Part V
Structures

1) Introduction
2) Defining Structures
3) Declaring Variables
4) Using Structures
5) Initialisation of Structures
6) Nested Structures
7) Arrays of Structures
8) Structures as Function Parameters
9) Dynamic Memory Allocation
10) Complex Example
5.1 Introduction

- **Structure**:  
  - Data structure  
  - Stores a collection of data items  
  - The data items may not be of the same type

- **Idea**:

- **Examples**:  
  - structure of personal info

<table>
<thead>
<tr>
<th>1985</th>
<th>“John D”</th>
<th>53121312</th>
<th>“EENG”</th>
</tr>
</thead>
</table>

  - address structure

  | 55   | “Oak Rd” | “Glasnevin” | “Dublin” | 9 |
  |------|----------|-------------|----------|

- **Components**:
  - Structure name  
  - List of items  
  - Items are indicated by: type and name
5.2 Definition

1) Structure Template

```
"struct" <struct_tag_name> '{ '<list_of_items>'}' ';' 
```

- Notes:
  - Defines a template for the structure
  - Indicates the structure’s components
  - Does not define any variable
  - Does not allocate memory

2) Structure Type

```
"typedef" "struct" <struct_tag_name>
 '{ '<list_of_items>' }' <struct_type_name> ' ; ' 
```

- Notes:
  - Defines a new structured type
  - Indicates the type’s components
  - Does not define any variable
  - Does not allocate memory

- Components:
  - Tag name:
    `<struct_tag_name> = <name>`

  - List of items:
    `<list_of_items> = [ <type> <item_name> 
    [<item_size>] ; ' + <list_of_items> ]`

    `<item_size> = '[' <const_expression> ']'`

  - Type name:
    `<struct_type_name> = <name>`
• **Usage:**

1) **Structure Template**

   ```
   struct address_template {
     int no;
     char street[20];
     char area[20];
     char city[20];
     int code;
   };
   
   struct personal_template {
     int birth_year;
     char name[20];
     int ID;
     char school[4];
   };
   ```

2) **Structure Type**

   ```
   typedef struct address_tag {
     int no;
     char street[20];
     char area[20];
     char city[20];
     int code;
   } address_type;
   
   typedef struct personal_tag {
     int birth_year;
     char name[20];
     int ID;
     char school[4];
   } personal_type;
   ```
5.3 Variable Declaration

1) “struct” <struct_tag_name>
   
   <struct_variable_name> ‘;’

   • Components:
     
     • Struct tag name: <struct_tag_name>
       - must be defined prior to usage
     
     • Variable name: <struct_variable_name>
       - variable to be declared

     <struct_variable_name> = <name>

   • Example:

     • struct address_template my_address;
     • struct personal_template personal_info;

2) <struct_type_name> <struct_variable_name> ‘;’

   • Components:

     • Struct type name: <struct_type_name>
       - must be defined prior to usage
     
     • Variable name: <struct_variable_name>
       - variable to be declared

     <variable_name> = <name>

   • Example:

     • address_type my_address;
     • personal_type personal_info;
• Effect:

• The computer allocates for the declared variable a contiguous memory block of size equal to the added size of all the structure’s component items

• Example:

• Variable my_address: ⇒ 68 Bytes

  int no; ⇒ 4 * 1 = 4 B
  char street [20]; ⇒ 1 * 20 = 20 B
  char area [20]; ⇒ 1 * 20 = 20 B
  char city [20]; ⇒ 1 * 20 = 20 B
  int code; ⇒ 4 * 1 = 4 B

• Variable personal_info: ⇒ 32 Bytes

  int birth_year; ⇒ 4 * 1 = 4 B
  char name [20]; ⇒ 1 * 20 = 20 B
  int ID; ⇒ 4 * 1 = 4 B
  char school [4]; ⇒ 1 * 4 = 4 B

```
my_address
  no
  street
  area
  city
  code
```

```
personal_info
  birth_year
  name
  ID
  school
```
5.4 Using Structures

<struct_variable_name> '.' <item_name>

• Components:
  - Struct variable name: <struct_variable_name>
    - must be declared prior to usage
  - Item name: <item_name>
    - must be defined as an item of either the structure template or structure type the variable <struct_variable_name> was declared of

• Example:
  - my_address.no
  - my_address.city
  - personal_info.name
  - personal_info.ID

• Notes:
  - Although allocated memory for, it is not required to use every item of a structured variable
  - C compiler DOES check the names of the structure items
    - e.g. my_address.no1 will cause compiling error
  - A structured type can be used as any other types already defined in C (e.g. declaration of variables, function parameters, etc.)
  - Regular operations with structures include:
    - initialisation and processing
5.5 Initialisation of Structures

- **Types of initialisation:**
  - **At declaration**
    - complete initialisation (all items of the structured variable)
      ```
      personal_type personal_info = {1984, "Ann", 53545367, "EENG"};
      ```
    - partial initialisation (some items only)
      ```
      personal_type personal_info = {1984, "Ann", 53545367};
      ```
  - **During execution**
    - with user input
      - complete initialisation (all items of the structured variable)
        ```
        personal_type personal_info;
        scanf("%d", &personal_info.birth_year);
        scanf("%s", personal_info.name);
        scanf("%d", &personal_info.ID);
        scanf("%s", personal_info.school);
        ```
      - partial initialisation (some items of the structured variable)
        ```
        personal_type personal_info;
        scanf("%d", &personal_info.birth_year);
        scanf("%s", personal_info.name);
        scanf("%d", &personal_info.ID);
        ```
- without user input
  - complete initialisation (all items of the structured variable)

  e.g.
  ```c
  personal_type personal_info;
  personal_info.birth_year = 1984;
  strcpy(personal_info.name, "Ann");
  personal_info.ID = 53545367;
  strcpy(personal_info.school, "EENG");
  ```

- partial initialisation (some items of the of the structured variable)

  e.g.
  ```c
  personal_type personal_info;
  personal_info.birth_year = 1984;
  strcpy(personal_info.name, "Ann");
  personal_info.ID = 53545367;
  ```
• Example:

```c
#include <stdlib.h>

/* structured type definition */
typedef struct address_tag
{
    int no;
    char street[20];
    char area[20];
    char city[20];
    int code;
} address_type;

/* variable declaration of structured type */
address_type my_address;

int main()
{
    ...
    /* initialisation */
    my_address.no = 154;
    strcpy(my_address.street, "Oak Road");
    strcpy(my_address.area, "Glasnevin");
    strcpy(my_address.city, "Dublin");
    my_address.code = 9;
    ...
    return (EXIT_SUCCESS);
}
```

<table>
<thead>
<tr>
<th>no</th>
<th>154</th>
</tr>
</thead>
<tbody>
<tr>
<td>street</td>
<td>Oak Road</td>
</tr>
<tr>
<td>area</td>
<td>Glasnevin</td>
</tr>
<tr>
<td>city</td>
<td>Dublin</td>
</tr>
<tr>
<td>code</td>
<td>9</td>
</tr>
</tbody>
</table>
5.6 Nested Structures

- **Idea:**
  - structures could be embedded in other structures in a tree-like fashion
  - there is no limit related to the depth of this tree

```
Student
  Name
  Home_address
  Personal_info
```

- **Usage:**

  `<struct_prefix> '.' <item_name>`

  - Struct prefix: `<struct_prefix>`

  `<struct_prefix> = <struct_variable_name> | <struct_member_name> '.' <struct_prefix>`

  - Struct member: `<struct_member_name>`

  `<struct_member_name> = <struct_variable_name>`

  - must be a member of the outer structure
• Example:

```c
#include <stdlib.h>

/* structured type definition */
typedef struct address_tag
{
    int no;
    char street[20];
    char area[20];
    char city[20];
    int code;
} address_type;

typedef struct personal_tag
{
    int birth_year;
    char name [20];
    int ID;
    char school [4];
} personal_type;

/* nested structured type definition */
typedef struct student_tag
{
    char Name[20];
    address_type Home_address;
    personal_type Personal_info;
} student_type;
```
int main()
{
    student_type my_student;

    printf(“Welcome to program!”);

    printf(“
Enter student details:”);

    printf(“
Name: ”);
    scanf(“%s”, my_student.Name);

    printf(“
Address:”);

    printf(“
No:”);
    scanf(“%d”, &my_student.Home_address.no);

    printf(“
Street:”);
    scanf(“%s”, my_student.Home_address.street);

    ...

    printf(“
Code:”);
    scanf(“%d”, &my_student.Home_address.code);

    printf(“
Personal info:”);

    printf(“
Birth year:”);
    scanf(“%d”, &my_student.Personal_info.birth_year);

    ...

    printf(“
School:”);
    scanf(“%s”, my_student.Personal_info.school);

    ...

    return (EXIT_SUCCESS);
}
5.7 Arrays of Structures

- **Declaration**

1) "struct" <struct_tag_name> 
   <struct_array_name> '[' <array_size> ']' ';

- **E.g.**
  
  - struct address_template my_address [20];
  - struct personal_template personal_info [30];

2) <struct_type_name> 
   <struct_array_name> '[' <array_size> ']' ';

- **E.g.**
  
  - address_type my_address [20];
  - personal_type personal_info [30];

- **Effect**

<table>
<thead>
<tr>
<th>Item 0</th>
<th>Item 1</th>
<th>Item 19</th>
</tr>
</thead>
<tbody>
<tr>
<td>no street</td>
<td>no street</td>
<td>no street</td>
</tr>
<tr>
<td>area</td>
<td>street area</td>
<td>area</td>
</tr>
<tr>
<td>city</td>
<td>city</td>
<td>city</td>
</tr>
<tr>
<td>code</td>
<td>code</td>
<td>code</td>
</tr>
</tbody>
</table>

my_address [0], my_address[1], ... my_address[19]

- **Usage:**

  my_address[1].area  
  my_address[19].code
• Example:

```c
#include <stdlib.h>
#include <stdio.h>

/* structured type definition */
typedef struct personal
{
    int birth_year;
    char name [20];
    int id;
    char school [4];
} personal_type;

int main()
{
    /*array declaration of structured type */
    personal_type persons[3];
    int i;
    /* initialise the items of the array */
    for (i=0; i<3; i++)
    {
        printf ("YoB=");
        scanf("%d", &persons[i].birth_year);
        printf ("Name=");
        scanf("%s", persons[i].name);
        printf ("ID=");
        scanf("%d", &persons[i].id);
        printf ("School=");
        scanf("%s", persons[i].school);
    }
    /* print the records */
    for (i=0; i<3; i++)
    {
        printf("Personal Info for person %d\n", i);
        printf("YOB =%d\n", persons[i].birth_year);
        printf("Name=%s\n", persons[i].name);
        printf("id=%d\n", persons[i].id);
        printf("School=%s\n", persons[i].school);
    }
    return (EXIT_SUCCESS);
}
```
5.8 Structures as Function Parameters

• **Notes:**
  - Structures are passed as parameters to any function using “by-address” strategy
  - In consequence functions receive only the reference to the beginning of the contiguous memory zone allocated for the structure
  - As it is clear what are the structure’s components, there is no need to indicate to the functions the size of the structure, its components or their number

• **Example:**
  - Function that fills the fields of a structure:

```c
void read_struct(address_type* my_addr)
{
    printf("Enter address:");
    printf("\nNo:");
    scanf("%d", &my_addr->no);
    printf("\nStreet:");
    scanf("%s", my_addr->street);
    ...
    printf("\nThank you!");
}

int main()
{
    address_type my_addr;
    ...
    read_struct(&my_addr);
    ...
    return (EXIT_SUCCESS);
}
```
5.9. Dynamic Memory Allocation

- Notes:
  - Sometimes there is no guarantee that a structure is going to be needed
  - In consequence it is not worthy to allocate memory anyway and not to used it in some situations
  - The solution is to **dynamically allocate memory** only when needed

- Steps:
  - Allocate the memory when needed:
    ```c
#include <stdlib.h>
void* malloc (size_t size);
    ```
  - Free the memory after usage:
    ```c
#include <stdlib.h>
void free (void* ptr);
    ```

- Notes:
  - `malloc()` returns a pointer to the allocated memory or 0 if unsuccessful
  - As the return pointer points towards a generic memory zone, type conversion (“casting”) is required prior to usage
    ```c
(address_type*) malloc(...);
    ```
  - A location of memory has to be secured to store the returned pointer for further use
    ```c
address_type* my_ptr;
my_ptr = (address_type*) malloc(...);
    ```
• `sizeof(operand)` that returns the amount of storage (in bytes) required to store an object of the type of the operand could be used when performing the allocation:

• header of function `sizeof`

```c
size_t sizeof(<type>);
```

```c
my_ptr = (address_type*) malloc (sizeof(address_type));
```

• `free()` de-allocates the allocated memory given the pointer towards it

```c
free(my_ptr);
```

• **Usage:**

  • The content of the allocated memory pointed by the `my_ptr` pointer is accessed by:

    ```c
    (*my_ptr)
    ```

  • As this is a structure, its components are accessed by:

    ```c
    (*my_ptr).street
    ```

  • A shorter version of this notation is:

    ```c
    my_ptr->street
    ```

**Example:**

```c
address_type* ptr;
ptr = (address_type*) malloc (sizeof(address_type));
```

```
<table>
<thead>
<tr>
<th>Mem Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td></td>
</tr>
</tbody>
</table>
```

```c
ptr = (address_type*) malloc (sizeof(address_type));
```

```
<table>
<thead>
<tr>
<th>Mem Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>9000</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Mem Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>9000</td>
<td></td>
</tr>
<tr>
<td>9001</td>
<td></td>
</tr>
</tbody>
</table>
```

```
ptr ->no = (*ptr).no
ptr ->street = (*ptr).street
```
• Example:

```c
#include <stdlib.h>
/* structured type definition */
typedef struct address_tag
{
    int no;
    char street[20];
    char area[20];
    char city[20];
    int code;
} address_type;

int main()
{
    address_type* ptr;
    int answer;

    printf("Enter address (y=1/n=0)?");
    scanf("%d", &answer);

    if (answer == 1)
    {
        /* allocate space for structure */
        ptr = (address_type*) malloc
               (sizeof(address_type));

        /* fill data in the structure */
        printf("No:"aise);    scanf("%d", &ptr->no);

        printf("Street:"aise);    scanf("%s", ptr->street); /* (*ptr).street */
        ...
        /* process the structure */
        ...
        /* de-allocate the space */
        free (ptr);
    } else
        ptr = NULL;
    return (EXIT_SUCCESS);
}
```
5.10. Complex Example

- Example:

  - A file named “grades.txt” stores the following information in text format:
    
    | student_id | grade_s1 | grade_s2 |
    |------------|----------|----------|

  - The data represents student IDs and student grades for two subjects

  - There are no more than 5000 records

  - Create a structure named stud_type to store information about a student with the following fields:
    
    | id | ee102 | ee105 | avg |
    |----|-------|-------|-----|

  - Create an array to store information related to all students and call it arr_stud[]

  - Read the student IDs, grades for the first subject and grades for the second subject and store them in the array arr_stud[]

  - For each student process the grades stored in the fields: ee102 and ee105, compute their average and store the result in the field avg

  - Print on the screen the results for all the students

  - Write data in the following format in a new file called “results.txt”
    
    | student_id | grade_s1 | grade_s2 | average |
    |------------|----------|----------|---------|

  - Use functions
#include <stdio.h>
#include <stdlib.h>

/* structured type definition */

typedef struct stud_type_tag
{
    int id;
    int ee102;
    int ee105;
    float avg;
} stud_type;

/* declarations */

/* function that reads data from a file, stores it in an array and returns the number of records read */
int read_records (char* file_name, stud_type stud[]);

/* function that computes the average for each student given the array with stud info and stores the results in the same array */
void compute_average (stud_type stud[], int size);

/* function that writes data taken from the array into a file */
void write_records (char* file_name, stud_type stud[], int size);

/* function that prints the studentIDs, grades and their average marks */
void print_results (stud_type stud[], int size);
/* main function */

int main()
{

    /* 1-dimensional array that stores student related data for all registered students */
    stud_type stud[5000];

    /* no current registered students (no records) */
    int norecs = 0;

    /* welcome message */
    printf("Welcome to the program!\n");

    /* read records */
    norecs = read_records("grades.txt", stud);

    /* compute the averages */
    compute_average(stud, norecs);

    /* write data to a file */
    write_records("results.txt", stud, norecs);

    /* print results */
    print_results(stud, norecs);

    printf("Goodbye!\n");

    return (EXIT_SUCCESS);
}
/* function definitions */

int read_records (char* file_name, stud_type stud[]) {
  FILE* fp;
  int norecs = 0;
  char ch;

  /* open file */
  fp = fopen(file_name, "r");

  while(!feof(fp)) {
    /* read on record from file */
    fscanf(fp, "%d%c%d%c%d%c",
          &stud[norecs].id, &ch,
          &stud[norecs].ee102, &ch,
          &stud[norecs].ee105, &ch);

    /* increase the number of records read */
    norecs++;
  }

  /* close file */
  fclose(fp);

  return norecs;
}

void compute_average (stud_type stud[], int size) {
  int i;

  /* parse grades and compute the average */
  for (i = 0; i < size; i++)
    stud[i].avg = (stud[i].ee102 +
                   stud[i].ee105)/2.0;
}
void write_records (char* file_name, 
        stud_type stud[], int size)
{
    FILE* fp;
    int i;

    /* open file */
    fp = fopen(file_name, "w");

    for (i = 0; i < size; i++)
      /* read on record from file */
      fprintf(fp, "%d %d %d %f\n",
              stud[i].id, stud[i].ee102, 
              stud[i].ee105, stud[i].avg);

    /* flush content into file */
    fflush(fp);

    /* close file */
    fclose(fp);
}

void print_results (stud_type stud[], int size)
{
    int i;

    printf("| StudID |ee102|ee105| Avg |\n");

    /* parse data and print grades and average */
    for (i = 0; i < size; i++)
      printf("|%8d|%5d|%5d|%5.2f|\n",
              stud[i].id, stud[i].ee102, 
              stud[i].ee105, stud[i].avg);
}