typedef

- It allows us to introduce synonyms for data types which could have been declared some other way.
- It is used to give New name to the Structure.
- New name is used for Creating instances, declaration etc…

Basic example

```
#include<stdio.h>
int main()
{
typedef int Number;
Number num1 = 40,num2 = 20;
Number answer;

answer = num1 + num2;
printf("Answer : %d“, answer);
return(0);
}
//Answer : 60
```

In this program we have used typedef to create alias name to data type which is ‘int’ in this example and new name to integer data type is now ‘Number’.

typedef with structures

```c
#include<stdio.h>
typedef struct b1 {
   char bname[30];
   int ssn;
   int pages;
}book;
book b1 = {"Let Us C",1000,90};
int main()
{
    printf("\nName of Book : %s",b1.bname);
    printf("\nSSN of Book : %d",b1.ssn);
    printf("\nPages in Book : %d",b1.pages);
    return(0);
}
```

```
Name of Book  : Let Us C
SSN  of Book  : 1000
Pages in Book : 90
```

Functions

Basics

- C allows programmers to define their OWN functions for carrying out specific task.
- The use of programmer-defined functions allows a large program to be broken down into smaller, self-contained components, each of which has some unique, identifiable purpose.
- The use of function avoids the need for redundant programming of same set of instructions.

What is a function?

- A User-defined functions is a self-contained program segment that carries out well defined specific task.

- Such function once defined can be called any number of times from any program.

- It allows user to build his own customized library of frequently used routines which will avoid redundancy and enhance modularity of program.

Classification

- Library Functions: Predefined functions which are meant for specific task. To use these functions in program one should include the appropriate header file.e.g. printf, scanf etc

- Programmer-defined Functions: Functions which are defined by user for a particular operation.

The main Function

- The main Function is the only function which does not need to be declared or called,as every C program must have a one and main is called by the operating system

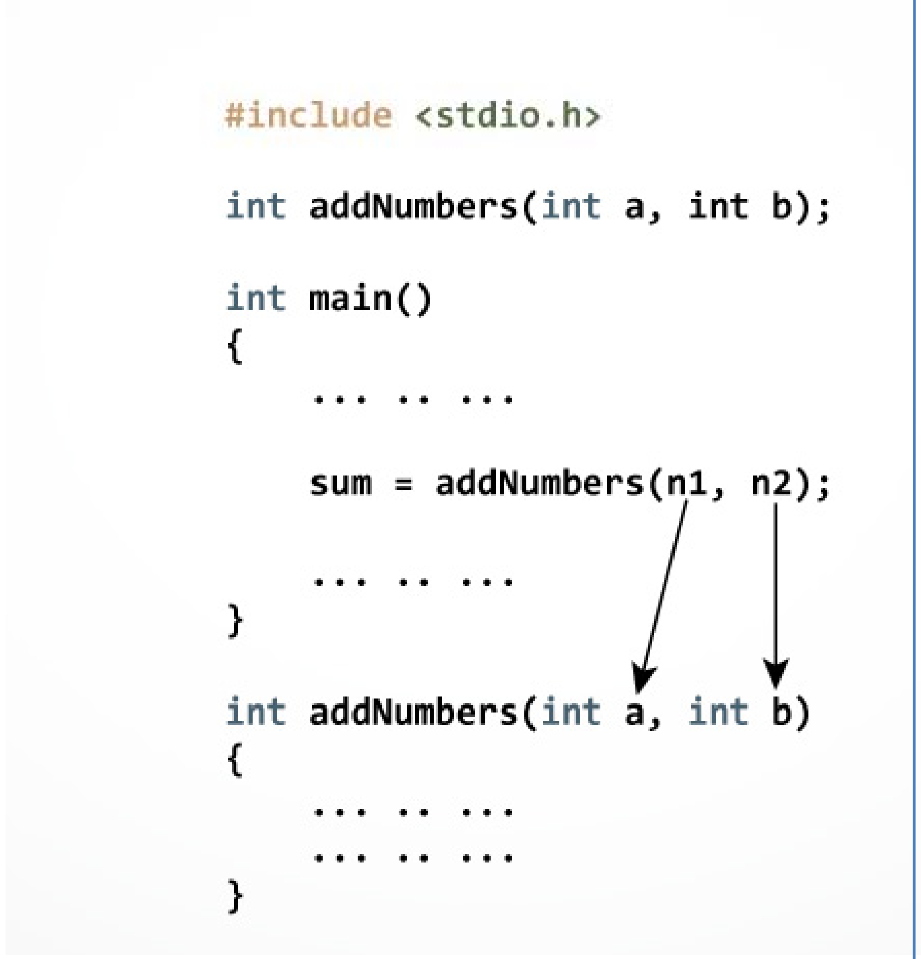
Program to calculate sum of 3 numbers

```
#include<stdio.h>
int calsum (int , int , int );          ←————————   Function declaration or
main( )                                             prototype (User defined
{                                                   function)
    int a, b, c, sum;
    scanf(“%d %d %d”, &a, &b, &c);

   sum = calsum(a, b, c);               ←————————   Function call
    printf(“\n\n Sum = %d”, sum);
}
int calsum (int x, int y, int z)        ←————————   Function definition
{ int d;
   d = x + y + z;
   return(d);
}
```

Program to calculate sum of 2 numbers

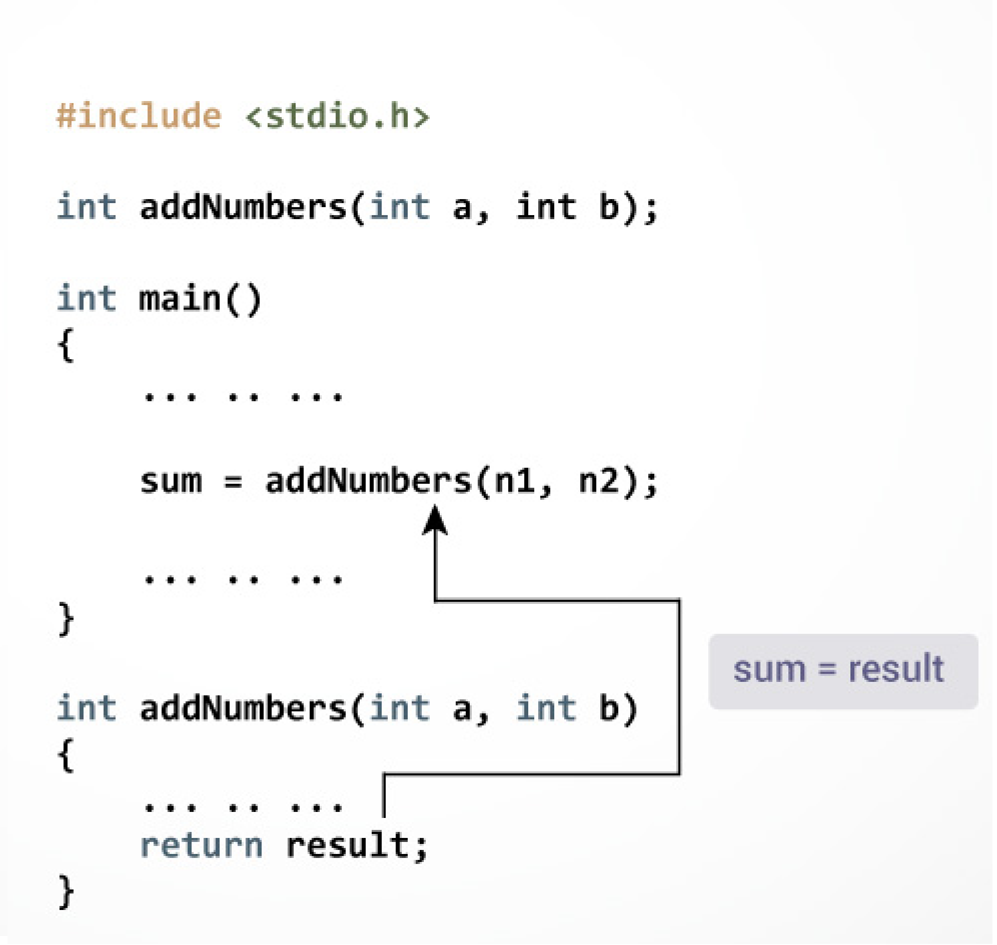
```
#include <stdio.h>
int addNumbers(int , int );          // function prototype
int main() {
   int n1,n2,sum;
   printf("Enters two numbers: ");
   scanf("%d %d",&n1,&n2);
   sum = addNumbers(n1, n2);       // function call
   printf("sum = %d",sum);
   return 0;
}
int addNumbers(int a, int b)        // function definition
{
   int result;
   result = a+b;
   return result;                   // return statement
}
```

How to pass arguments to a function?



```c
#include <stdio.h>

int addNumbers(int a, int b);

int main()
{
    ... .. ...

    sum = addNumbers(n1, n2);

    ... .. ...
}

int addNumbers(int a, int b)
{
    ... .. ...
    ... .. ...
}
```

Return statement of a Function



```c
#include <stdio.h>

int addNumbers(int a, int b);

int main()
{
    ... .. ...

    sum = addNumbers(n1, n2);

    ... .. ...
}

int addNumbers(int a, int b)
{
    ... .. ...
    return result;
}
```

- The three important things regarding a functions :
  - A function can appear in a program in three ways, as a Declaration , Call and Definition.
  - A function has three attributes, the Return type, Identifier (or name) and Parameter List.
  - The Body of a function is represented by a code block (not optional) and has local declarations, expressions and a return type.

Declaration

- A function has to be declared in the program before it is called.
- Function declaration is called "PROTOTYPE" of the function and consist of three attributes :
  - » Return type
  - » Identifier (name)
  - » Parameter list.

- General syntax of prototype:

  return type name (parameter list) ;

Return type

- It is a way of sending data back to the “calling” function.
- Using “ return” keyword called function can send manipulated data back to the calling function.
- When no information or data is to be sent back to the calling function return type of called function is set to “void” which means nothing.
- Function can return integer, character, float or double, pointer.

Function call

- A call to a function is made when ever we want to execute the code of the function in program.
- Example

```
int main ( void )
{
        ……………..
        funct1(  ) ;
        return 0;
 }
```

$\xrightarrow{\hspace{3cm}}$ Call to function

Accessing a Function

- A function is called by specifying its name, followed by the actual arguments in parenthesis.
- Each actual argument must be of the same data type as its corresponding formal argument.
- Control returns to the point of call.

Example

Program to calculate sum of three integer quantities

```
#include<stdio.h>
int calsum (int , int , int );
main( )
{
     int a, b, c, sum;
     scanf("%d %d %d", &a, &b, &c);

    sum = calsum(a, b, c);
     printf("\n\n Sum = %d", sum);
}
int calsum (int x, int y, int z)
{ int d;
   d = x + y + z;
   return(d);}
```

Actual arguments

Formal Arguments

Function definition

- The Function Definition contains the same information as the declaration; return type, name and parameter list.
- The Function Definition also contains a block which contains the function’s code.

Syntax of definition

```
return_type identifier ( parameter_list )
                              {
                                        local declarations;
                                        local statements;
                                        return return_value;
                              }
```

Example:

```
        int  maximum( int x, int y)
                        {       int z;
                                 z=(x>=y) ? x : y ;
                                return z;
                         }
```

Write a program to calculate greatest of two numbers using function.

```
# include<stdio.h>
                                        Function declaration
int check( int , int ) ;

int main(void)
{          int a,b,z;  /* local variable definition */
           //prompt the user to input 2 numbers
           z= check( a, b) ;          Function call to get max value
           printf("The greatest of two numbers =%d",z);
}
int check( int a1, int b1)            Function definition
{ int a; /* local variable declaration */
  a = ( a1>= b1) ? a1 : b1 ;
  return a;   }
```

Function definition inside declaration or prototype

```
# include<stdio.h>
int check( int , int )
{ int a; /* local variable declaration */
  a = ( a1>= b1) ? a1 : b1 ;
  return a;
}
int main(void)
{           int a,b,z;  /* local variable definition */
            //prompt the user to input 2 numbers
            z= check( a, b) ;
            printf("The greatest of two numbers =%d",z);
}
```

Function definition inside function declaration

Function call to get max value

Calling a function multiple times

```
#include<stdio.h>
int square( int); /* function prototype */
int main()
{
int x;
printf("the squares of numbers from 1 to 10 are:\n");
 for(x=1 ;x <= 10; x++)
{
 y = square(x);   /*function call */
printf("the sqare of %d = %d\n",x, y); }
return 0;    }

/*function definition */
int square (int a)
   {
 int b;
   b = a * a;
return  b;
}
```

Class exercise

1. Write a program to check whether the number is prime or not using function.
2. Write a program to compute square of a given number using function.
3. Write a program to generate prime numbers for a given range using function.

Write a program to convert a lowercase character to uppercase using a programmer defined function

```
# include<stdio.h>
                                                Function declaration
char lower_to_upper( char ) ;        ←

int main(void)
{          char lower, upper;
printf(“please enter a lowercase character:”);
scanf(“%c”,&lower);
           upper=lower_to_upper(lower) ; ←  Function call
printf(“The uppercase equivalent is %c”,upper);
}

char lower_to_upper(char c1) ←         Function definition
{ char c2;
  c2 = ( c1>=‘a’&&c1<=‘z’) ? (‘A’+c1-’a’) : c1 ;
  return c2;   }
```

Function calls in C

- There are two ways that a C function can be called from a program. They are,
  - Call by value
  - Call by reference

Call by value and reference

- In call by value method, the value of the variable is passed to the function as parameter.
- The value of the actual parameter can not be modified by formal parameter.
- Different Memory is allocated for both actual and formal parameters. Because, value of actual parameter is copied to formal parameter.

- In call by reference method, the address of the variable is passed to the function as parameter.
- The value of the actual parameter can be modified by formal parameter.
- Same memory is used for both actual and formal parameters since only address is used by both parameters.

Call by value example

```
void swap(int a, int b);
int main() {
   int m = 22, n = 44;
   printf(" values before swap  m = %d \nand n = %d", m, n);
   swap(m, n);
}
void swap(int a, int b)
{
   int tmp;
   tmp = a;
   a = b;
   b = tmp;
   printf(" \nvalues after swap m = %d\n and n = %d", a, b);
}
```

```
values before swap m = 22 and n = 44
values after swap m = 44 and n = 22
```

Explanation : Call by Value

- While Passing Parameters using call by value , **xerox copy of original parameter is created** and passed to the called function.
- Any update made inside method will not affect the **original value of variable in calling function**.
- In the above example m and n are the original values and xerox copy of these values is passed to the function and these values are copied into a,b variable of sum function respectively.
- As their scope is limited to only function so they **cannot alter the values inside main function**.

Greatest of two numbers (Revisited): by Call by value

```
#include <stdio.h>
int max(int num1, int num2);
int main () {
  /* local variable definition */
  int a = 100;
  int b = 200;
  int ret;
  ret = max(a, b);     /* calling a function to get max value */
  printf( "Max value is : %d\n", ret );
  return 0; }
/* function returning the max between two numbers */
int max(int num1, int num2) {
  /* local variable declaration */
  int result;
  if (num1 > num2)
    result = num1;
  else
    result = num2;
  return result;  }
```

- Call by reference will be covered in next semester in SDF-II

Passing structure to functions

- A structure can be passed to any function from main function or from any sub function.
- Structure definition will be available within the function only.
- It won’t be available to other functions unless it is passed to those functions by value or by address(reference).
- Else, we have to declare structure variable as global variable. That means, structure variable should be declared outside the main function. So, this structure will be visible to all the functions in a C program.

PASSING STRUCTURE TO FUNCTION IN C:

- It can be done in below 3 ways.
  1. Passing structure to a function by value
  2. Passing structure to a function by address(reference) → Next semester as require use of pointers
  3. No need to pass a structure – Declare structure variable as global

Program

- Write a C program to create a structure student, containing name and roll number.
  - Ask user the name and roll number of a student in main function.
  - Pass this structure to a function and display the information in that function.

Passing structure to a function by value

```
struct student {
        int id;
        char name[20];
        float percentage; };
void func(struct student record);          /* function prototype should be written after
                              structure declaration otherwise complier will show error*/
int main()  {
        struct student record;
         record.id=1;
        strcpy(record.name, "Raju");
        record.percentage = 86.5;
        func(record);                       /*passing structure instance as argument*/
        return 0;
}
void func(struct student record) {
        printf(" Id is: %d \n", record.id);
        printf(" Name is: %s \n", record.name);
        printf(" Percentage is: %f \n", record.percentage); }
```

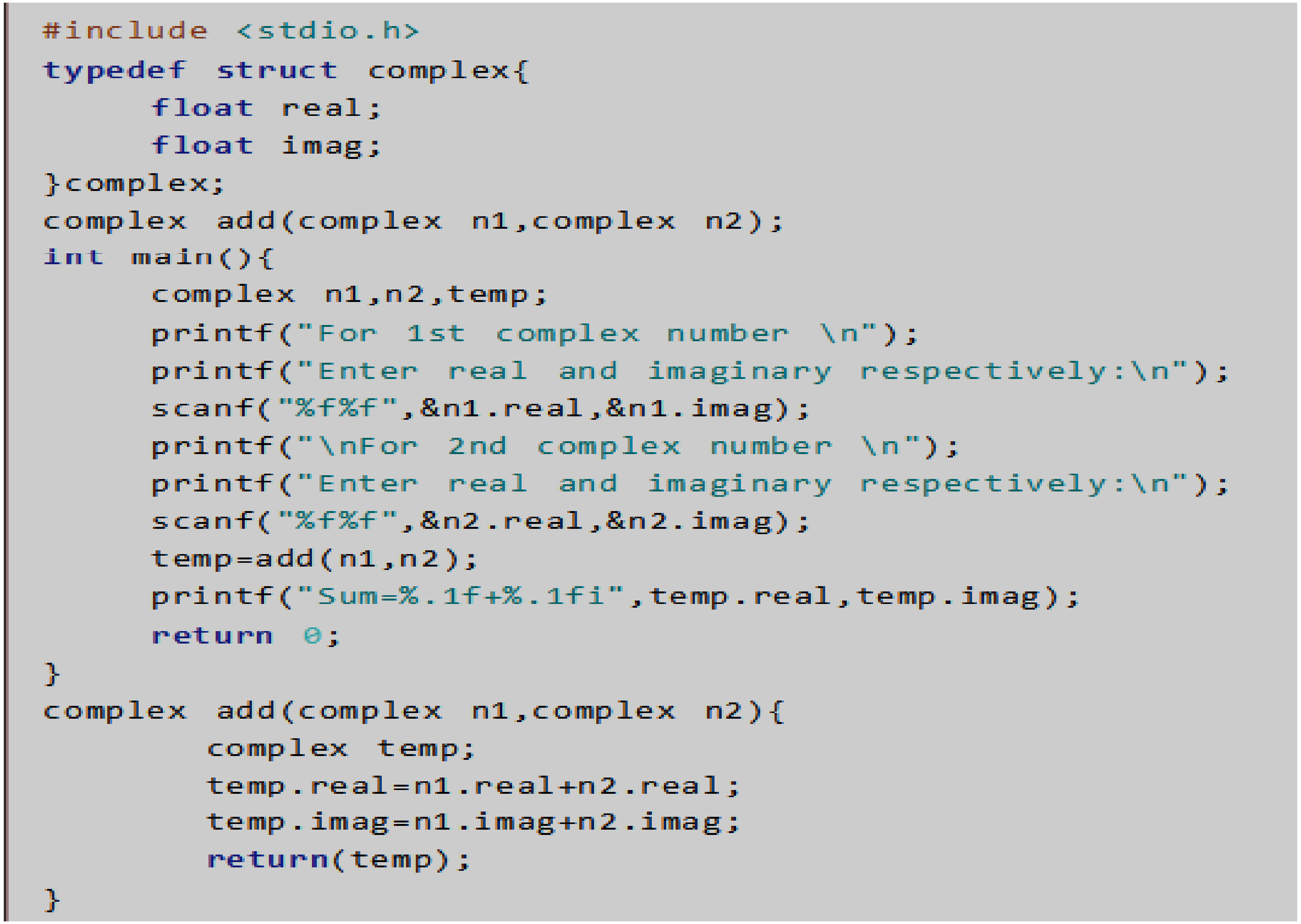
Declare structure variable as global

```
struct student {
        int id;
        char name[20];
        float percentage; };
struct student record; // Global declaration of structure
 void func();
 int main()  {
        record.id=1;
        strcpy(record.name, "Raju");
        record.percentage = 86.5;
         func();
        return 0;
}
 void func() {
        printf(" Id is: %d \n", record.id);
        printf(" Name is: %s \n", record.name);
        printf(" Percentage is: %f \n", record.percentage);
}
```

C Program to Add Two Complex Numbers by Passing Structure to a Function

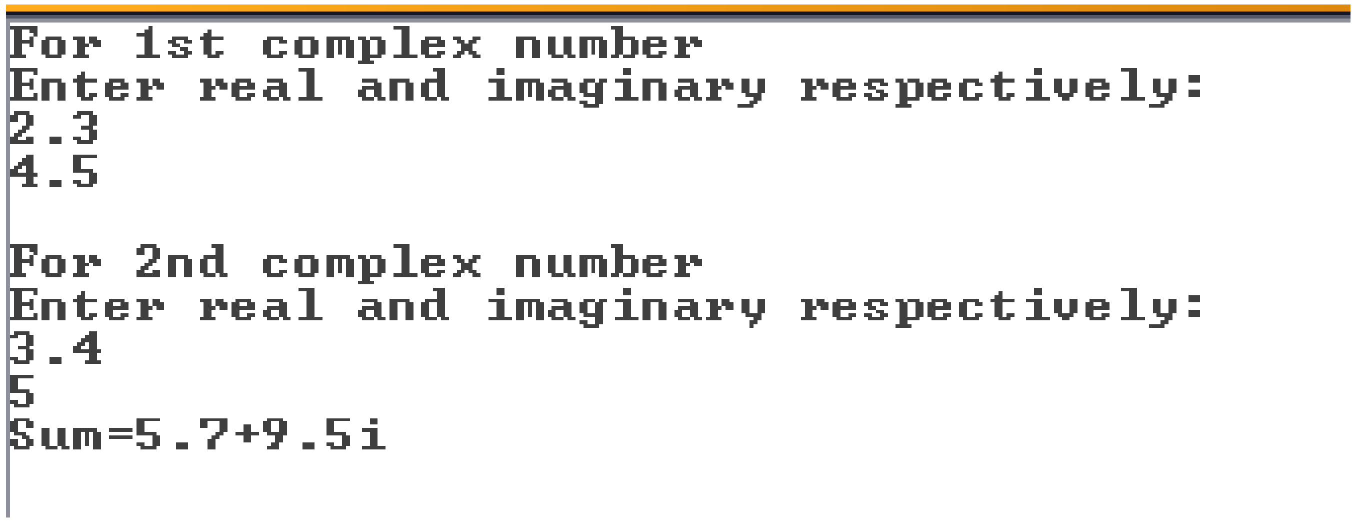
- This program takes two complex numbers having real and imaginary part and store them inside data members of two structure variables.
- Then, this program calculates the sum of two complex numbers by passing it to a function and result is displayed in main() function.

In this program structures *n1* and *n2* are passed as an argument of function add(). This function computes the sum and returns the structure variable *temp* to the main() function.



```c
#include <stdio.h>
typedef struct complex{
    float real;
    float imag;
}complex;
complex add(complex n1,complex n2);
int main(){
    complex n1,n2,temp;
    printf("For 1st complex number \n");
    printf("Enter real and imaginary respectively:\n");
    scanf("%f%f",&n1.real,&n1.imag);
    printf("\nFor 2nd complex number \n");
    printf("Enter real and imaginary respectively:\n");
    scanf("%f%f",&n2.real,&n2.imag);
    temp=add(n1,n2);
    printf("Sum=%.1f+%.1fi",temp.real,temp.imag);
    return 0;
}
complex add(complex n1,complex n2){
      complex temp;
      temp.real=n1.real+n2.real;
      temp.imag=n1.imag+n2.imag;
      return(temp);
}
```

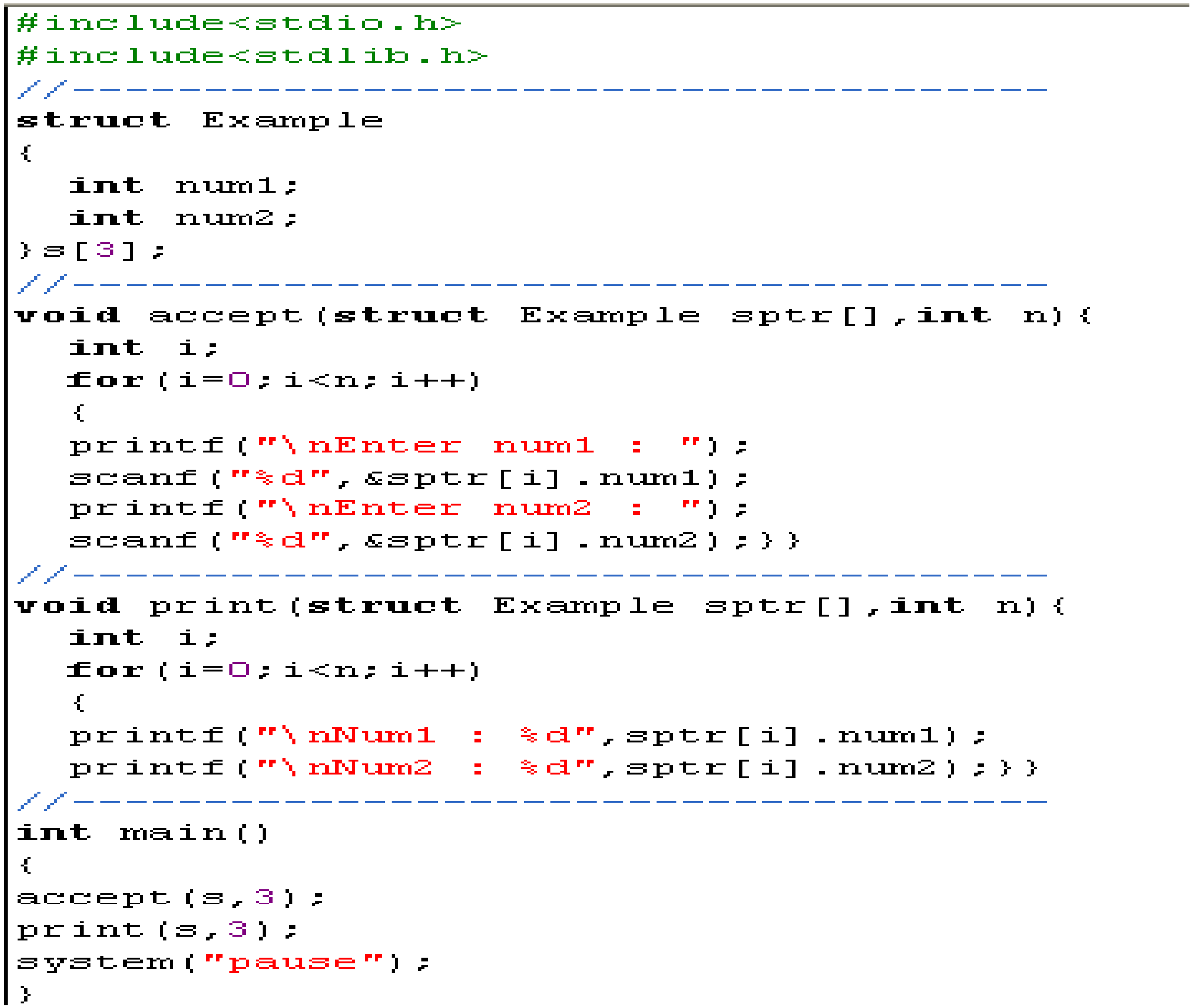
Output



```
For 1st complex number
Enter real and imaginary respectively:
2.3
4.5

For 2nd complex number
Enter real and imaginary respectively:
3.4
5
Sum=5.7+9.5i
```

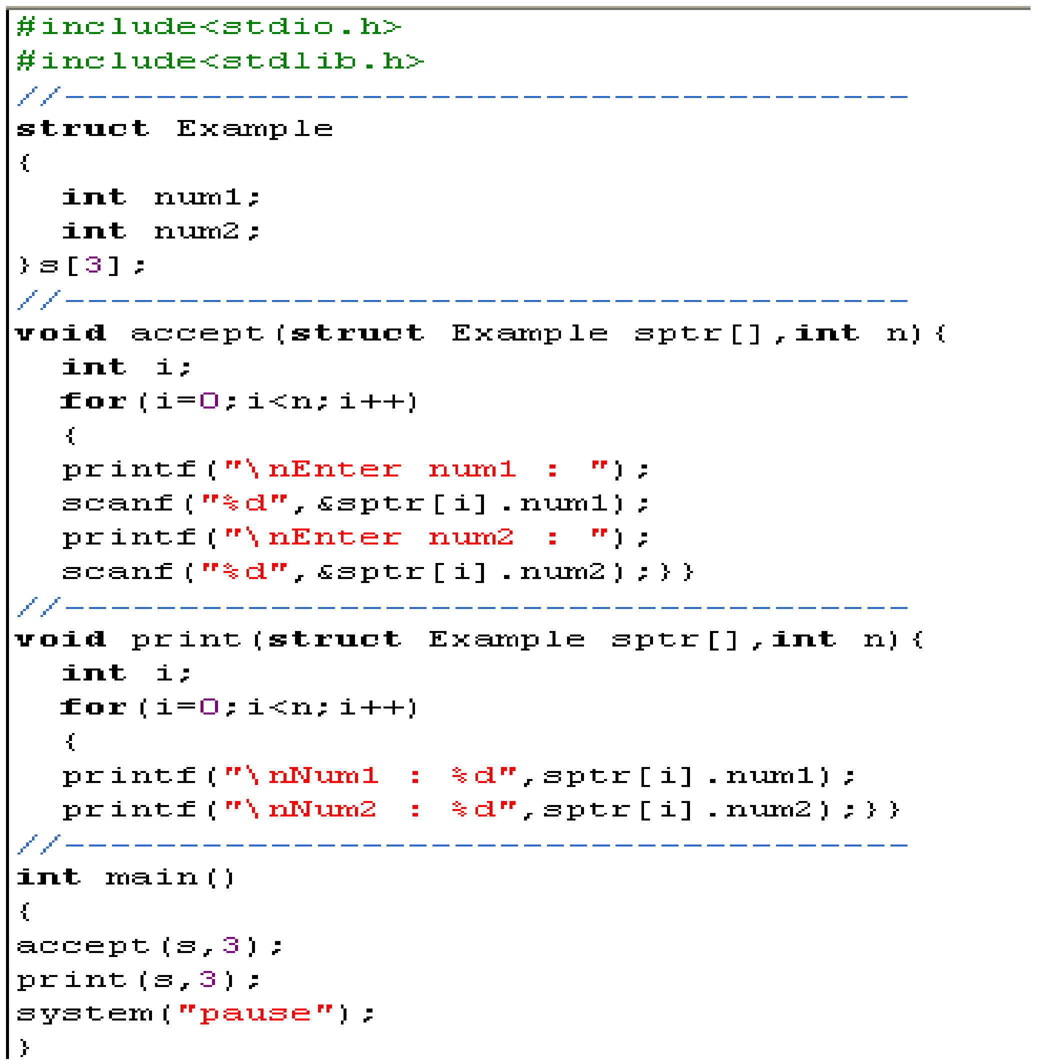
Passing array of structure to function

- Write a program in C that has two member variables number1 and number2 inside a structure.
- Write a function **accept()** takes structure as a argument and number of elements that need to be scanned.
- Write a function **print()** takes structure as a argument and number of elements that need to be printed.
- Create a structure, call the methods from main method.

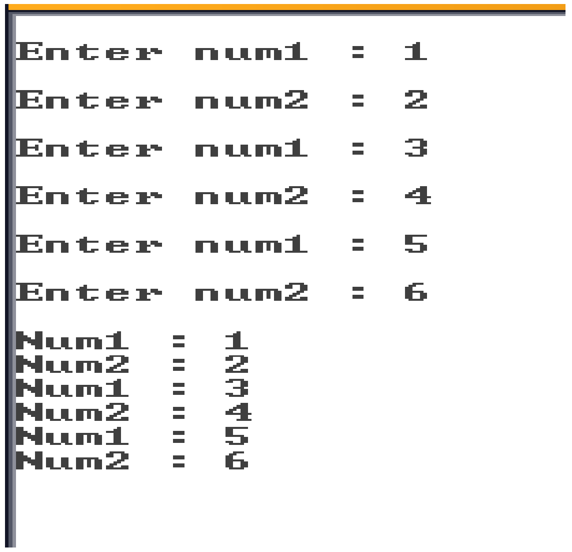


```c
#include<stdio.h>
#include<stdlib.h>
//-----------------------------------------
struct Example
{
    int num1;
    int num2;
}s[3];
//-----------------------------------------
void accept(struct Example sptr[],int n){
    int i;
    for(i=0;i<n;i++)
    {
    printf("\nEnter num1 : ");
    scanf("%d",&sptr[i].num1);
    printf("\nEnter num2 : ");
    scanf("%d",&sptr[i].num2);}}
//-----------------------------------------
void print(struct Example sptr[],int n){
    int i;
    for(i=0;i<n;i++)
    {
    printf("\nNum1 : %d",sptr[i].num1);
    printf("\nNum2 : %d",sptr[i].num2);}}
//-----------------------------------------
int main()
{
accept(s,3);
print(s,3);
system("pause");
}
```

Output



```c
#include<stdio.h>
#include<stdlib.h>
//-------------------------------------
struct Example
{
   int num1;
   int num2;
}s[3];
//-------------------------------------
void accept(struct Example sptr[],int n){
   int i;
   for(i=0;i<n;i++)
   {
   printf("\nEnter num1 : ");
   scanf("%d",&sptr[i].num1);
   printf("\nEnter num2 : ");
   scanf("%d",&sptr[i].num2);}}
//-------------------------------------
void print(struct Example sptr[],int n){
   int i;
   for(i=0;i<n;i++)
   {
   printf("\nNum1 : %d",sptr[i].num1);
   printf("\nNum2 : %d",sptr[i].num2);}}
//-------------------------------------
int main()
{
accept(s,3);
print(s,3);
system("pause");
}
```



```
Enter num1 : 1

Enter num2 : 2

Enter num1 : 3

Enter num2 : 4

Enter num1 : 5

Enter num2 : 6

Num1 : 1
Num2 : 2
Num1 : 3
Num2 : 4
Num1 : 5
Num2 : 6
```

Explanation

- Inside main function : structure and size of structure array is passed.
- When reference (i.e. &ampersand) is not specified in the main , so the passing is simple pass by value.
- Elements are accessed using .dot operator