1. Objective.................................................................2
2. Structure Definition ..............................................2
3. Struct Syntax........................................................3
4. Manipulating Structure Types.................................6
5. Arrays of Structures................................................8
6. Hierarchical structure..............................................12
7. Function with a Structure Input Parameter ..............15
8. Questions/Practice .................................................16
1. Objective
   - What is C structure?
   - When to use structures.
   - Syntax of a structure.
   - How to declare variable of type structure?
   - Fields of a structure and how to initialize them.
   - How to manipulate structure type

2. Structure Definition
   - A Structure is a collection of related data items, possibly of different types.
   - Structures are also called records.
   - A structure type in C is called struct.
   - Unlike arrays, a struct is composed of data of different types.
   - You use structures to group data that belong together.
   - Examples:
     - Student information:
       - student id,
       - last name,
       - first name
       - major,
       - gender,
       - …
Bank account information:
  - account number,
  - account type
  - account holder
    - first name
    - last name
  - balance

- Data elements in a structure are called **fields** or **members**
- Complex data structures can be formed by defining arrays of structs.

### 3. Struct Syntax
- Syntax of the structure type:

```c
typedef struct{
    type1 id1;
    type2 id2;
    ...
} struct_name;
```

Required
Example:

- The following structure has three fields:

```c
typedef struct  {
    int day;
    int month;
    int year;
} eventDate;
```

```c
typedef struct {
    char name[20];
    int age;
} person;
```

```c
struct telephone {
    char name[30];
    int number;
};
```
• How to declare variable of type structure?
  o Create a variable, var1, using this structure:
    
    ```c
    typedef struct {
      type1 id1;
      type2 id2;
      ...
    } struct_name;
    
    sturct_name var1;
    ```
  
  o **Examples:**
    
    ```c
    person p1;
    person p2;
    ```
4. Manipulating Structure Types

- **How to access a field in a structure:**
  - Use the **direct component selection operator**, which is a **period**.
  - The **direct component selection operator** has the **highest priority** in the operator precedence.
  - Examples:
    - `person p1;`
    - `p1.name;`
    - `p1.age;`

- **Structure assignment:**
  - The copy of an entire structure can be easily done by the assignment operator.
  - Each component in one structure is copied into the corresponding component in the other structure.
- Examples:
  - Given the following structure:
    ```c
    typedef struct  {
        int day;
        int month;
        int year;
    } eventDate;
    ```
    ```c
    eventDate ev1;
    ev1.day = 2; ev1.month = 11; ev1.year = 2017;
    eventDate ev2 = { 20, 10, 2017};
    ```
  - Given the following structure
    ```c
    typedef struct {
        char name[20];
        int age;
    } person;
    ```
    ```c
    person p1 = { “Mary”, 24};
    person p2;
    strcpy(p2.name, "Paul");
    p2.age = 27;
    ```
5. Arrays of Structures

- We can also declare an array of structures.
- Recall the syntax of an array:
  
  ```c
  type array_name[size];
  ```

  type can any C type including struct type.

- The array of structures can be simply manipulated as **arrays** of simple data types.

- Example:

  ```c
  typedef struct {
    char name[20];
    int age;
  } person;
  
  person class[5];
  
  strcpy(class[0].name, “John”);
  class[0].age = 19;
  strcpy(class[1].name, “Sara”);
  class[1].age = 18;
  ```
• Application: Bank Account
  • We would like to write a C program that stores customers at a bank.
  • Here are the different steps:
    ▪ Step 1: design your structure
      ```c
      typedef struct {
          int account_number;
          char account_type[20];
          char account_holder_name[40];
          double balance;
      } bank_account;
      ```
    ▪ Step 2: declare the array of structs
      ```c
      bank_customer bank_customers[100];
      ```
- **Step 3: Initialization**

  ```
  bank_customers[0].account_number = 1001;
  strcpy(bank_customers[0].account_type, "Checking");
  strcpy(bank_customers[0].account_holder_name, "John Paul");
  bank_customers[0].balance = 2100.50;
  ```

- **Full Program:**

  ```
  //gcc 5.4.0

  #include <stdio.h>

  int main(void)
  {
      typedef struct {
          int account_number;
          char account_type[20];
          char account_holder_name[40];
          double balance;
      } bank_customer;
  ```
bank_customer bank_customers[5];

bank_customers[0].account_number = 1001;
strcpy(bank_customers[0].account_type, "Checking");
strcpy(bank_customers[0].account_holder_name, "John Paul");
bank_customers[0].balance = 2100.50;

for(int i=0;i<5;++i){

    printf("Account Number: %d
Account Type: %s
Account Holder Name: %s
Balance: %.2lf
-----------
",
    bank_customers[i].account_number, bank_customers[i].account_type, bank_customers[i].account_holder_name, bank_customers[i].balance);
}
}
6. Hierarchical structure

- A hierarchical structure is a structure containing components which are also structures.
- Let us review the description of the bank account of a customer:
  - Here is the initial definition:
    ```
    typedef struct {
        int account_number;
        char account_type[20];
        char account_holder_name[40];
        double balance;
    } bank_account;
    ```
  - Now, let us break the name of a customer into:
    - First name
    - Last name
  - Let us define a name structure as follows:
    ```
    typedef struct {
        char first_name[20];
        char last_name[20];
    } customer_name;
    ```
Here is the updated program:

```c
#include <stdio.h>
#include <string.h>
int main(void)
{
    typedef struct {
        char first_name[20];
        char last_name[20];
    } customer_name;

typedef struct {
    int account_number;
    char account_type[20];
    customer_name account_holder_name;
    double balance;
} bank_customer;

bank_customer    bank_customers[5];

bank_customers[0].account_number = 1001;
strcpy(bank_customers[0].account_type, "Checking");

strcpy(bank_customers[0].account_holder_name.first_name, "John");

strcpy(bank_customers[0].account_holder_name.last_name, "Paul");
```
bank_customers[0].balance = 2100.50;

bank_customers[1].account_number = 1002;
strcpy(bank_customers[1].account_type, "Checking");

strcpy(bank_customers[1].account_holder_name.first_name, "Mary");

strcpy(bank_customers[1].account_holder_name.last_name, "Paul");
bank_customers[1].balance = 30100.50;

for(int i=0; i<5; ++i){
    printf("Account Number: %d
Account Type: %s
Account Holder First Name: %s
Account Holder Last Name: %s
Balance: %.2lf
--------
\n", bank_customers[i].account_number, bank_customers[i].account_type,
        bank_customers[i].account_holder_name.first_name,
        bank_customers[i].account_holder_name.last_name,
        bank_customers[i].balance);
}
7. Function with a Structure Input Parameter

- When a structure variable is passed as an input argument to a function, all its component values are copied into the local structure variable.

- Example:

```c
#include <stdio.h>

struct student
{
    int id;
    char name[20];
    char grade;
};

void func(struct student stud);

int main()
{
    struct student astud;

    astud.id = 9401;
    strcpy(astud.name, "Joe");
    astud.grade = 'A';

    func(astud);
    return 0;
}
```
8. Questions/Practice

- Write a C program that implement complex numbers (see solution below)
- Write a C program that implement fractions (Lab)
- Modify the last program in section 7 to include the address of a student as a separate structure. The address should include the following:
  - Address as an array of 30 characters
  - City as a an array of 20 characters
  - Zipcode as an integer.
9. Partial Solution

- Complex numbers:

```c
//gcc 5.4.0

#include <stdio.h>

typedef struct {
    double realpart;
    double imaginarypart;
} complex;

complex addcomp (complex a, complex b){
    complex addc;
    addc.realpart = a.realpart + b.realpart;
    addc.imaginarypart = a.imaginarypart + b.imaginarypart;

    return (addc);
}
complex subcomp (complex a, complex b){
    complex addc;
    addc.realpart = a.realpart - b.realpart;
    addc.imaginarypart = a.imaginarypart - b.imaginarypart;

    return (addc);
}

int main(void) {
```
complex c1;
complex c2;
complex c3;
c1.realpart = 2.3;
c2.realpart = 2.3;
c1.imaginarypart = 2.3;
c2.imaginarypart = 2.3;
c3 = addcomp(c1, c2);
printf("%.2lf+%.2lfi + %.2lf+%.2lfi = %.2lf+%.2lfi\n",
c1.realpart, c1.imaginarypart, c2.realpart,
c2.imaginarypart, c3.realpart, c3.imaginarypart);
c1.imaginarypart = 2.3;
c2.imaginarypart = 2.3;
c3 = subcomp(c1, c2);
printf("%.2lf+%.2lfi - %.2lf+%.2lfi = %.2lf+%.2lfi\n",
c1.realpart, c1.imaginarypart, c2.realpart,
c2.imaginarypart, c3.realpart, c3.imaginarypart);
return 0;
}