

UESTC 1005 — Introductory Programming

Lecture 10 - Strings and Structures

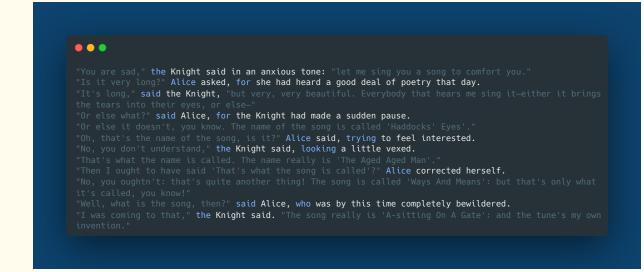
Hasan T Abbas

Hasan.Abbas@glasgow.ac.uk

Strings hold data - Structures organise it

Lecture Outline

- Strings
- Structure
- Structure Pointers



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 **I**

- 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";
     55
          56
              57
                   58
                        59
                            60
                                 61
                                     62
                                          63
                                               64
                                                   65
                                                        66
                                                             67
                                                                 68
                                                                      69
 54
                    62
                             55
                                          | h | e
```

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 multile 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;
};
```

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;
}</pre>
```

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

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)