if ((high-low+1) <= 1)
        return;

    pivot = split(a, low,
   high);
    quicksort(a, low, pivot-1);
    quicksort(a, pivot+1,
   high);
}
```

```
int main()
{
   ….// read unsorted numbers
   into the array a
   quicksort(a, 0, n-1);
   … // print sorted numbers
   from array a
}

a[0..9]
quicksort(a, 0, 9) calls
   split() -> a[0..2] a[3]
   a[4..9]
   quicksort(a, 0, 2)
   quicksort(a, 4, 9)
```

14

Homework 4-1

Implement the Quicksort algorithm by completing the `split()` function as well as other code for input and output.

- (Step 1) Implement & **test** the input & output code, e.g., to read numbers into an array and print numbers from an array.
- (Step 2) Implement & **test** the `split()` function (without the sorting part)
- (Step 3) Fill in the quicksort() code and test

```
How many numbers? 7
Input 5 numbers: 12 3 6 18 7 15 10
In sorted order: 3 6 7 10 12 15 18
```

15

Other Sorting Algorithm

- Other sorting approaches?

16