

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*
Spring Semester 2022

1 Arrays

- 1D Array
- 2D Array

2 Strings

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??

```
1 #include <stdio.h>
2 int main()
3 {
4     float x1, x2, x3, x4, x5, x6, x7, x8, x9, x10;
5     float sum = 0, avg = 0;
6     scanf("%f", &x1);
7     sum += x1;
8     scanf("%f", &x2);
9     sum += x2;
10    ...
11    avg = sum/10;
12    return 0;
13 }
```

Opening Discussion (2)

```
1 #include <stdio.h>
2 int main()
3 {
4     float x1, x2, x3, x4, x5, x6, x7, x8, x9, x10;
5     float sum = 0, avg = 0;
6     scanf("%f", &x1);
7     sum += x1;
8     scanf("%f", &x2);
9     sum += x2;
10    ...
11    avg = sum/10;
12    return 0;
13 }
```

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

1 Arrays

- 1D Array
- 2D Array

2 Strings

1D Array: declaration (1)

- 1D array is defined in following form

`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**

`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**

```
1 int main()  
2 {  
3     float x[40];  
4     ....  
5     return 0;  
6 }
```

```
1 int main()  
2 {  
3     const int N = 40;  
4     float x[N];  
5     ....  
6     return 0;  
7 }
```

```
1 int main()  
2 {  
3     int N = 40;  
4     float x[N];  
5     ....  
6     return 0;  
7 }
```

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**

```
1 #define N 40
2 int main()
3 {
4     float x[N];
5     ....
6     return 0;
7 }
```

```
1 int main()
2 {
3     const int N = 40;
4     float x[N];
5     ....
6     return 0;
7 }
```

```
1 #define N 40
2 int main()
3 {
4     float x[3*N];
5     ....
6     return 0;
7 }
```


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 " $x[2]$ "

```
1 #include <stdio.h>
2 int main()
3 {
4     float x[40];
5     x[0] = 5.0;
6     x[2] = 3.1;
7     printf("x[0] = %f", x[0]);
8     return 0;
9 }
```

1D Array: visit array element (2)

```
1 int main()  
2 {  
3     float x[40];  
4     int i = 0;  
5     for(i = 0; i < 40; i++)  
6     {  
7         printf("Input %d:", i);  
8         /*—be careful below—*/  
9         scanf("%f", &(x[i]));  
10    }  
11    return 0;  
12 }
```

- 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

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

- 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)

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

[Output]

```
1 a: 40
2 b: 4
3 c: 10
```

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

1D Array: initialization (1)

- No initialization, what happens

[1: local]

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

[2: static]

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

[3: external]

```
#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**

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

```
1 #include <stdio.h>
2 int main()
3 {
4     int a[] = {3, 2, 5, 1};
5     int i = 0;
6     for (; i < 4; i++)
7         printf("%d_", a[i]);
8     return 0;
9 }
```

1D Array: initialization (3)

- Initializations as follows are **invalid**

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

```
1 #include <stdio.h>
2 int main()
3 {
4     int a = {3, 2, 5, 1};
5     int i = 0;
6     for (; i < 4; i++)
7         printf("%d_", a[i]);
8     return 0;
9 }
```


1D Array Example1 (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 Example1 (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 Example1 (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 Example1 (4)

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

```
1 #include <stdio.h>
2 int main()
3 {
4     int a[10] = {3,21,5,8,5,11,22,14,9,51};
5     int t = 0, i = 0;
6     for(; i < 5; i++){
7         t = a[i];
8         a[i] = a[10-i-1];
9         a[10-i-1] = t;
10    }
11    for(i = 0; i < 10; i++){
12        printf("%d_", a[i]);
13    }
14    return 0;
15 }
```

1D Array Example2 (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 Example2 (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 Example2 (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

do bubble sort on this again

Max

Figure: Demo of one round of bubble sort

1D Array Example2 (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 Example2 (5): the code

```
1 #include <stdio.h>
2 int main()
3 {
4     int a[10] = {3, 5, 5, 8, 9, 11, 14, 21, 22, 51};
5     int i = 0, j = 0, t = 0;
6     for(i = 0; i < 10; i++) {
7         for(j = 0; j < (10-i-1); j++) {
8             if(a[j] > a[j+1])
9                 {
10                     t = a[j];
11                     a[j] = a[j+1];
12                     a[j+1] = t;
13                 } // if(a[j])
14         } // for(j)
15     } // for(i)
16     for(i = 0; i < 10; i++) {
17         printf("%d_", a[i]);
18     }
19     return 0;
20 }
```

1 Arrays

- 1D Array
- 2D Array

2 Strings

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

```
1 int main()  
2 {  
3     float math[40];  
4     float c[40];  
5     float phis[40];  
6     float bio[40];  
7     ...  
8 }
```

```
1 int main()  
2 {  
3     float courses  
4         [40][4];  
5     ...  
6 }
```

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

```
1 int main()  
2 {  
3     float a[40][4]; //there 40 rows and 4 columns in each row  
4     a[3][2] = 3.14;  
5     return 0;  
6 }
```

2D Array: initialization (1)

```
1 int main()  
2 {  
3     float a[3][4] = {{1,3,1,1},{1,2,1,3},{1,12,1,2}};  
4     return 0;  
5 }
```

- Following way is also **valid**

```
1 int main()  
2 {  
3     float a[3][4] = {1,3,1,1,1,2,1,3,1,12,1,2};  
4     return 0;  
5 }
```

2D Array: initialization (2)

```
1 int main()  
2 {  
3     float a[][4] = {{1,3,1,1},{1,2,1,3},{1,12,1,2}};  
4     return 0;  
5 }
```

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

```
1 int main()  
2 {  
3     float a[][4] = {1,3,1,1,1,2,1,3,1,12,1,2};  
4     return 0;  
5 }
```

- If no initialization, set to **0** by default

2D Array: initialization (3)

```
1 int main()  
2 {  
3     float a[][4] = {{1,3,1,1},{1,2,1,3},{1,12,1,2}};  
4     return 0;  
5 }
```

- Following way is also **invalid**

```
1 int main()  
2 {  
3     float a[4][] = {1,3,1,1,1,2,1,3,1,12,1,2};  
4     return 0;  
5 }
```

- 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

- $3(\text{row}) \times 4(\text{column}) \times 4 \text{ bytes}$

2D Array: visit element of the array

```
1 int main()  
2 {  
3     float a[][4] = {1,3,1,1,1,2,1,3,1,12,1,2};  
4     int i = 0, j = 0;  
5     for(i = 0; i < 3; i++)  
6     {  
7         for(j = 0; j < 4; j++)  
8         {  
9             printf("%f_", a[i][j]);  
10        }  
11        printf("\n");  
12    }  
13    return 0;  
14 }
```

1 Arrays

- 1D Array
- 2D Array

2 Strings

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'

```
1 #include <stdio.h>
2 int main()
3 {
4     char hi[8] ={'h','e','l','l','o'};
5     int i = 0;
6     for(i < 8; i++)
7     {
8         printf("%c", hi[i]);
9     }
10    return 0;
11 }
12 [Output: hello ]
13
```

```
1 #include <stdio.h>
2 int main()
3 {
4     char hi[8] ={'h','e','l','l','o'};
5     int i = 0;
6     printf("%s", hi);
7     return 0;
8 }
9 [Output: hello]
10
```

String: definition and initialization

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

```
1 #include <stdio.h>
2 int main()
3 {
4     char ch[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
5     char ch[] = {'H', 'e', 'l', 'l', 'o', '\0'};
6     /*we have 6 chars there*/
7     char ch[6] = {'H', 'e', 'l', 'l', 'o'};
8     /*'\0' is automatically appended */
9     char ch[6] = {"Hello"};
10    char ch[6] = "Hello";
11    char ch[] = "Hello";
12    return 0;
13 }
```

String Operation: strcpy

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

```
1 #include <stdio.h>
2 #include <string.h>
3 int main()
4 {
5     char ch[10];
6     strcpy(ch, "hi");
7     printf("%s\n", ch);
8     strcpy(ch, "ha");
9     printf("%s\n", ch);
10    return 0;
11 }
```

[Output:]

```
1 hi
2 ha
```

String Operation: strcmp (1)

- **C**ompare whether two strings are equal or not

```
1 #include <stdio.h>
2 #include <string.h>
3 int main()
4 {
5     char ch1[10], ch2[10];
6     strcpy(ch1, "hi");
7     strcpy(ch2, "ha");
8     if(strcmp(ch1, ch2) == 1){
9         printf("ch1 > ch2\n");
10    } else if(strcmp(ch1, ch2) == -1)
11    {
12        printf("ch1 < ch2\n");
13    }
14    else if(strcmp(ch1, ch2) == 0){
15        printf("identical");
16    }
17 }
```

[Output]
ch1 > ch2

String Operation: strcmp (2)

- Compare whether two strings are equal or not

```
1 #include <stdio.h>
2 #include <string.h>
3 int main()
4 {
5     char ch1[10], ch2[10];
6     strcpy(ch1, "he");
7     strcpy(ch2, "we");
8     if(strcmp(ch1, ch2) == 1)
9     {
10         printf("ch1 > ch2\n");
11     } else if(strcmp(ch1, ch2) == -1)
12     {
13         printf("ch1 < ch2\n");
14     }
15     else if(strcmp(ch1, ch2) == 0)
16     {
17         printf("identical");
18     }
19 }
```

[Output]
ch1 < ch2

String Operation: strcmp (3)

- Compare whether two strings are equal or not

```
1 #include <stdio.h>
2 #include <string.h>
3 int main()
4 {
5     char ch1[10], ch2[10];
6     strcpy(ch1, "hi");
7     strcpy(ch2, "hi");
8     if(strcmp(ch1, ch2)!=0)
9     {
10         printf("different");
11     }
12     else if(strcmp(ch1, ch2)==0)
13     {
14         printf("identical");
15     }
16     return 0;
17 }
```

[Output]
identical

String Operation: strlen (1)

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

```
1 #include <stdio.h>
2 #include <string.h>
3 int main()
4 {
5     char a[20] = "hello";
6     int l = strlen(a);
7     printf("length is: %d", l);
8     return 0;
9 }
```

[length is: 5]

```
1 #include <stdio.h>
2 #include <string.h>
3 int main()
4 {
5     char a[20] = "hello_world";
6     int l = strlen(a);
7     printf("length is: %d", l);
8     return 0;
9 }
```

[length is: 11]

String Operation: strlen (2)

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

```
1 #include <stdio.h>
2 #include <string.h>
3 int main()
4 {
5     char a[20] = "hello_world";
6     int l = strlen(a);
7     printf("length is: %d", l);
8     return 0;
9 }
```

[length is: 11]

```
1 #include <stdio.h>
2 #include <string.h>
3 int main()
4 {
5     char a[20]="hello\0world";
6     int l = strlen(a);
7     printf("length is: %d", l);
8     return 0;
9 }
```

[length is: 5]

String Operation: strcat

- Concatenate two strings into one

```
1 #include <stdio.h>
2 #include <string.h>
3 int main()
4 {
5     char a[20] = "hello ";
6     char b[10] = "world";
7     printf("a=%s\n", a);
8     printf("b=%s\n", b);
9     strcat(a, b);
10    printf("a=%s\n", a);
11    return 0;
12 }
```

```
1 hello
2 world
3 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
<code>strcpy(str1, str2)</code>	Copy <code>"str2"</code> to <code>"str1"</code> , the content of <code>"str1"</code> will be overwritten
<code>strlen(str1)</code>	Calculate the number of characters before <code>'\0'</code>
<code>strcat(str1, str2)</code>	Concatenate <code>"str2"</code> to <code>"str1"</code> and save to <code>"str1"</code>
<code>strcmp(str1, str2)</code>	Compare two strings, returns -1, 1 or 0 if they are identical