C Programming

Lecture 6:

A	**r**	**r**	**a**	**y**	**_**	**a**	**n**	**d**	**_**	**S**	**t**	**r**	**i**	**n**	**g**	**s**	**\0**
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Lecturer: *Dr*. Wan-Lei Zhao

Autumn Semester 2022

Outline

Opening Discussion (1)

- Given we have following problem
  - We have 10 students in the class
  - We want to get average/sum/max/min score of their math course
  - We also want to rank the scores
  - Based on what we learned
  - We should keep 10 variables of the same type
- How about we have 100 students??

```c
#include <stdio.h>
void main()
{
    float x1, x2, x3, x4, x5, x6,x7,x8,x9,x10;
    float sum = 0, avg = 0;
    scanf("%f", &x1);
    sum += x1;
    scanf("%f", &x2);
    sum += x2;
    ...
    avg = sum/10;
}
```

Opening Discussion (2)

```c
#include <stdio.h>
void main()
{
    float x1, x2, x3, x4, x5, x6,x7,x8,x9,x10;
    float sum = 0, avg = 0;
    scanf("%f", &x1);
    sum += x1;
    scanf("%f", &x2);
    sum += x2;
    ...
    avg = sum/10;
}
```

- Even that it is hard to do sorting
  - Try your best to figure out how you can put fourty variables in order
- This is where the array comes

Outline

1D Array: declaration (1)

- 1D array is defined in following form

$$\text{type arrayName}[\textbf{size}];$$

- type could be any type defined in C, e.g. int, float,...
- "arrayName" should be unique
- It is actually a variable/constant, so rules to other variables/constants apply too
- "**size**" should be an integer or an integer constant greater than 0

```
int a[0]; //it is grammar OK, but meaningless
```

1D Array: declaration (2)

type arrayName[size];

- type could be any type defined in C, e.g. int, float,...
- "arrayName" should be unique
- It is actually a variable/constant, so rules to other variables/constants apply too
- "**size**" should be an integer or an integer constant greater than 0

```c
int main()
{
    float x[40];
    ....
    return 0;
}
```

```c
int main()
{
    const int N = 40;
    float x[N];
    ....
    return 0;
}
```

```c
int main()
{
    int N = 40;
    float x[N];
    ....
    return 0;
}
```

1D Array: declaration (3)

type arrayName[size];

- type could be any type defined in C, e.g. int, float,...
- "arrayName" should be unique
- It is actually a variable/constant, so rules to other variables/constants apply too
- "**size**" should be an integer or an integer constant greater than 0

```c
#define N 40
int main()
{
    float x[N];
    ....
    return 0;
}
```

```c
int main()
{
    const int N = 40;
    float x[N];
    ....
    return 0;
}
```

```c
#define N 40
int main()
{
    float x[3*N];
    ....
    return 0;
}
```

1D Array: visit array element (1)

- Element in an array is visited by the **subscript**
- Subscript starts from '0' to 'N-1'
- For example, visit the 3rd element of x[N], we write "x[2]"

```c
#include <stdio.h>
int main()
{
    float x[40];
    x[0] = 5.0;
    x[2] = 3.1;
    printf("x[0] = %f", x[0]);
    return 0;
}
```

1D Array: visit array element (2)

```c
int main()
{
    float x[40];
    int i = 0;
    for(i = 0; i < 40; i++)
    {
        printf("Input %d:", i);
        /*--be careful below--*/
        scanf("%f", &(x[i]));
    }
    return 0;
}
```

- You are not allowed to use subscript beyond 39
- You invade other's territory!!

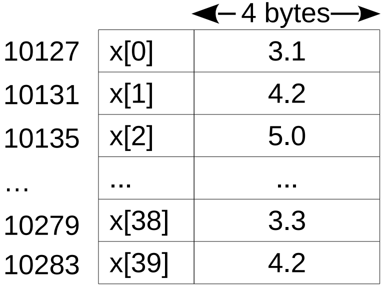
1D Array: how array looks like (1)

		← 4 bytes →
10127	x[0]	3.1
10131	x[1]	4.2
10135	x[2]	5.0
...	...	...
10279	x[38]	3.3
10283	x[39]	4.2

		← 1 byte →
10127	ch[0]	c
10128	ch[1]	b
10129	ch[2]	e
...	...	...
10165	ch[38]	f
10166	ch[39]	x

- The system opens a continuous memory block for an array
- Actual size depends on both the type and length of an array

1D Array: how array looks like (2)



		←4 bytes→
10127	x[0]	3.1
10131	x[1]	4.2
10135	x[2]	5.0
...	...	...
10279	x[38]	3.3
10283	x[39]	4.2

```c
#include <stdio.h>
int main()
{
    int a[10], b = 3;
    char c[10];
    printf("a: %d\n", sizeof(a));
    printf("b: %d\n", sizeof(b));
    printf("c: %d\n", sizeof(c));
    return 0;
}
```

- The system opens a continuous memory block for an array
- Actual size depends on both the type and length of an array

1D Array: how array looks like (3)

```c
#include <stdio.h>
int main()
{
    int a[10], b = 3;
    char c[10];
    printf("a: %d\n", sizeof(a));
    printf("b: %d\n", sizeof(b));
    printf("c: %d\n", sizeof(c));
    return 0;
}
```

[Output]

```
a:  40
b:  4
c:  10
```

- Actual size depends on both the type and length of an array

1D Array: initialization (1)

- No initialization, what happens

[1: local]

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

[2: static]

```c
#include <stdio.h>
int main()
{
 static int a[10];
 int i = 0;
 for(;i<10; i++)
 printf("%d-",a[i]);
 return 0;
}
```

[3: external]

```c
#include <stdio.h>
extern a[10];
int main()
{
 int i = 0;
 for(; i<10; i++)
 printf("%d-",a[i]);
 return 0;
}
```

1. Initialize to random numbers
2. Initialize to zeros
3. Initialize to zeros

1D Array: initialization (2)

- Initializations as follows are valid

```c
#include <stdio.h>
int main()
{
    int a[10] = {3, 2, 5,
     1};
    int i = 0;
    for(; i < 10; i++)
    printf("%d-",a[i]);
    return 0;
}
```

```c
#include <stdio.h>
int main()
{
    int a[] = {3, 2, 5, 1};
    int i = 0;
    for(; i < 4; i++)
    printf("%d-",a[i]);
    return 0;
}
```

1D Array: initialization (3)

- Initializations as follows are invalid

```
#include <stdio.h>
int main()
{
    int a[10];
    a[10] = {3, 2, 5, 1};
    int i = 0;
    for(; i < 10; i++)
    printf("%d-",a[i]);
    return 0;
}
```

```
#include <stdio.h>
int main()
{
    int a = {3, 2, 5, 1};
    int i = 0;
    for(; i < 4; i++)
    printf("%d-",a[i]);
    return 0;
}
```

1D Array Example-1 (1)

- Given an array: a[10] = {3, 21, 5, 8, 5,11, 22,14,9,51}
- Flip the array to: {51, 9, 14, 22, 11, 5, 8, 5, 21, 3}

5 minutes to think about the solution

1D Array Example-1 (2)

- Given an array: $a[10] = \{3, 21, 5, 8, 5,11, 22,14,9,51\}$
- Flip the array to: $\{51, 9, 14, 22, 11, 5, 8, 5, 21, 3\}$
- The idea is that, we only need to swap two elements each time
- One for the header, one from the rear
- We do this for $\frac{10}{2}$ times

1D Array Example-1 (3)

- Given an array: a[10] = {3, 21, 5, 8, 5,11, 22,14,9,51}
- Flip the array to: {51, 9, 14, 22, 11, 5, 8, 5, 21, 3}

```
1 For i from 0 to $\frac{N}{2}$ do
2     Exchange a[i] with a[N-i-1]
3 End-for
```

- Let's do it, give you another 5 minutes ...

1D Array Example-1 (4)

```
1 For i from 0 to $\frac{N}{2}$ do
2     Exchange a[i] with a[N-i-1]
3 End-for
```

```c
#include <stdio.h>
int main()
{
    int a[10] ={3,21,5,8,5,11,22,14,9,51};
    int t = 0, i = 0;
    for(; i < 5; i++){
        t = a[i];
        a[i] = a[10-i-1];
        a[10-i-1] = t;
    }
    for(i = 0; i < 10; i++){
        printf("%d-", a[i]);
    }
    return 0;
}
```

Dynamic Array Example-2 (1)

- An integer number given by user
- int a[n]
- Input "n" numbers assign to "a[n]"
- Print out array "a"

5 minutes to think about the solution

Dynamic Array Example-2 (2)

```c
#include <stdio.h>
int main()
{
    int n = 0, i = 0;
    scanf("%d", &n);
    int a[n];
    for(i = 0; i < n; i++)
    {
        scanf("%d", &a[i]);
    }
    printf("%d\n", sizeof(a));
    for(i = 0; i < n; i++)
    {
        printf("a[%d] = %d\n", i, a[i]);
    }
    return 0;
}
```

1D Array Example-3 (1)

- Given a sorted array: a[7] = {3, 14, 15, 18, 22,35}
- Insert an input number to the array
- Keep the array sorted after the insertion

5 minutes to think about the solution...

1D Array Example-3 (2)

```c
#include <stdio.h>
int main(){
   int a[7] = {3,14,15,18,22,35};
   int b = 7, i = 0, j = 0;
   scanf("%d", &b);
   for(i = 0; i < 6; i++){
      if(b < a[i]){
         break;
      }
   }
   for(j = 6; j > i; j--){
      a[j] = a[j-1];
   }
   a[j] = b;
   for(i = 0; i < 7; i++){
      printf("a[%d] = %d\n", i, a[i]);
   }
   return 0;
}
```

1D Array Example-4 (1)

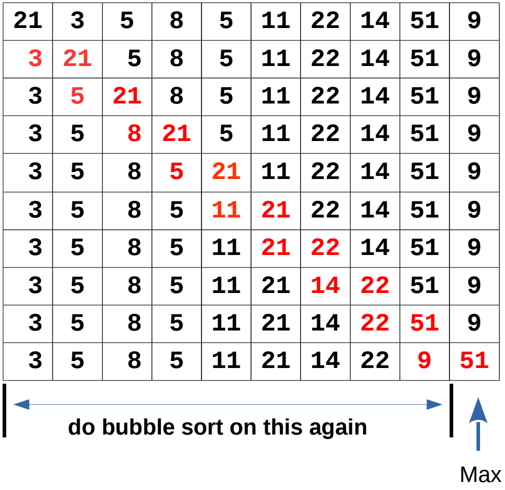
- Given an array: a[10] = {21, 3, 5, 8, 5,11, 22,14,51,9}
- Sort the array in ascending order: {3, 5, 5, 8, 9, 11, 14, 21, 22, 51}

5 minutes to think about the solution...

1D Array Example-4 (2)

- Given an array: a[10] = {21, 3, 5, 8, 5,11, 22,14,9,51}
- Sort the array in ascending order: {3, 5, 5, 8, 9, 11, 14, 21, 22, 51}
- The idea is bubble sort, which is a classic method for sorting
- Each time, we move the largest to the rear of the array
- Repeat this on sub-array for N times

1D Array Example-4 (3)



21	3	5	8	5	11	22	14	51	9
3	21	5	8	5	11	22	14	51	9
3	5	21	8	5	11	22	14	51	9
3	5	8	21	5	11	22	14	51	9
3	5	8	5	21	11	22	14	51	9
3	5	8	5	11	21	22	14	51	9
3	5	8	5	11	21	22	14	51	9
3	5	8	5	11	21	14	22	51	9
3	5	8	5	11	21	14	22	51	9
3	5	8	5	11	21	14	22	9	51

Figure: Demo of one round of bubble sort

1D Array Example-4 (4)

- Let's now outline the procedure

```
1 For i from 0 to N do
2   For j from 0 to N-i do
3     Check a[j] and a[j+1]
4     If a[j] > a[j+1]
5       swap them
6     End-if
7   End-for(j)
8 End-for(i)
```

1D Array Example-4 (5): the code

```c
#include <stdio.h>
int main()
{
    int a[10] = {3, 5, 5, 8, 9, 11, 14, 21, 22, 51};
    int i = 0, j = 0, t = 0;
    for(i = 0; i < 10; i++) {
        for(j = 0; j < (10-i-1); j++) {
            if(a[j] > a[j+1])
            {
                t = a[j];
                a[j] = a[j+1];
                a[j+1] = t;
            }//if(a[j])
        }//for(j)
    }//for(i)
    for(i = 0; i < 10; i++) {
        printf("%d, ", a[i]);
    }
    return 0;
}
```

Outline

Opening Discussion: 2D Array

- Continue with the opening example in the last section
- In your class, you might have several courses for each student
- So we need several 1D arrays
- Alternatively, we can use a 2D array

```
int main()
{
    float math[40];
    float c[40];
    float phis[40];
    float bio[40];
    ...
}
```

```
int main()
{
    float courses[40][4];
    ...
}
```

2D Array: declaration

type arrayName[row][column];

- Similar as 1D array, type is required
- "arrayName" should be unique
- "row" and "column" should be constant expressions

```
int main()
{
    float a[40][4]; //there 40 rows and 4 columns in each row
    a[3][2] = 3.14;
    return 0;
}
```

2D Array: initialization (1)

```c
int main()
{
    float a[3][4] = {{1,3,1,1},{1,2,1,3},{1,12,1,2}};
    return 0;
}
```

- Following way is also valid

```c
int main()
{
    float a[3][4] = {1,3,1,1,1,2,1,3,1,12,1,2};
    return 0;
}
```

2D Array: initialization (2)

```c
int main()
{
    float a[][4] = {{1,3,1,1},{1,2,1,3},{1,12,1,2}};
    return 0;
}
```

- Following way is also valid, $row = \lceil \frac{N}{4} \rceil$

```c
int main()
{
    float a[][4] = {1,3,1,1,1,2,1,3,1,12,1,2};
    return 0;
}
```

- If no initialization, set to 0 by default

2D Array: initialization (3)

```
int main()
{
    float a[][4] = {{1,3,1,1},{1,2,1,3},{1,12,1,2}};
    return 0;
}
```

- Following way is also invalid

```
int main()
{
    float a[4][] = {1,3,1,1,1,2,1,3,1,12,1,2};
    return 0;
}
```

- It is organized in row major order

2D Array: how it looks like

		◄– 4 bytes–►
10127	a[0][0]	3.1
10131	a[0][1]	4.2
10135	a[0][2]	5.0
...	a[0][3]	0
	a[1][0]	7
	...	...
???	a[2][1]	3.1
???	a[2][2]	3.3
10171	a[2][3]	4.2

- 3(row)×4(column)×4 bytes

2D Array: visit element of the array

```c
int main()
{
    float a[][4] = {1,3,1,1,1,2,1,3,1,12,1,2};
    int i = 0, j = 0;
    for(i = 0; i < 3; i++)
    {
        for(j = 0; j < 4; j++)
        {
            printf("%f-", a[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

Example-1: transpose a matrix (1)

- Given a 2D matrix, kept in a 1D array
- a[12] = {12,13,5,5,7,21,6,4,10,5,5,9}

$$A = \begin{bmatrix} 12 & 13 & 5 \\ 5 & 7 & 21 \\ 6 & 4 & 10 \\ 5 & 5 & 9 \end{bmatrix}_{4\times 3} \quad \Rightarrow \quad B = A^T = \begin{bmatrix} 12 & 5 & 6 & 5 \\ 13 & 7 & 4 & 5 \\ 5 & 21 & 10 & 9 \end{bmatrix}_{3\times 4}$$

Example-1: transpose a matrix (2)

- Given a 2D matrix, kept in a 1D array
- a[12] = {12,13,5,5,7,21,6,4,10,5,5,9}

$$A = \left[\begin{array}{cccc} & & j & \\ & 12 & 13 & 5 \\ & 5 & 7 & 21 \\ i & 6 & 4 & 10 \\ & 5 & 5 & 9 \end{array}\right]_{4\times3} \Rightarrow B = A^T = \left[\begin{array}{ccccc} & & & i & \\ & 12 & 5 & 6 & 5 \\ j & 13 & 7 & 4 & 5 \\ & 5 & 21 & 10 & 9 \end{array}\right]_{3\times4}$$

1. A[i][j] $\rightleftarrows$ B[j][i]

Example-1: transpose a matrix (3)

- Given a 2D matrix, kept in a 1D array
- a[12] = {12,13,5,5,7,21,6,4,10,5,5,9}

$$A = \begin{bmatrix} 12 & 13 & 5 \\ 5 & 7 & 21 \\ 6 & 4 & 10 \\ 5 & 5 & 9 \end{bmatrix}_{4\times 3} \quad \Rightarrow \quad B = A^T = \begin{bmatrix} 12 & 5 & 6 & 5 \\ 13 & 7 & 4 & 5 \\ 5 & 21 & 10 & 9 \end{bmatrix}_{3\times 4}$$

- b[12] ={12,5,6,5,13,7,4,5,5,21,10,9}

1. A[i][j] $\rightleftarrows$ B[j][i]
2. A[i][j] $\Rightarrow$ a[i*c1+j], where c1=3
3. B[j][i] $\Rightarrow$ b[j*c2+i], where c2=4
4. a[i*c1+j] $\rightleftharpoons$ b[j*c2+i]

Example-1: transpose a matrix (4)

```
#include <stdio.h>
int main(){
  int a[12] = {12,13,5,5,7,21,6,4,10,5,5,9};
  int b[12] = {0};
  int i = 0, j = 0;
  for(i = 0; i < 4; i++){
    for(j = 0; j < 3; j++){
      b[j*4+i] = a[i*3+j];
    }
  }
  for(i = 0; i < 3; i++){
    for(j = 0; j < 4; j++){
       printf("%4d", b[i*4+j]);
    }
    printf("\n");
  }
  return 0;
}
```

Outline

Opening discussion

- Now, we are going to discuss a special kind of array
- Array of chars, we give it a new name **string**
- Different from integer array, empty elements are set to '\0'

```
#include <stdio.h>
 int main()
 {
     char hi[8] ={'h','e','l'
     ,'l','o'};
     int i = 0;
     for(i < 8; i++)
     {
         printf("%c", hi[i]);
     }
     return 0;
 }
     [Output: hello    ]

```

```
#include <stdio.h>
 int main()
 {
     char hi[8] ={'h','e','l'
     ,'l','o'};
     int i = 0;
     printf("%s", hi);
     return 0;
 }
     [Output: hello]

```

String: definition and initialization

- First of all, it is an array
- We can initialize it as an array

```c
#include <stdio.h>
int main()
{
    char ch[6] = {'H','e','l','l','o','\0'};
    char ch[]  = {'H','e','l','l','o','\0'};
    /*we have 6 chars there*/
    char ch[6] = {'H','e','l','l','o'};
    /*'\0' is automatically appended **/
    char ch[6] = {"Hello"};
    char ch[6] = "Hello";
    char ch[]  = "Hello";
    return 0;
}
```

String Operation: strcpy

- Copy one string to another
- strcpy(destine, source)

```c
#include <stdio.h>
#include <string.h>
int main()
{
    char ch[10];
    strcpy(ch, "hi");
    printf("%s\n", ch);
    strcpy(ch, "ha");
    printf("%s\n", ch);
    return 0;
}
```

[Output:]

```
hi
ha
```

String Operation: strcmp (1)

- Compare whether two strings are equal or not

```
#include <stdio.h>
#include <string.h>
int main()
{
   char ch1[10], ch2[10];
   strcpy(ch1, "hi");
   strcpy(ch2, "ha");
   if(strcmp(ch1, ch2) == 1){
        printf("ch1 > ch2\n");
   }else if(strcmp(ch1,ch2)==-1)
   {
        printf("ch1 < ch2\n");
   }
   else if(strcmp(ch1,ch2)==0){
        printf("identical");
   }
}
```

[Output]
ch1 > ch2

String Operation: strcmp (2)

- Compare whether two strings are equal or not

```c
#include <stdio.h>
#include <string.h>
int main()
{
   char ch1[10], ch2[10];
   strcpy(ch1, "he");
   strcpy(ch2, "we");
   if(strcmp(ch1, ch2) == 1)
   {
        printf("ch1 > ch2\n");
   }else if(strcmp(ch1,ch2)==-1)
   {
        printf("ch1 < ch2\n");
   }
   else if(strcmp(ch1,ch2)==0)
   {
        printf("identical");
   }
}
```

[Output]

```
ch1 < ch2
```

String Operation: strcmp (3)

- Compare whether two strings are equal or not

```c
#include <stdio.h>
#include <string.h>
int main()
{
  char ch1[10], ch2[10];
  strcpy(ch1, "hi");
  strcpy(ch2, "hi");
  if(strcmp(ch1, ch2)!=0)
  {
      printf("different");
  }
  else if(strcmp(ch1, ch2)==0)
  {
      printf("identical");
  }
  return 0;
}
```

[Output]
identical

String Operation: strlen (1)

- Calculate the length of the string
- Pass the string until it encounters '\0'

```c
#include <stdio.h>
#include <string.h>
int main()
{
    char a[20] = "hello";
    int l = strlen(a);
    printf("length is: %d", l);
    return 0;
}
```

```c
#include <stdio.h>
#include <string.h>
int main()
{
    char a[20] = "hello world";
    int l = strlen(a);
    printf("length is: %d", l);
    return 0;
}
```

[length is: 5]

[length is: 11]

String Operation: strlen (2)

- Calculate the length of the string
- Pass the string until it encounters '\0'

```c
#include <stdio.h>
#include <string.h>
int main()
{
    char a[20] = "hello world";
    int l = strlen(a);
    printf("length is: %d", l);
    return 0;
}
```

```c
#include <stdio.h>
#include <string.h>
int main()
{
    char a[20]="hello\0world";
    int l = strlen(a);
    printf("length is: %d", l);
    return 0;
}
```

```
[length is: 11]
```

```
[length is: 5]
```

String Operation: strcat

- Concatenate two strings into one

```c
#include <stdio.h>
#include <string.h>
int main()
{
    char a[20] = "hello ";
    char b[10] = "world";
    printf("a=%s\n", a);
    printf("b=%s\n", b);
    strcat(a, b);
    printf("a=%s\n", a);
    return 0;
}
```

```
hello
world
hello world
```

Summary over string and char array

- Array of chars could be used as string, '\0' should be appended at the end
- One more byte should be reserved for '\0'
- String can be used as an array of chars
- Functions such as "strcpy", "strlen", "strcat" etc require string input

Usage	Comments
strcpy(str1, str2)	Copy "str2" to "str1", the content of "str1" will be overwriten
strlen(str1)	Calculate the number of characters before '\0'
strcat(str1, str2)	Concatenate "str2" to "str1" and save to "str1"
strcmp(str1, str2)	Compare two strings, returns -1, 1 or 0 if they are identical