Day11 A

Young W. Lim

2017-10-24 Tue

Outline

Based on

"C How to Program",
Paul Deitel and Harvey Deitel

I, the copyright holder of this work, hereby publish it under the following licenses: GNU head Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled GNU Free Documentation License.

CC BY SA This file is licensed under the Creative Commons Attribution ShareAlike 3.0 Unported License. In short: you are free to share and make derivative works of the file under the conditions that you appropriately attribute it, and that you distribute it only under a license compatible with this one.

Passing an array to functions

- to pass an array argument to a function
  - the name of an array without brackets
  - the size of an array
- always to pass by reference
  - the start address of an array is passed by reference
  - the called function can modify the elements of the caller's array
  - use `const`, if the prevention of this modification is desired

Receiving an array through a function call

- in the parameter list of a function definition
  - specify the array name with empty brackets
    - the number inside the brackets represents the array size
    - the arrary size is ignored, for 1-d dimensional arrays
    - therefore, any non-negative number is ok
    - or just empty barckets
  - specify the size of an array as a separate parameter

Passing an individual element of an array

- like an ordinary variable, an individual array element can
  - passed by value : `a[1]`, `a[i]`
  - passed by reference : `&a[1]`, `&a[i]`, `(a+1)`, `(a+i)`
- such simple pieces of data are called scalars
- use the subscripted name of the array element as an argument

Receiving an individual element of an array

- like an ordinary variable, an individual array element can be
  - received by value : `int a1`, `int ai`
  - received by reference : `int *p1`, `int *pi`
- do not need the subscripted name in the parameter list

2-dimensional arrays

- representing tables of values
  arranged in rows and columns

- to identify an element of a table
  two subscripts are needed
  (row subscript, col subscript)
- double scripted arrays

2-dimensional array initialization

- the intializer values are grouped by row in braces { }
- when not enough initializers for a given row
  the remaining elements of that row are intialized with zero

```
{ row_0, row_1, ..., row_m }

{ {a_00, a_01, ..., a_0n},
  {a_10, a_11, ..., a_1n},
  {a_00, a_01, ..., a_0n},
  {a_m0, a_m1, ..., a_mn}  }   (m+1)x(n+1)
```

2-dimensional array initialization examples

```
#include <stdio.h>

int main(void) {
  int a[3][4] = { {1, 2, 3, 4},
                  {5, 6, 7, 8},
                  {9,10,11,12} };
  int i, j;

  for (i=0; i<3; ++i) {
    for (j=0; j<4; ++j)
      printf("%3d ", a[i][j]);
    printf("\n");
  }
}
```

```
----
  1   2   3   4
  5   6   7   8
  9  10  11  12
```

```
#include <stdio.h>

int main(void) {

  int b[3][4] = { {1, 2, 3, 4},
                  {5, 6} };

  int i, j;

  for (i=0; i<3; ++i) {
    for (j=0; j<4; ++j)
      printf("%3d ", b[i][j]);
    printf("\n");
  }
}
```

```
----
  1   2   3   4
  5   6   0   0
  0   0   0   0
```

Row major order

- all array elements are stored consecutively in memory
  regardless of the number of subscripts
- 2-dimensional array,
  the first row is stored in memory
  then the second row follows the first row in memory

Passing a muti-dimensional array

- a multi-dimensional array in a parameter list
  - each script represents the size of a corresponding dimension
  - the first subscript size is not required
  - all subsequent subscript sizes are required
  - to identify a memory location of an element
    the size informations for each dimension are necessary
    except for the 1st dimension
  - `a[][M][N]` in a parameter list

Accessing a muti-dimensional array

- `a[][M][N]` in a parameter list
- `a[i][j][k] = *a( ( (i*M+j)*N + k ) )` in accessing an element
- each row can be viewed as a 2-dimensional array
  p = a[i];

a[i]	a[i][j][k]
p	p[j][k]

The type size_t

- unsigned integral type
  - `unsigned int` for one computer
  - `unsigned long` for another computer
- translation may be required
  - the code for one computer
  - the code for another computer
  - `size_t` provides portability of a code
- defined in `<stddef.h>`
  which is often included by `<stdio.h>`

- `size_t` is recommended for any variable
  that represents an array's size or an array's subscripts

sizeof Operator

- unary operator sizeof determines the size in bytes
  of a variable or a type at compile time
- When applied to the name of an array,
  sizeof returns the total number of bytes in the array
- The type size_t is an integral type
  - unsigned int
  - unsigned long int
  - returned by operator sizeof
  - defined in <stddef.h>
- Operator sizeof can be applied to any variable name, type, or value
- The parenthesis used with sizeof required
  if a type name is supplied as its operand

Sorting algorithm

- placing the data into a particular order
  - ascending order
  - descending order
- various sorting algorithms
  - bubble sort

Bubble sorting algorithm

- the smaller values gradually "bubble" their way upward to the top
- the larger values gradually sink down to the bottom

- several passes over the array
- on each pass, successive pairs of elements are compared
  - if a pair is in non-decreasing order, no action
  - otherwise, swap the elements of the pair
  - a small value can be moved up by only one position
  - a large value can be moved down by many positions

Search algorithm

- the processing of finding a particular element in an array
- such a particular element to be found : a search key
- find the location (subsript) of a search key in the searched array

- linear search algorithm
- binary search algorithm

Linear search algorithm

1. when the given array is unsorted
   - only linear search can be applied
   - on average, the program will have to compare the search key with the half the elements

1. when the given array is sorted
   - linear search still can be used for a small size array
   - binary search is more efficient for a large size array

Binary search algorithm (1)

- eliminates from consideration one-half of the elements in the sorted array whenever comparison is made
- comparision is made only with the middle element of the sorted array
- recursively reduce the given problem by half
- repeat until
  - the search key is equal to the middle element
  - or the reduced problem contains only one element which is not equal to the search key

Binary search algorithm (2)

- locates the middle element of the sorted array (increasing order)
- compare this middle element with the search key
  - if equal, the search key is found
  - if not equal, the problem is reduced to searh only the half
    - when the seach key is less than the middle element
      search only the first half of the array
    - when the seach key is greater than the middle element
      search only the second half of the array