Computer Programming

Sino-European Institute of Aviation Engineering

Module 5 Array

Outline

- Introduction
- One Dimensional Array
- Array Definition and Accessing
- Passing 1-D Array to a Function
- Sorting Array
- Two Dimensional Array
- String operation

Introduction

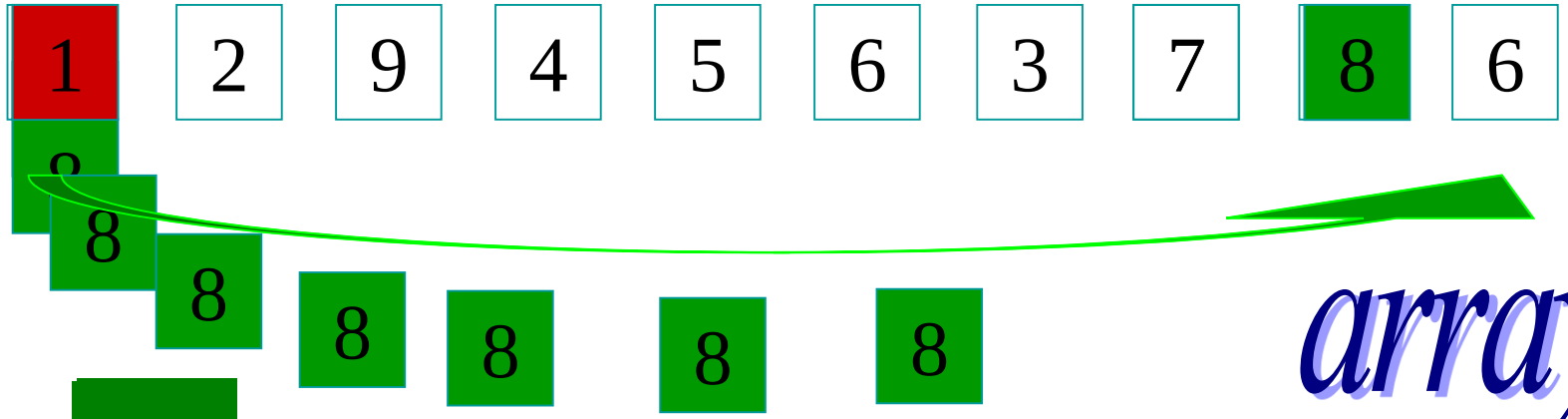
Question

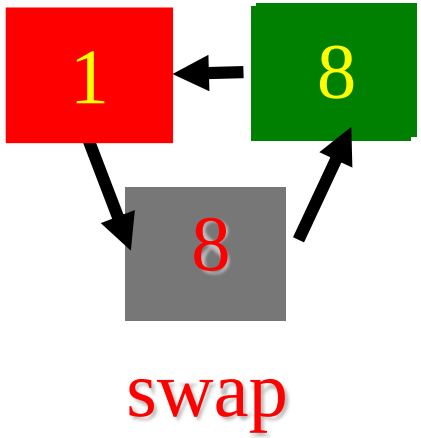
- **How to deal with scores of a class?**
- **How to store a string?**
- **How to describe a matrix?**

- ***We have learned basic data types (int, char, float and double), but all cannot solve the above problems.***
- ***C language defines a new data type: array to describe above data.***

Introduction

How to make a series of integers in increasing order?





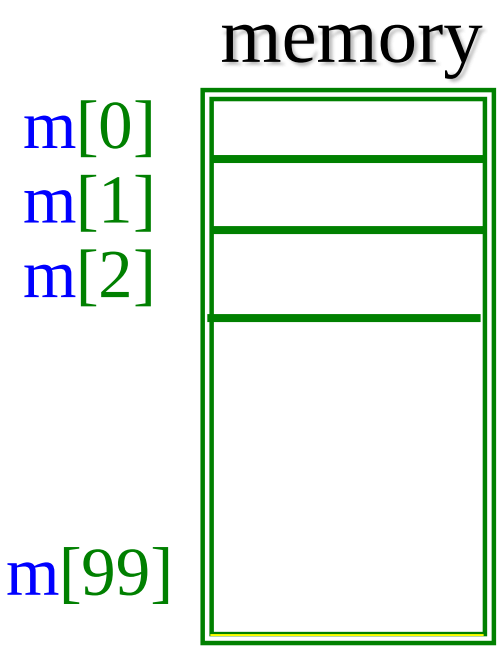
- How to store these integers?
- After sorting, How to store them keeping in order?

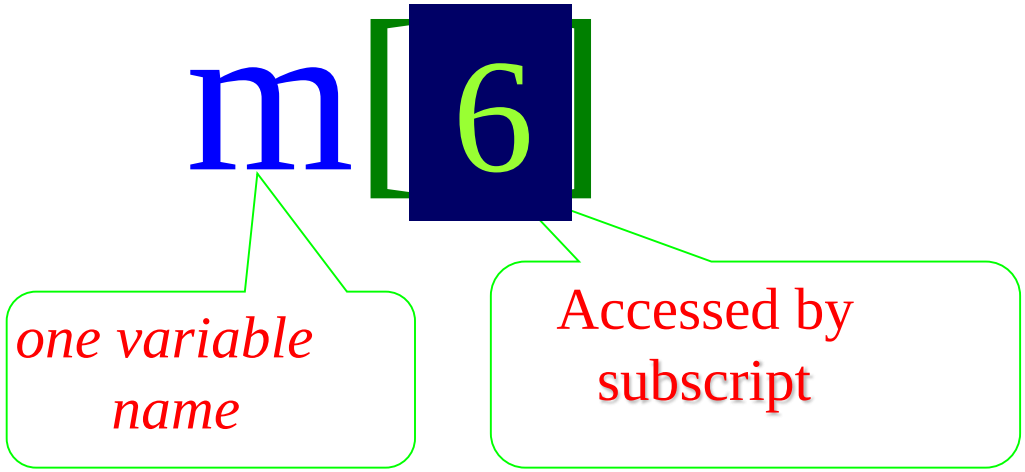
Introduction

- Arrays
  - Structures of related data items
  - Static entity – same size throughout program
  - Dynamic data structures discussed in other chapter

Introduction

Consecutive storage, same type, subscript control





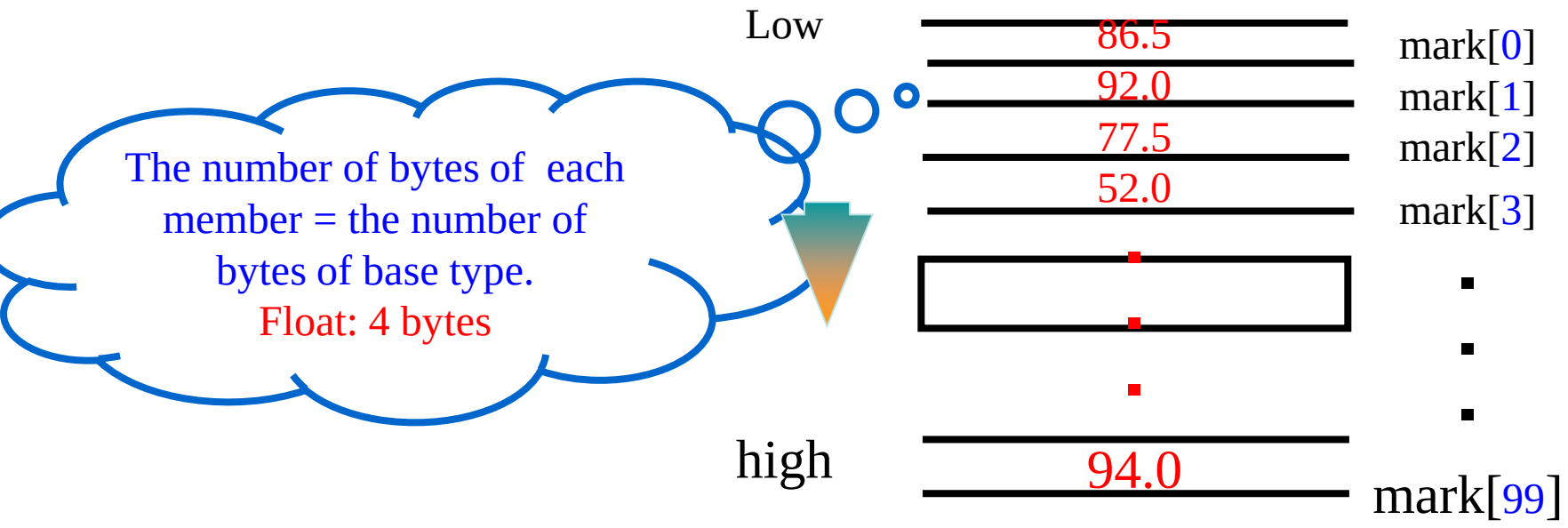
application: string processing

One Dimensional Array

- Array
  - Group of consecutive memory locations
  - A block of many variables of the same type、 Same name
  - can be declared for any type
- To refer to an element, specify
  - Array name
  - Position number
- Format: arrayname[position number]
  - First element at position 0
  - n element array named c:c[0], c[1]...c[n – 1]

One Dimensional Array

- float mark[100];



Array Definition and Accessing

Array

- The same type
- Limited numbers
- Store in sequence

To group several data with the same type together

Array Definition and Accessing

- Declaring an array

data type array-name [size]

How many members in array. Size is constants.

All array members are the same type

Example:

```
int a [ 10 ];
float score[20];
char str[10];
int matrix[10][10];
```

array dimension

Array can have more than 1 dimension.

Array Definition and Accessing

- Several typical errors in array declaration
  - float a[0]; /*Size of array can/t be 0*/
  - int b(2)(3); /* can use ()*/
  - int k, a[k]; Size of array must be constant
  - int n;

    scanf (”%d”, &n);

    int a[n];

Array Definition and Accessing

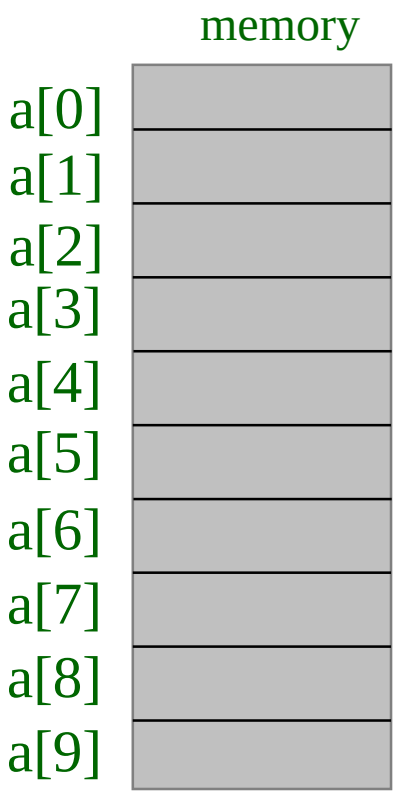
- Accessing array members

from 0 to size-1

array-name [subscript]

Example: int a [10];

a[0], a[1], a[2]… a[9]



Notes:

An array element is equivalent to an ordinary variable functionally.

Array Definition and Accessing

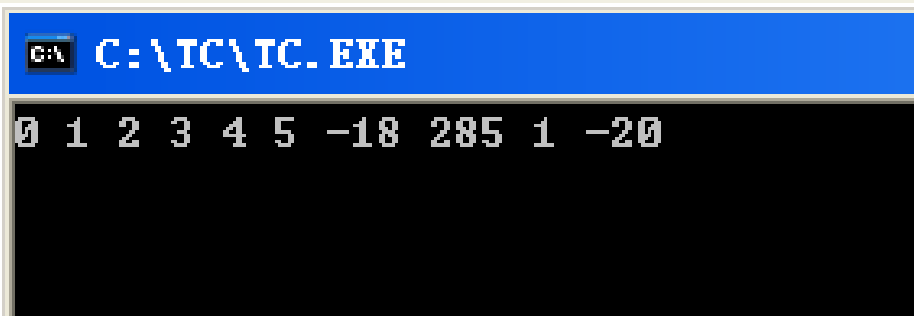
- A single array element *a[i]* is accessed when *i* has a value greater than or equal to 0 and less than or equal to *N – 1* .
- If *i* has a value outside this range, a run-time error will occur when *a[i]* is accessed.
- Overrunning the bounds of an array is a common programming error.

Array Definition and Accessing

- In c, there is no check to see if the subscript used for an array exceeds the size of the array.
- Example for bounds checking

```
#include <stdio.h>
  void main()
  {     int num[6],i;
        for(i=0;i<6;i++)
                num[i]=i;
        for(i=0;i<10;i++)
                printf("%d ",num[i]);
  }
```

What is the output ?



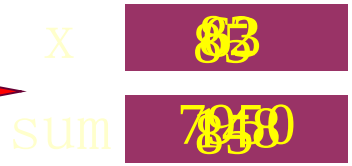
Array Definition and Accessing

- Each member of the array is a simple variable

  Example: Enter the 100 student's score， then calculate the total score.

Variable

```
for (i=0; i<100; i++)
 { scanf("%f\n", &x);
   sum += x;
 }
```

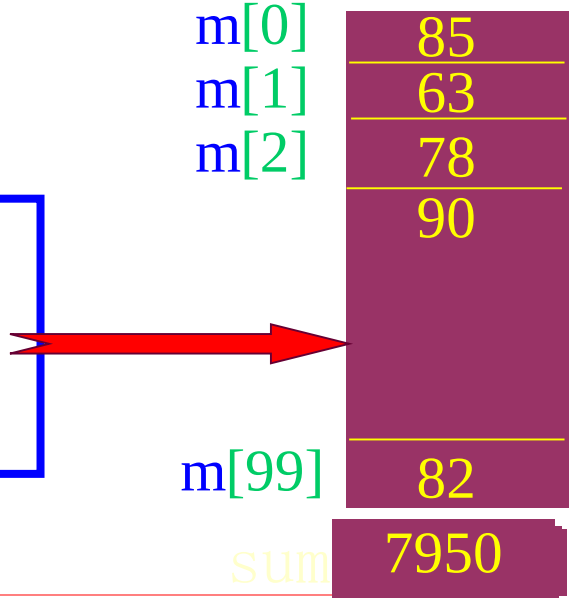


Array

```
float m[100],sum=0;
for ( i=0; i<100; i++ )
{ scanf("%f\n", &m[i]);
  sum += m[i];
}
```

Reusable

Ordered



Array Definition and Accessing

- Initialize array
  - General Method

```
int a[10]={1,2,3,4,5,6,7,8,9,10};
```

  - Initialize part of elements

```
float x[5] = { 1.9, 2.0 }; // other elements be assigned 0.
```

  - Define the array length according to the data

```
int b[ ] = {1, 2, 3, 4};// 4 elements in array b
```

  - Input data dynamically

```
int i;
for(i=0;i<10;i++)
    scanf("%d",&a[i]);
```

Array Definition and Accessing

- Array name Vs Array member
  - Array name is the address of first array element. It is a address constant.
  - The element of array is numerical value.

2000_H	86.5	mark[0]
2004_H	92.0	mark[1]
2008_H	77.5	mark[2]
$200C_H$	52.0	mark[3]
	▪	.
.	▪	.
.	▪	.
.		
$218C_H$	94.0	mark[99]

mark[2]:
(1)Calculate mark+ 2*4
=2000+2*4
(2)Obtain the value of 2008

We can access the array elment according to it's address
mark [2] or *(mark+2)

Array Definition and Accessing

- Non Character Array
  - Array I/O： int x[10];

Right： for (k=0; k<10; k++)
```
        scanf (”%d”, &x[k]);
    for (k=0; k<=9; ++k)
        printf (”%d\n”, x[k]);
```

Wrong：
```
scanf (”%d%d%d%d%d%d%d%d%d%d”, x);
scanf (”%d”, x);
printf (”%d”, x);
printf (”%d%d%d%d%d%d%d%d%d%d”, x);
```

- Character Array

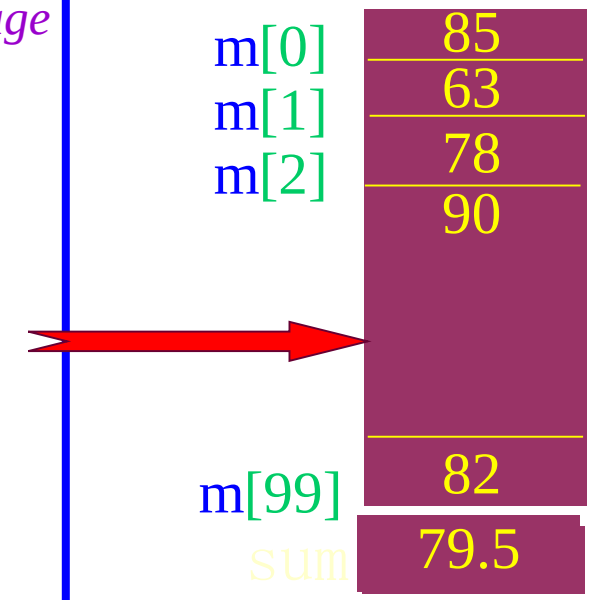
```
scanf(”%s”, string);
printf (”%s”, string);
```

Array Definition and Accessing

Example: Enter the 100 student's score for Data Structure and Algorithm course, then calculate the average score and display it.

```
#define STUNUM 100
void main{
    float score[STUNUM],sum=0,average;
    int studentNum = STUNUM;
     for ( i=0; i< studentNum; i++ )
     { scanf("%f", &score[i]);
       sum += score[i];
     }
     average = sum/ studentNum ;
     printf("The average score for Data
            Structure and Algorithm course is
           %f.\n", average);
}
```

Array storage



Array Definition and Accessing

Example : Calculate the value of the first thirty items (N=30) of Fibonacci list.

$a_1=a_2=1$ $a_n=a_{n-1}+a_{n-2}$ i.e.: 1,1,2,3,5,8,13

```
#define NUM 30
void main( ) {
    int fiboList[NUM];
    fiboList[0] = 1; fiboList[1] = 1;
    int i;
    for (i=2; i<NUM; ++i)
        fiboList [i] = fiboList [i-1]+fiboList [i-2];
    for (i=0; i<NUM; i++)
        printf("%d\t", fiboList[i]);
}
```

Array Definition and Accessing

- #define directive
  - Magic Numbers in the program convey little information to the reader
  - Hard to change in a systematic way
  - #define defines a symbolic name
  - During preprocessing phase, symbolic names are replaced by the replacement text

Array Definition and Accessing

- Find Minimum value with #define

```
#include <stdio.h>

#define ARRAY_SIZE 10
void main()
{
    int i, min, array[ARRAY_SIZE];

    printf("please enter %d numbers:\n", ARRAY_SIZE);
    for(i = 0; i < ARRAY_SIZE; ++i)
      scanf("%d", &array[i]);
    min = array[0];
    for(i = 1; i < ARRAY_SIZE; ++i) {
        if (array[i] < min)
            min = array[i];
    }
    printf("the minimum is: %d\n", min);
    return 0;
}
```

Use capital letters

Array Definition and Accessing

- Linear search
  - Simple
  - Compare each element of array with key value
  - Useful for small and unsorted arrays

Array Definition and Accessing

- Binary search
  - For sorted arrays
  - Compares middle element with key
    - If equal, match found
    - If key < middle, looks in first half of array
    - If key > middle, looks in last half
    - Repeat
  - Very fast; at most n steps, where 2^n > number of elements
    - 30 element array takes at most 5 steps

      $2^5 > 30$ so at most 5 steps

Passing 1-D Array to a Function

- Function prototype

```
void modifyArray( int b[], int arraySize );
```

  - Parameter names optional in prototype
  - int b[] could be written int []
  - int arraySize could be simply int

Passing 1-D Array to a Function

- Passing arrays
  - To pass an array argument to a function, specify the name of the array without any brackets
    - int myArray[24]; myFunction(myArray, 24);
    - Array size usually passed to function
  - Arrays passed call-by-reference
  - Name of array is address of first element
  - Function knows where the array is stored
- Passing array elements
  - Passed by call-by-value
  - Pass subscripted name (i.e., myArray[3]) to function

Passing 1-D Array to a Function

Example of passing array elements to a function.

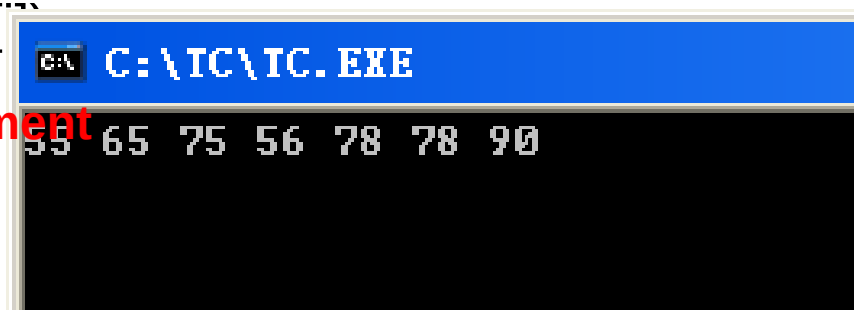
```
#include <stdio.h>
void twice_element(int);
void main()
{
    int i;
    int marks[]={55,65,75,56,78,78,90};
    for(i=0;i<=6;i++)
        twice_element (marks[i]);
    for(i=0;i<=6;i++) printf("%d ",marks[i]);

}
void twice_element(int m)
{
    m = m*2;
}
```

Use marks[i] as argument

Use m as parameter

What is the output ? Does the value of element change?



Passing 1-D Array to a Function

Example of passing array elements to a function.

```
#include <stdio.h>
int get_sum(int a[],int n);
void main()
{
    int i,sum;
    int marks[]={55,65,75,56,78,78,90};
    sum  =      get_sum (marks,  6);
    printf("the sum is %d ",sum);
}
int get_sum(int a[],int n)
{   int i,sum=0;
    for(i=0;i<=n;i++) sum = sum+a[i];
    return sum;
}
```

Passing the name of the array

The size of the array

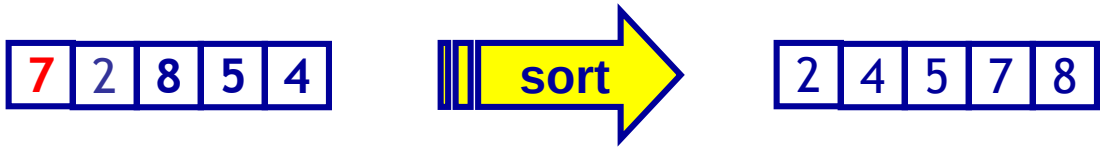
Just like marks[i]

Sorting Array

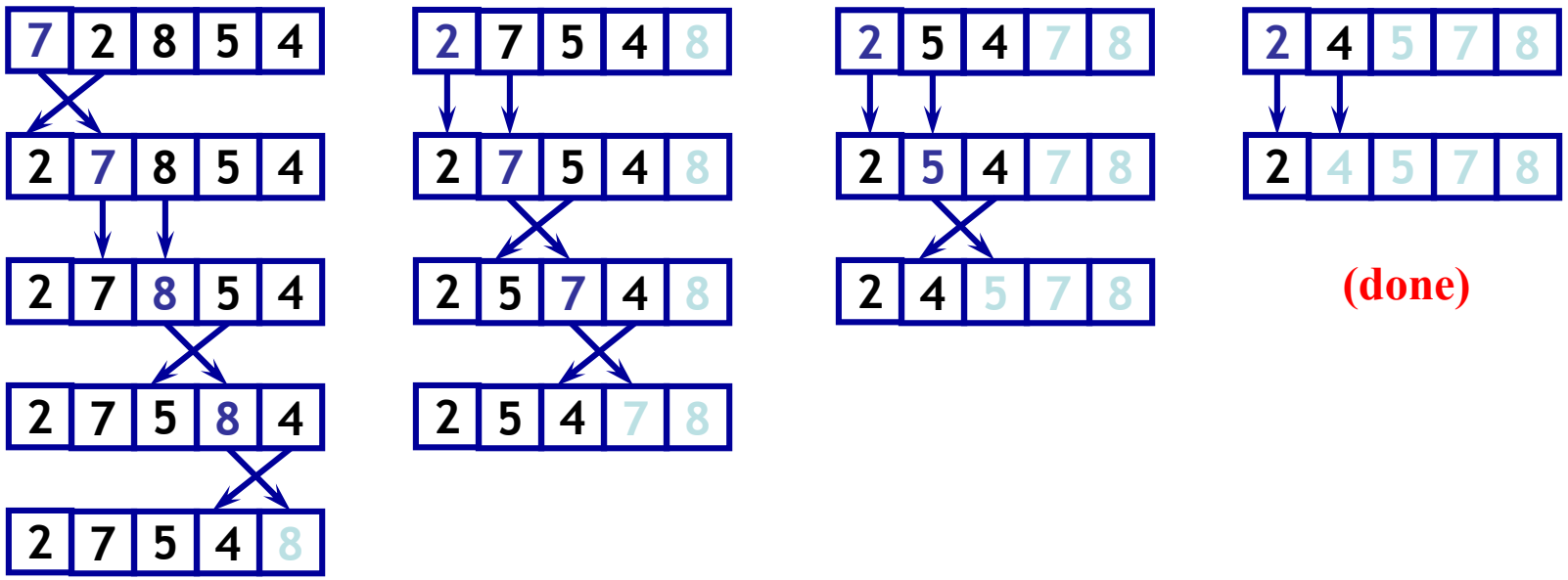
- Sorting data
  - Important computing application
  - Virtually every organization must sort some data
- Bubble sort (sinking sort)
  - Several passes through the array
  - Successive pairs of elements are compared
    - If increasing order (or identical), no change
    - If decreasing order, elements exchanged
  - Repeat

Sorting Array

- Bubble sort
  - We would like to sort the elements in an array in an ascending order



Sorting Array



Sorting Array

- Bubble sort

```
void sort(int a[], int size)
{   int i, j, temp;

    for (i = size - 1; i >= 0; --i)  /* counting down      */
    {   for (j = 0; j < i; ++j)        /* bubbling up          */
        {  if (a[j] > a[j+1])           /* if out of order...   */
           {                                  /* ... then swap        */
              temp = a[j];
              a[j] = a[j+1];
              a[j+1] = temp;
           }
        }
    }
}
```

Sorting Array

- Using bubble sort

```
#include <stdio.h>
#define ARRAY_SIZE 5
void sort(int a[], int size);

int main()
{  int array[ARRAY_SIZE] = {7, 2, 8, 5, 4};
    int i = 0;

    sort(array, ARRAY_SIZE);

   for (i = 0; i < ARRAY_SIZE; ++i) /* print the sorted array */
        printf("%d ", array[i]);
    return 0;
}
```

Two Dimensional Array

- The C language allows arrays of any type, including arrays of arrays.
- With two bracket pairs, we obtain a two-dimensional array. This idea can be iterated to obtain arrays of higher dimension.

Declarations of arrays	Remarks
int a[100];	A one-dimensional array
int b[2][7];	A two-dimensional array
int c[5][3][2];	A three-dimensional array

Two Dimensional Array

- Declaring a two dimensional array

Data type array-name [size1] [size2]

Example: int matrix[3][3];

1	2	3
4	5	6
7	8	9

Two Dimensional Array

- A two-dimensional array can be considered as a one-dimensional array with its element of another one-dimensional array.
- The construct process of a two-dimensional array is shown as below:

$$
\mathbf{a}\left\{
\begin{array}{lcl}
\mathbf{a[0]} & \dashrightarrow & \mathbf{a_{00}\ \ a_{01}\ \ a_{02}\ \ a_{03}} \\
\mathbf{a[1]} & \dashrightarrow & \mathbf{a_{10}\ \ a_{11}\ \ a_{12}\ \ a_{13}} \\
\mathbf{a[2]} & \dashrightarrow & \mathbf{a_{20}\ \ a_{21}\ \ a_{22}\ \ a_{23}}
\end{array}
\right.
$$

Two Dimensional Array

- We can think of a[i][j] as the element in the *i*th row, *j*th column of the array (counting from 0).

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Column subscript

Array name

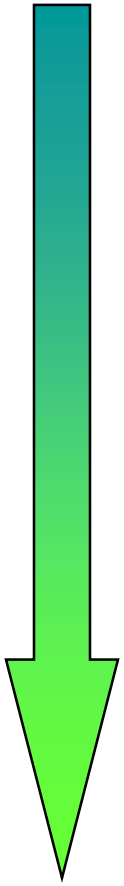
Row subscript

Two Dimensional Array

- **Multi-D Array**

int $c[a_1][a_2]...[a_n]$

Low



c[0][0]...[0][0]
c[0][0]...[0][1]
.....
c[0][0]...[0][a_n-1]
c[0][0]...[1][0]
c[0][0]...[1][1]
.....
c[0][0]...[1][a_n-1]
.....
c[0][0]...[a_{n-1}-1][a_n-1]
.....
c[a_1-1][a_2-1]...[a_{n-1}-1][a_n-1]

High

Two Dimensional Array

- Initialize array
  - General Method

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

  - Initialize part of elements

```
int a[2][3]={{1},{2,3}}; // other elements be assigned 0
int a[2][3]={1,2,3}; // other elements be assigned 0
int a[2][3]={{0},{1,2,3}}; // other elements be assigned 0
```

  - Input data dynamically

```
for ( i=0 ; i< 2 ; i++ )
for ( j=0 ; j< 3 ; j++ )
   scanf ( "%d",&a[i][j] ) ;
```

Two Dimensional Array

Address of 2-D Array

- the name of 2-D array a represent its first address
- a[i] represent the address of first elment in the *ith* row
- a+i equal to a[i]

  a[i] ⇔ a+i ⇔ &a[i][0]
- a[i]+j ⇔ &a[i][j]

a

a	a[0]	2	5	9
a+1	a[1]	3	2	5
a+2	a[2]	4	7	1

Two Dimensional Array

- Array Acessing

Row, from 0 to size1-1

Column, from 0 to size2-1

array-name [subscript1] [subscript2]

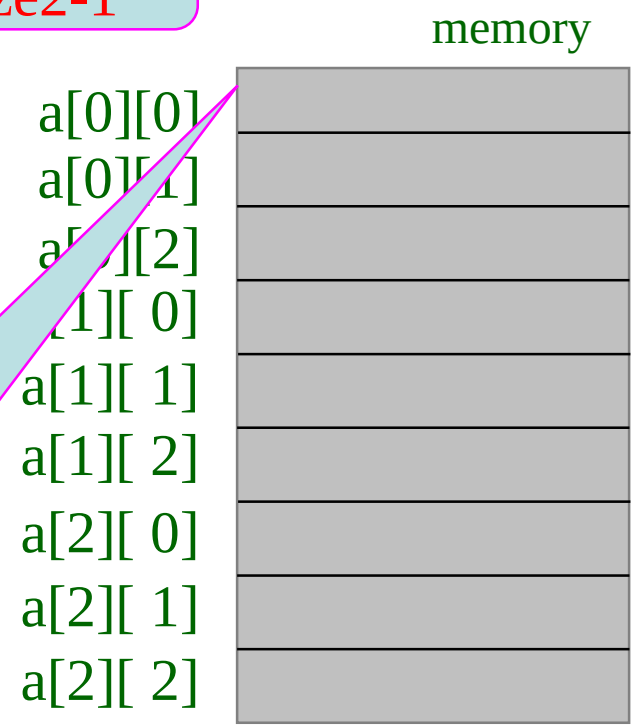
Example: int a [3][3];

a[0][0], a[0][1], a[0][2]

…

a[2][0], a[2][1], a[2][2]

Storage row by row



Two Dimensional Array

- Array Acessing

```
printf(“ %d \n”, *(a[i]+j));
printf(“ %d \n”,  a[i][j]);
```

*(a[i]+j)

First address of *ith* row

Two Dimensional Array

- Display the following matrix

$$A=\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 2 & 1 & 1 & 1 & 1 \\ 3 & 2 & 1 & 1 & 1 \\ 4 & 3 & 2 & 1 & 1 \\ 5 & 4 & 3 & 2 & 1 \end{bmatrix}$$

3−2+1

Analysis:

- **Use two dimensional array to store matrix data.**
- **The rules of each element value:**
  - ***Row-subscript* <= *column-subscript*:**
    value=1;
  - ***Row-subscript* > *column-subscript*:**
    value= row-subscript – column-subscript +1

Two Dimensional Array

```
#include <stdio.h>
void main( ) {
     int i, j, a[5][5];
    for (i=0; i<=4; i++)        /* i  for row subscript */
    {
            for (j=0; j<=4; j++)      /* j for column subscript */
            {
                  if  ( i<=j ) a[i][j]=1; /* create matrix */
                   else   a[i][j]=i-j+1;
             }
     }
    for (i=0; i<=4; i++) /* display matrix */
    {
            for (j=0; j<=4; j++)
                 printf("%d ", a[i][j]);
            printf("\n");
    }
}
```

Two Dimensional Array

- Get the reverse matrix (B) of matrix (A)

Matrix A(2×3), Reverse Matrix B(3×2)

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

```
#include <stdio.h>
void main( )
{  int i, j, b[3][2], a[2][3] = { {1,2,3}, {4,5,6} };

    for (i=0; i<=1; i++)
        for (j=0; j<=2; j++)
            b[j][i] = a[i][j];     /* reversing matrix */
    for (i=0; i<=2; i++) {
        for (j=0; j<=1; j++)
            printf("%d ", b[i][j]);
            printf("\n");
    }
}
```

Algorithm for reversing matrix:

b[j][i] = a[i][j]

Two Dimensional Array

- Yang Hui Triangle

	0	1	2	3	4	5	6	7	8	9
0	1									
1	1	1								
2	1	2	1							
3	1	3	3	1						
4	1	4	6	4	1					
5	1	5	10	10	5	1				
6	1	6	15	20	15	6	1			
7	1	7	21	35	35	21	7	1		
8	1	8	28	56	70	56	28	8	1	
9	1	9	36	84	126	126	84	36	9	1

Two Dimensional Array

- Yang Hui Triangle

```
#define SIZE 10
    main()
{  int i,j;   int a[SIZE][SIZE];

        printf("\n");
    for (i=0; i<SIZE; i++)
    {  a[i][0]=1;   a[i][i]=1;  }

        for (i=2; i<SIZE; i++)
      for (j=1;j<i;j++)
          a[i][j]=a[i-1][j-1]+a[i-1][j];
    for (i=0;i<SIZE;i++)
      {  for (j=0;j<=i;j++)
            printf("%5d",a[i][j]);
          printf("\n");
      }
}
```

Two Dimensional Array

```
#include <stdio.h>
int get_sum(int a[][4],int n, int m);
void main()
{    int i,sum;
     int a[3][4]={1,2,3,4,5,6,7,8,9,0,1,6};
     sum = get_sum(a,  3,  4);
     printf("the sum is %d",sum);

}
int get_sum(int arr[][4],int row, int col)
{    int i,j,sum=0;
     for(i=0;i<row;i++)
          for(j=0;j<col;j++)
               sum = sum+arr[i][j];
     return sum;
}
```

Passing the name of the array

Colums of this array

Rows of this array

Just like a[i][j]

String Operation

- Character arrays
  - We use character array to store string
  - String “first” is really a static array of characters
  - Character arrays can be initialized using string literals

```
char string1[] = "first";
```

    - Null character '\0' terminates strings
    - string1 actually has 6 elements

```
char string1[] = { 'f', 'i', 'r', 's', 't', '\0' };
```

String Operation

- Character arrays
  - Can access individual characters
    - string1[3] is character ‘s’
  - Array name is address of array, so & not needed for scanf
    - scanf("%s", string2);
  - Reads characters until whitespace encountered
  - Can write beyond end of array, be careful

String Operation

- String Initializing
  - Using string

```
char ch[6]={ "CHINA " };
char ch[6]= " CHINA ";
char ch[ ] = "CHINA";
```

```
char str[20];
gets( str);
```

  - Using characters

```
char ch[6]={ 'C','H','I','N','A','\0'};
```

```
char str[20];
scanf("%s", str);
```

String Operation

String Operation

- String input/output
- Get string length
- String concatenation
- String copying
- String comparing
- String reversing
- Extracting parts of a string

String Operation

- String input/output
  - gets (str_name);
    - Input the string from keyboard, end with \n
  - puts (str_name);
    - Print the string (end with \0) to screen

```
void main()
{
        char str[20];
        gets(str);
        puts(str);
}
```

String Operation

- String Input/Output
  - By character (c%)

```
static char a[3];
for (i=0;i<3;i++)
   scanf("%c",&a[i]);
for (i=0;i<3;i++)
  printf("%c",a[i]);
printf("\n");
```

Input:
d
o
s
Output:
dos

String Operation

- String Input/Output
  - By string (s%)

```
static char a[7];
scanf("%s",a);
printf("%s\n",a);
```

- Input string can't contain ' ','\t' and '\n'.
- '\0' will be added automatically
- '\0' will not be printed

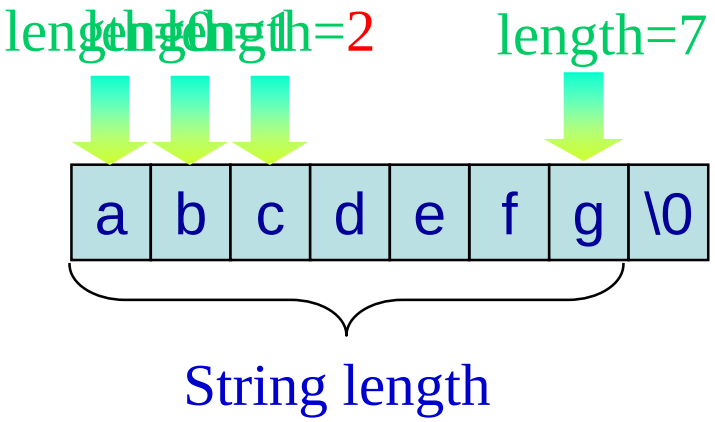
String Operation

- Get String Length

```
void main()
{
    int a;
    char x[20];
    scanf("%s",x);
    a=strlen(x);
    printf("%d\n",a);
}
```

String Operation

```
#include <stdio.h>
void main ( )
{    int length;
     char str[100];
     gets(str);
     length=0;
     while ( str[length] != ’ \0’  )
         length ++;
     printf (” String length = %d” , length);
}
```

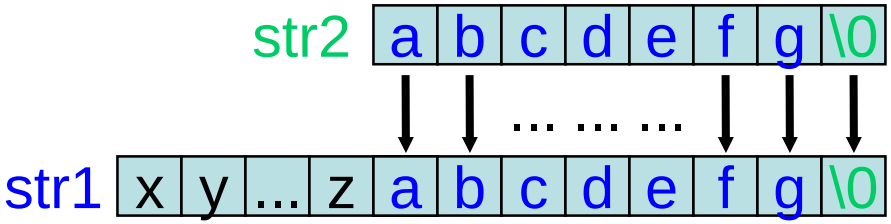


String Operation

String Concatenation

Format：strcat（ char_array$_1$， char_array$_2$ ）

Long enough



String Operation

String Concatenation

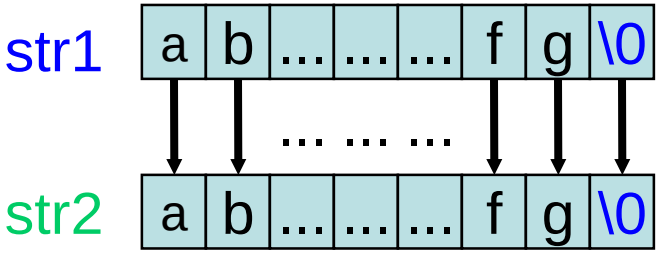
```
#include <stdio.h>
void main( )
{     char str1[100],str2[100];
      int i, j;
      printf (” Enter string 1:” );
      gets (str1);
      printf (” Enter string 2:” );
      gets (str2);
      for ( i=0; str1[i]!=’ \0’ ; i++ ) ;   /* find \0 in str1 */
      for ( j=0; (str1[i]=str2[j]) != ’ \0’ ; i++, j++ ) ;
      printf(” Output string 1:%s\n” , str1);
}
```

String Operation

String Copy

Format：strcpy（ $char\_array_1$， $char\_array_2$ ）

```
#include <stdio.h>
 void main ( )
 { char str1[100], str2[100];
   int i;
   printf ("Enter string 1:");
   gets (str1); /* input str1 */
   for (i=0;(str2[i]=str1[i])!='\0'; i++) ;
                                /* copy*/
   printf("Output string 2:%s\n", str2);
 }
```



String Operation

String Comparing

Format：strcmp（ $char\_array_1$， $char\_array_2$ ）

Return value:

- =0: $char\_array_1 = char\_array_2$
- >0: $char\_array_1 > char\_array_2$
- <0: $char\_array_1 < char\_array_2$

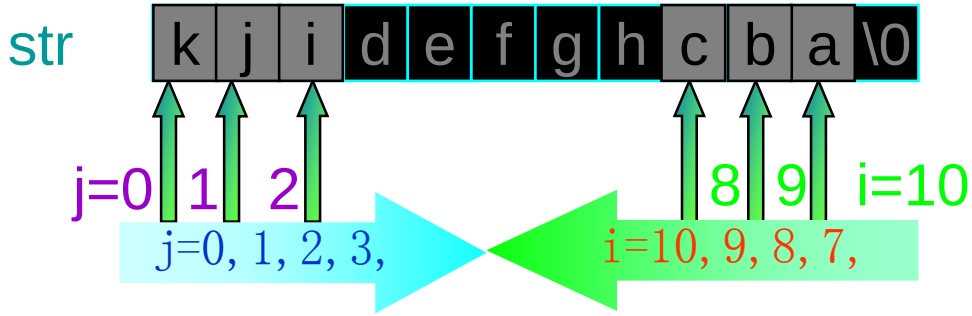
Accoding to the ASCII value:

”abc” = ”abc”
”abc” < ”b”
”abc” > ”abadef”
”abc” > ”ab”

“abc” = ”abc”
‘a’ < ‘b’
‘c’ > ‘a’
‘c’> ‘\0’

String Operation

- String Reversing



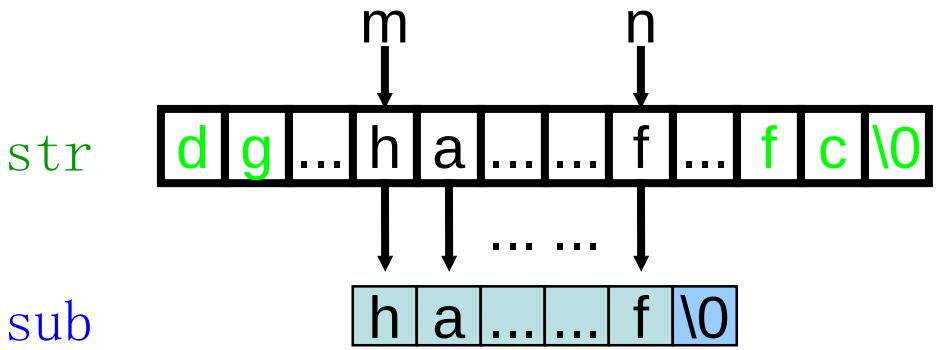

```
void main ( )
{
   char str[100], c;
   int i, j;
   printf (” Enter string:” );    gets (str);
   for ( i=0; str[i]!=’ \0’ ; i++ ) ;
   i--;
   for ( j=0;  j<i; i--, j++ )
      { c=str[i];  str[i]=str[j];  str[j]=c;  }
   printf(” Output string:%s\n” , str);
}
```

String Operation

Extracting parts of a string

- Assume the length of string is L,
- start position: m , extracting length: n

1. If m<L, m+n<=L — Normal extracting
2. m<L, m+n>L — Extracting from m(<n)
3. m>=L — Can't get the sub string



String Operation

```
#include <stdio.h>
void main( )
{   char str[100], sub[100];
    int m, n, k, l;
    printf (” Enter string:” ); gets(str);
    printf (” Enter m n:” ); scanf(” %d%d” , &m, &n);
    for ( k=0; k<m-1 && str[k]!=’ \0’ ; k++ ) ;
                    /* find start position*/
    for ( l=0; l<n && (sub[l]=str[k])!=’ \0’ ; k++,l++) ;
    sub[l]=’ \0’ ;
    printf (” sub=%s\n” , sub);
}
```

Summary

- Definition and accessing of array
- Passing array to a function
- How do elements of array store in the memory
- The relationship between string and 1-D char array
- String operation:
  - concatenation,
  - Copy
  - comparing

TERMS

- Array 数组
- Subscript 下标
- Overflow 溢出
- Dimension 维数
- Sort 排序
- Bubble Sort 冒泡排序
- Selection Sort 选择排序

Thank you!