




UESTC 1005 — Introductory Programming

Lecture 10 - Strings and Structures

Hasan T Abbas

Hasan.Abbas@glasgow.ac.uk

Lecture Outline

- Strings 
- Structure 
- Structure Pointers 

```
"You are sad," the Knight said in an anxious tone: "let me sing you a song to comfort you."  
"Is it very long?" Alice asked, for she had heard a good deal of poetry that day.  
"It's long," said the Knight, "but very, very beautiful. Everybody that hears me sing it—either it brings  
the tears into their eyes, or else—"  
"Or else what?" said Alice, for the Knight had made a sudden pause.  
"Or else it doesn't, you know. The name of the song is called 'Haddocks' Eyes'."  
"Oh, that's the name of the song, is it?" Alice said, trying to feel interested.  
"No, you don't understand," the Knight said, looking a little vexed.  
"That's what the name is called. The name really is 'The Aged Aged Man'."  
"Then I ought to have said 'That's what the song is called'?" Alice corrected herself.  
"No, you oughtn't: that's quite another thing! The song is called 'Ways And Means': but that's only what  
it's called, you know!"  
"Well, what is the song, then?" said Alice, who was by this time completely bewildered.  
"I was coming to that," the Knight said. "The song really is 'A-sitting On A Gate': and the tune's my own  
invention."
```

But First a Recap ..

- A **pointer** is a variable that stores the address of another variable.
- Efficient memory management
- Direct access to data
- Dynamic memory handling.

```
#include <stdio.h> // which has definitions of printf function

int main() // void means nothing
{
    int x = 10;
    int *ptr = &x;
    printf("%d\n", *ptr); // Outputs 10
    return 0;
}
```

Strings

- An array of characters terminated by the **null character** (`\0`).
- A contiguous block of memory where each character is accessible via indexing or pointers.
- A nice library `string.h` with many useful functions

```
char str[10] = "Chengdu";
```

54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69
	58			62		55		C	h	e	n	g	d	u	\0

The `string` library

- The purpose of header files is to call library functions
- `#include<string.h>` has many useful strings functions

```
char str1[10] = "UESTC";  
char str2[10] = "1005"  
printf("%s %s", str1, str2); // %s placeholder for strings  
strcat(str1, str2); // concatenates two strings  
strcpy(str1, str2); // copies one strings to another  
strcmp(str1, str2); // compare the elements of two strings  
strlen(str1);
```

Strings and Pointers

- All we said about arrays, also true for strings
- The name of the string points to the first string element
- Used in efficient data processing involving text (names, passwords)

```
char str[] = "UESTC 1005";  
char *ptr = str; // Points to the first character  
printf("%c", *(ptr + 1)); // Outputs 'E'  
}
```

```
char *str = (char *)malloc(10 * sizeof(char));
```

String Tokenisation

- Split a single string to multiple components (tokens)
- Like an IP address `192.168.1.1` .
- `strtok()` function splits a string based on specified delimiter
- Example - IP address

```
char *strtok(char *str, const char *delim);
```


The `strtok()` Function

- First call is different from the subsequent calls

```
#include <stdio.h>
#include <string.h>

int main() {
    char str[] = "apple,banana,orange";
    char *token;

    // Get the first token
    token = strtok(str, ",");
    while (token != NULL) {
        printf("%s\n", token); // Print the token
        token = strtok(NULL, ","); // Get the next token
    }

    return 0;
}
```

Structures in C

So what is a Structure

- A user-defined data type with group of variables of different types
- Better data organisation
- Allocate storage all at once

```
struct StructureName {  
    dataType1 member1;  
    dataType2 member2;  
};
```

```
struct {  
    char    name[10];  
    int     age;  
    char    gender;  
    double  weight ; // in kgs  
} alien;
```

Why Structures? 🤔

- Organise related data together.
- Manage complex datasets effectively.
- Modular Code Design by grouping similar data types

```
struct Coordinate {  
    int x, y, z;  
};  
struct Coordinate p1 = {10, 20, 30};
```

```
// doing it dynamically  
struct Coordinate *p = (struct Coordinate *)malloc(sizeof(struct Coordinate));  
p->x = 10; p->y = 20; p->z = 30;
```

Accessing the Structure Members

```
p1.x = 15;  
printf("%d" , p1.x);
```

- Using Pointers

```
struct Coordinate *ptr = &p1;  
ptr->x = 15;
```

An Array of Structures

- An example of a log-book (bike shop customer record)
- Best to use when individual records have multiple attributes of different data types.
- Arrays of structures allow passing complex data layouts to functions in a more structured and efficient way.

```
// Structure containing an array
struct Student {
    char name[50];
    int age;
    float grade;
};
```

Structures Array

```
// Declare an array of 100 students
struct Student students[100];
// Function to calculate average grade
float calculateAverageGrade(struct Student students[], int size) {
    float total = 0;
    for (int i = 0; i < size; i++) {
        total += students[i].grade;
    }
    return total / size;
}
```

Structure Pointers

- A pointer to a structure points to the memory address where the structure is located
- The name of the structure is not a pointer
- In this sense, it is closer to a variable
- If the structure contains an array, the pointer can be used to indirectly access the array elements.
- Structure can contain a self-referencing pointer to another structure

```
#include <stdio.h>
struct Student {
    char name[50];           // Name of the student
    int grades[5];           // Array to store 5 grades
};
```


Structure Pointers

```
int main() {  
    // Declare and initialize a structure  
    struct Student ip_student = {"DaXue Sheng", {85, 90, 88, 92, 67}};  
    // Declare a pointer to the structure  
    struct Student *ptr = &student1;  
    // Access array elements using the pointer  
    printf("Student Name: %s\n", ptr->name);  
    printf("Grade 1: %d\n", ptr->grades[0]);  
    printf("Grade 5: %d\n", ptr->grades[4]);  
    return 0;  
}
```

Mixing it all Together

- We could have a pointer to the structure allowing indirect access
- We can have an array of structures
- Many possible solutions to a given problem!

Next Up

- Bit Manipulation and Structures (Dr Syed Raza)