Module 2: Variables, Data Types

C Variables

- Variables are Containers for Storing data values.
- A variable is nothing but a name given to a storage area thar our program can manipulate.
- Each variable in C has a specific type, which determines the size and layout of the variable's memory
- Ex. a=3; // a is assigned "3:

In C, there are different types of variables.

- An integer defined with the keyword int stores integers (whole numbers), without decimals, such as 19 or -7.
- A floating point variable defined with keyword float stores floating point numbers, with decimals, such as 78.25 or -2.56.
- A character variable defined with the keyword char stores single characters, such as 'A' or 'Z'. char values are bound to be surrounded by single quotes.

Declaration

- We can not declare a variable without specifying its data type.
- The data types of a variable depends on what be want to store in the variable and how much space we want it to hold.
- Syntax for declaring a variable is simple:

```
data_type  variable_name;
```

OR

```
data_type  variable_name = value;
```

Naming a variable:

There is no limit to what we can call a variable. Yet there are specific rules we must follow while naming a variable:

- A variable name can only contain alphabets, digits, and underscores(_).
- A variable cannot start with a digit.
- A variable cannot include any white space in its name.
- The name should not be a reserved keyword or any special character.

A variable, as its name is defined, can be altered, or its value can be changed, but the same is not true for its type.

If a variable is of integer type, then it will only store an integer value through a program.

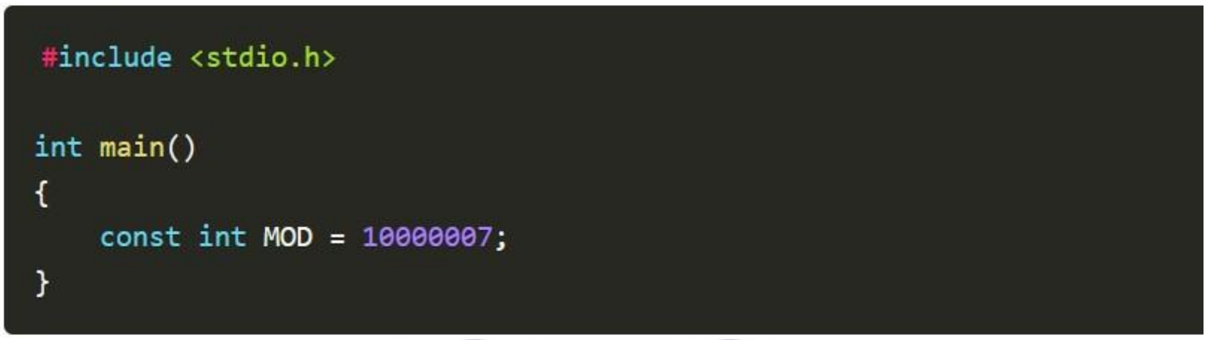
We cannot assign a character type value to an integer variable.

We can not even store a decimal value into an integer variable.

C Constants

- When you don't want the variables you declare to get modified intentionally or mistakenly in the later part of your program by you or others, you use the const keyword (this will declare the variable as "constant", which means unchangeable and read-only).
- You should always declare the variable as constant when you have values that are unlikely to change, like any mathematical constant as PI.
- When you declare a constant variable, it must be assigned a value.

Here’s an example of how we declare a constant.



```
#include <stdio.h>

int main()
{
    const int MOD = 10000007;
}
```

Types of constants:

Primarily, there are three types of constants:

- Integer Constant- 1,-2,7,9
- Real Constant - -322.5, 3.5,4.5
- Charcters constant – ‘a’, ‘%’ (must be enclosed within single inverted comma)

Receiving input from user

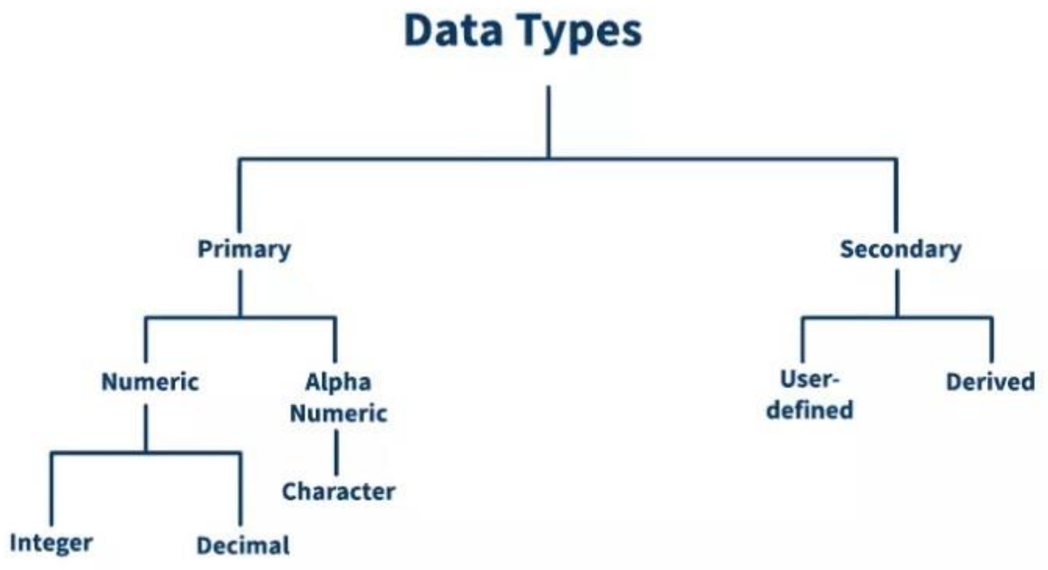
- In order to take input from user and assign it to a variable, we use scanf function.
- Syntax for using scanf: scanf(“%d”,&i);
- ‘&’- this is the “address of” operator and it means that the supplied valued should be copied to the address which is indicated by variable I;

Data Types

- C language is rich in its data types.
- It is the type of value which is the variable holds.
- A data type specifies the type of data that a variable can store such as integer, floating, character, etc.
- The data type specifies the size and type of information the variable will store.

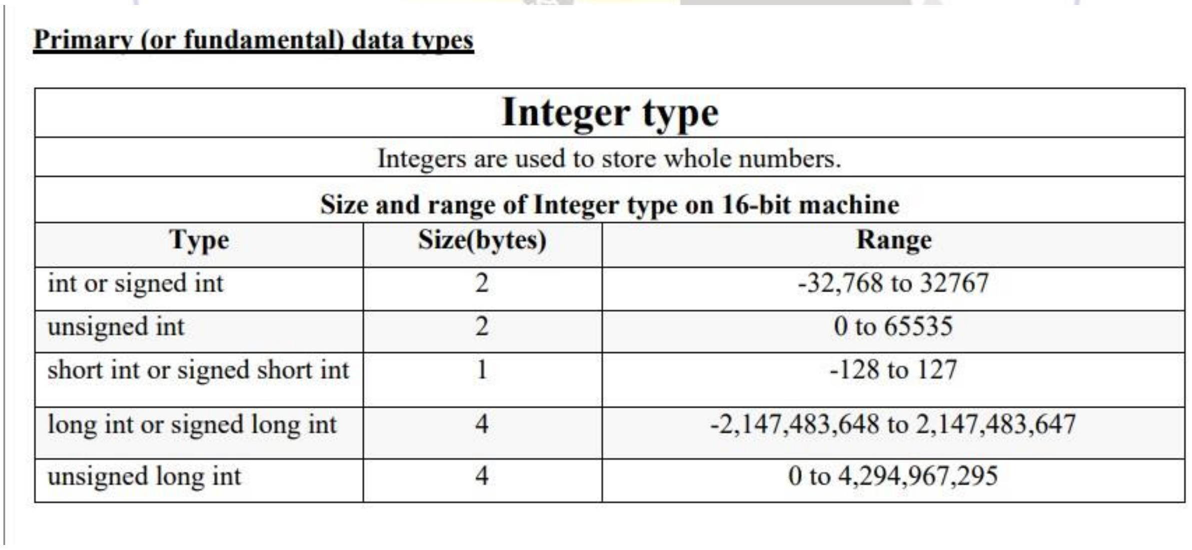
ANSI C Supports four classes of data types.

- Primary (or fundamental)
- User-Defined Data Types
- Derived Data Types
- Empty Data set



Primary or Fundamental Data Types:

Fundamental data types are basic built-in types of c programming language. These are integer data type (int), floating data type (float), and character data type (char).



Primary (or fundamental) data types

Integer type		
Integers are used to store whole numbers.		
Size and range of Integer type on 16-bit machine		
Type	**Size(bytes)**	**Range**
int or signed int	2	-32,768 to 32767
unsigned int	2	0 to 65535
short int or signed short int	1	-128 to 127
long int or signed long int	4	-2,147,483,648 to 2,147,483,647
unsigned long int	4	0 to 4,294,967,295

Floating type

Floating types are used to store real numbers.

Size and range of Float type on 16-bit machine

Type	Size(bytes)	Range
Float	4	3.4E-38 to 3.4E+38
double	8	1.7E-308 to 1.7E+308
long double	10	3.4E-4932 to 1.1E+4932

Character type

Character types are used to store characters value.

Size and range of Char type on 16-bit machine

Type	Size(bytes)	Range
char or signed char	1	-128 to 127
unsigned char	1	0 to 255

User-Defined Data Types

- The data types that are defined by the user depending upon the use case of the programmer are called User-Defined data types.
- These User-defined data types are constructed using a combination of fundamental data types and derived data types.
- User-defined data types are created by the user using a combination of fundamental and derived data types.
- To create a user-defined data type, the c programming language has the following constructs:
  i) Structures (Struct)
  ii) Union
  iii) Type definitions (Typedef)
  iv) Enumerations (enum)

Why do we need User-Defined Data Types in C?

- User-defined data types in C are highly customizable, depending upon the use case of the programmer.
- User-defined data types in C increase the readability of the program as data types can be given meaningful names.
- User-defined data types increase the extensibility of the program to introduce new data types and manage data in the application code.
- If we do not use user defined data types in C, then it is very difficult to maintain multiple variables that represent information belonging to the same entity in the program logic.

➢ Let us take an example to understand why we need user-defined data types. Suppose we have to implement a C program where we want to maintain the following information:

Records of student like name, age, school name and class, and date of birth.
To do this, if we do not use any user defined data types in C, then we will have to use multi-dimensional arrays to represent the information in the following way:

```
// We need an array to store name, age, schoolName, class and date of birth.
char [100][100] student_name;
//student_name[i] = Name of i-th student.
int [100] age;
// age[i] = Age of i-th student.
char [100][100] schoolName;
// schoolName[i] = school name of i-th student.
char [100][100] class;
// class[i] = class of i-th student.
int [100][3] date_of_birth;
// date_of_birth[i][0] = Birth day of i-th student
// date_of_birth[i][1] = Birth Month of i-th student
// date_of_birth[i][2] = Birth Year of i-th student
```

We can represent the above data using user defined data types in C. Below we will discuss the use of structures in C to represent the data and typedef to give a meaningful name to the structure.

```
typedef struct {
    // Create a struct to store the date of birth.
    int day, month, year;
} DOB;

typedef struct {
    // Create a struct to store information about a particular student.
    char name[100];
    int age;
    char schoolName[100];
    char class[100];
    DOB* date_of_birth;
} Student;

// Create an array of the type `Student` to store information about all students.
Student [100] student_details;
```

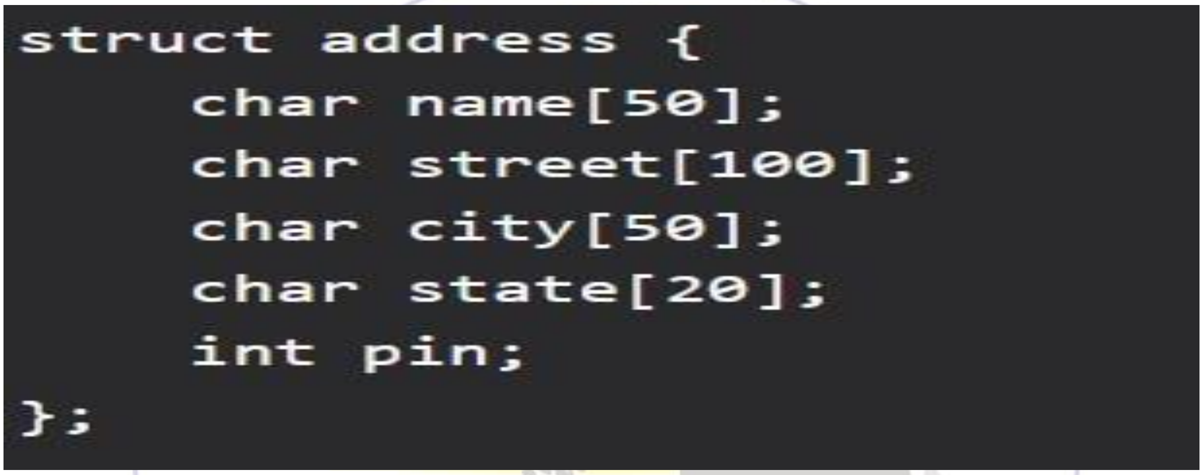
You can see the difference in implementation in the two examples given above. In general, when a software program is written, the developer needs to make sure that the code is readable, clean, and maintainable. Using user-defined data types, these things can be implemented.

Types of User-Defined Data Types in C

Structures

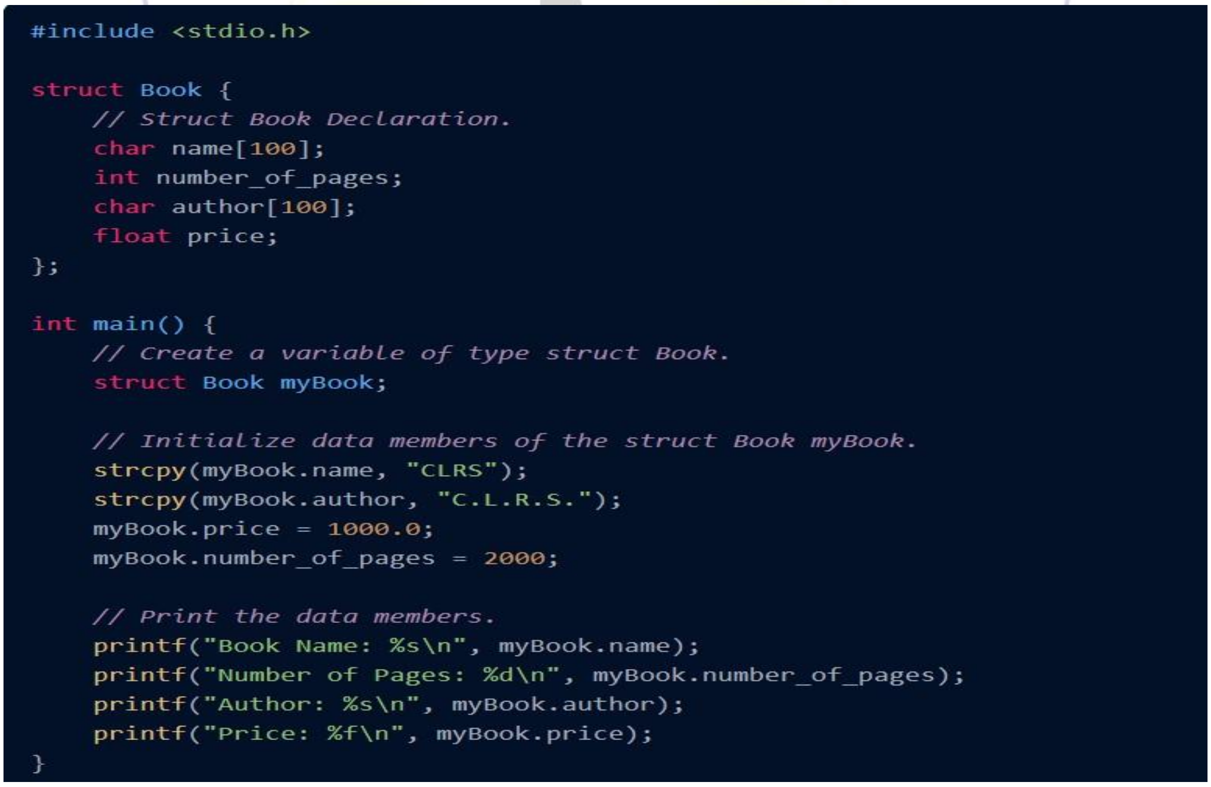
A Structure is a user defined data type in C/C++. A Structure creates a data type that can be used to group items of possibly different types into a single type. A struct (or structure) is a collection of variables (can be of different types) under a single name.

Syntax-



```
struct address {
      char name[50];
      char street[100];
      char city[50];
      char state[20];
      int pin;
};
```

Ex.



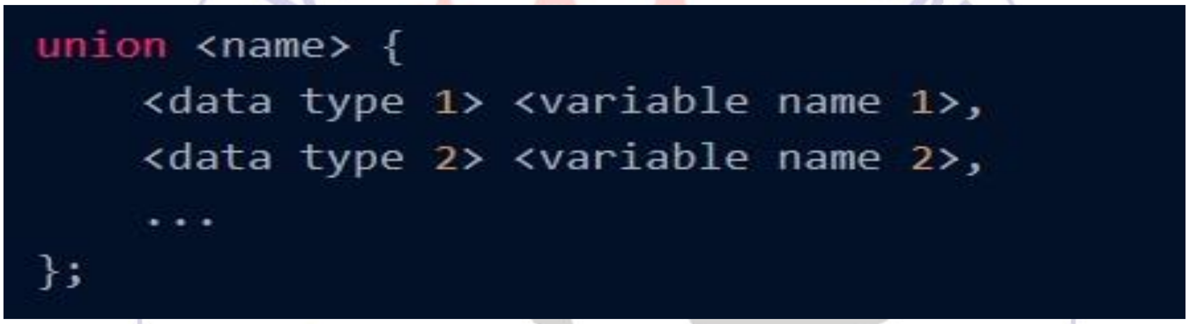
```
#include <stdio.h>

struct Book {
    // Struct Book Declaration.
    char name[100];
    int number_of_pages;
    char author[100];
    float price;
};

int main() {
    // Create a variable of type struct Book.
    struct Book myBook;

    // Initialize data members of the struct Book myBook.
    strcpy(myBook.name, "CLRS");
    strcpy(myBook.author, "C.L.R.S.");
    myBook.price = 1000.0;
    myBook.number_of_pages = 2000;

    // Print the data members.
    printf("Book Name: %s\n", myBook.name);
    printf("Number of Pages: %d\n", myBook.number_of_pages);
    printf("Author: %s\n", myBook.author);
    printf("Price: %f\n", myBook.price);
}
```

Union:

- Union is a user defined data type.
- It is a collection of different data types in the c programming language that allows users to store different data types at the same memory location in the RAM.
- Unions are like structures in the C programming language with an additional property that all members of a union share the same memory location in the RAM.
- We can define a Union with many data members, but only one member can contain a value at any given time.
- The memory of the Union data type is equal to the size of the data member of the Union with the highest number of bytes in size while the memory required in the case of structures in C is the sum of total bytes required by all the data members of the structure.

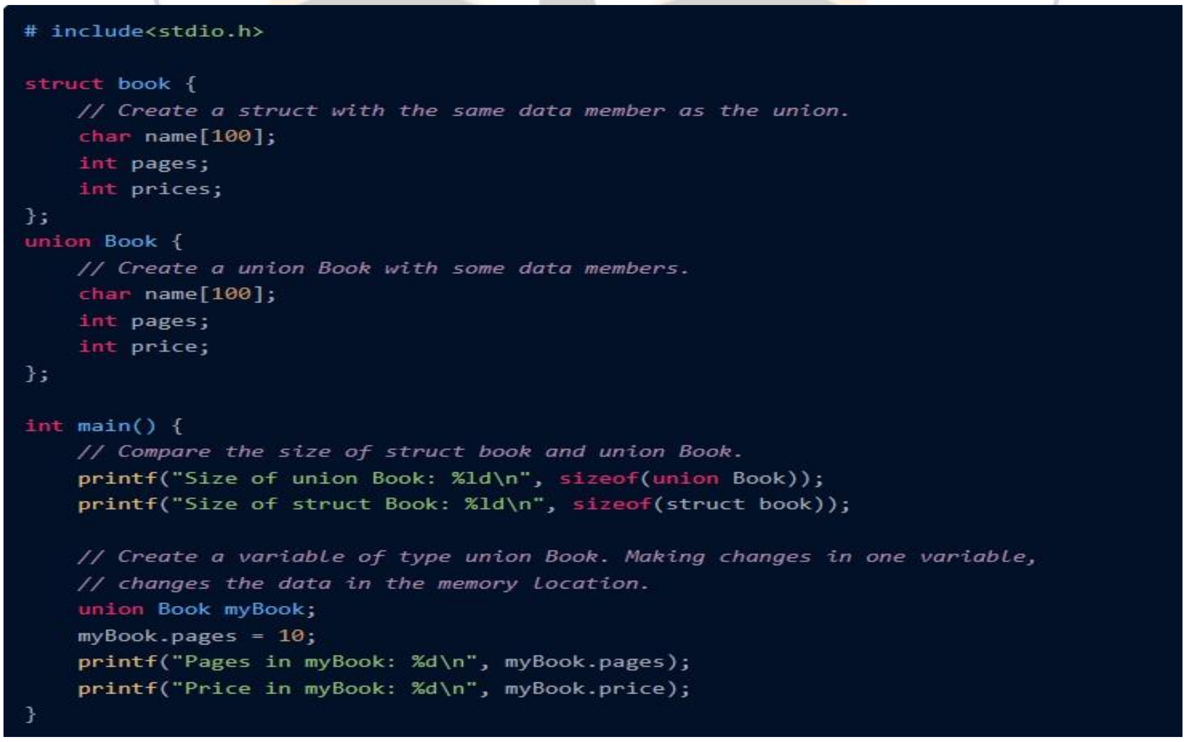
Declaration

A Union is declared using the keyword union and members are accessed using the dot (.) operator.



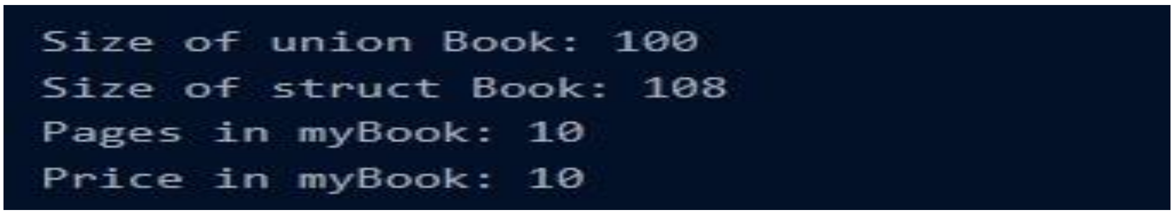
```
union <name> {
    <data type 1> <variable name 1>,
    <data type 2> <variable name 2>,
    ...
};
```

Example



```
# include<stdio.h>

struct book {
    // Create a struct with the same data member as the union.
    char name[100];
    int pages;
    int prices;
};
union Book {
    // Create a union Book with some data members.
    char name[100];
    int pages;
    int price;
};

int main() {
    // Compare the size of struct book and union Book.
    printf("Size of union Book: %ld\n", sizeof(union Book));
    printf("Size of struct Book: %ld\n", sizeof(struct book));

    // Create a variable of type union Book. Making changes in one variable,
    // changes the data in the memory location.
    union Book myBook;
    myBook.pages = 10;
    printf("Pages in myBook: %d\n", myBook.pages);
    printf("Price in myBook: %d\n", myBook.price);
}
```

Output:



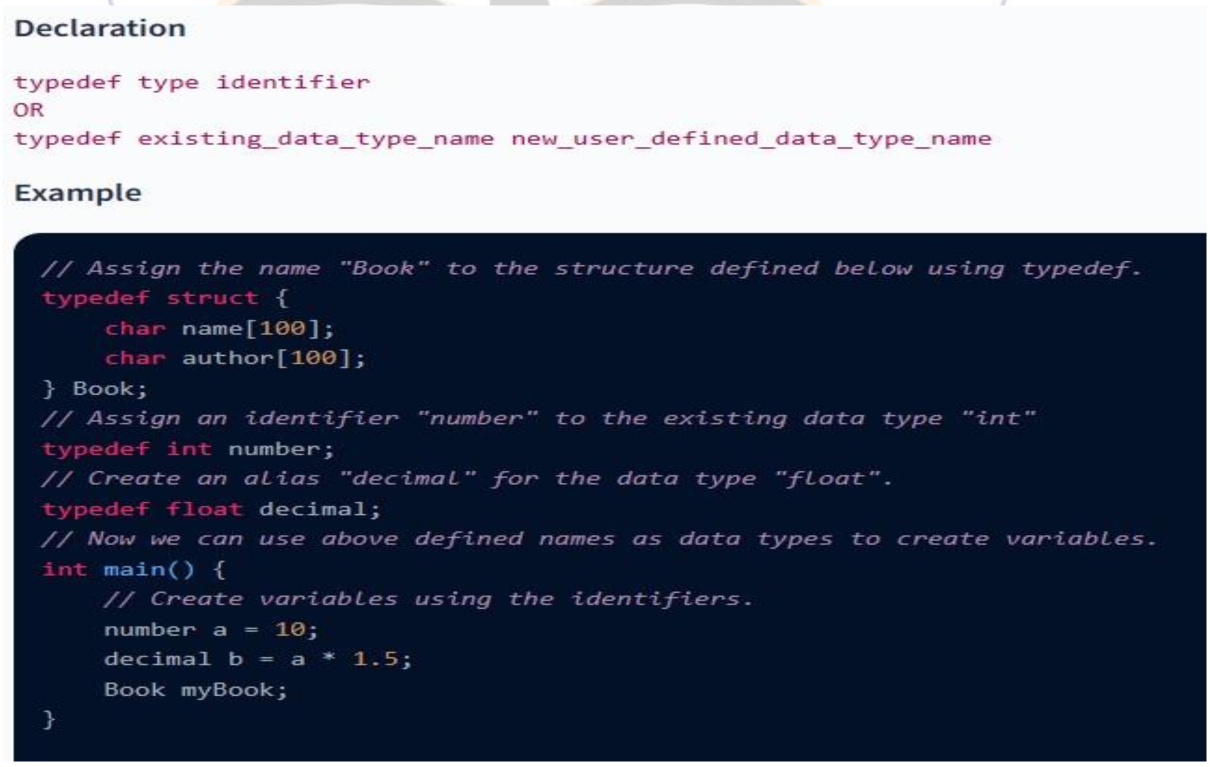
```
Size of union Book: 100
Size of struct Book: 108
Pages in myBook: 10
Price in myBook: 10
```

Explain-

- In the above example, if we compare the size of the struct and union with the same data members then we can see that the size of the struct book is the sum of the size of all its data members.
- In contrast, the size of the union Book is the size of the data member with the maximum size in bytes.
- Moreover, changes in the data member pages is reflected in the data member prices. This means both the variables share the same memory location and changes in one will be reflected upon the other.

<u>Type Definitions (typedef)</u>

- Type definitions allow users to define an identifier that would represent a data type using existing or predefined data types.
- It is used to create an alias or an identifier for an existing data type by assigning a meaningful name to the data type.
- The identifier is defined using the keyword typedef.
- The main advantage of using typedef is that it allows the user to create meaningful data type names.
- This increases the readability of the program.



Declaration

```
typedef type identifier
OR
typedef existing_data_type_name new_user_defined_data_type_name
```

Example

```c
// Assign the name "Book" to the structure defined below using typedef.
typedef struct {
    char name[100];
    char author[100];
} Book;
// Assign an identifier "number" to the existing data type "int"
typedef int number;
// Create an alias "decimal" for the data type "float".
typedef float decimal;
// Now we can use above defined names as data types to create variables.
int main() {
    // Create variables using the identifiers.
    number a = 10;
    decimal b = a * 1.5;
    Book myBook;
}
```

Explain-

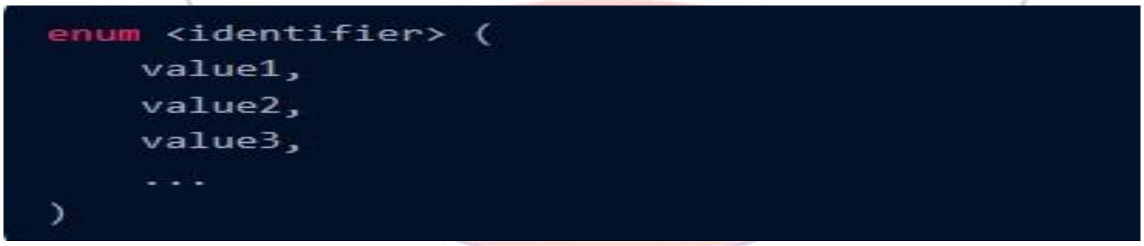
- In this first example, we declare a struct with two fields and define its type as Book.
- In the second example, we define the type of int as number so we can use the data type number to initialize a variable with an integer value.
- In the third example, we define the type of float as decimal so we can use the keyword decimal to declare a float variable.

Enumerations (enum)

- Enumerations is another user-defined data type in C.
- Enumerations in C are a way to define constants so that they can be used in the program.
- These constants are usually defined globally and are used throughout the code.
- The constant values are generally integers and are given meaningful names to increase the readability of the program and highlight the need for the constants being used.
- Related constants can be defined in the set using Enumerations in C.
- This set of named values is called elements or members.
- The enumeration in C is used to assign a constant integer value to variable names.
- The constant values can be assigned by the user.
- Each value is associated with a unique name.
- This unique named constant is global and cannot be reused in some other enumeration.
- Users can assign the value or if these values are not assigned then the compiler will assign the values by default starting from 0.
- Note that two enums cannot have the same variable names as constants as enums are global in the C programming language.

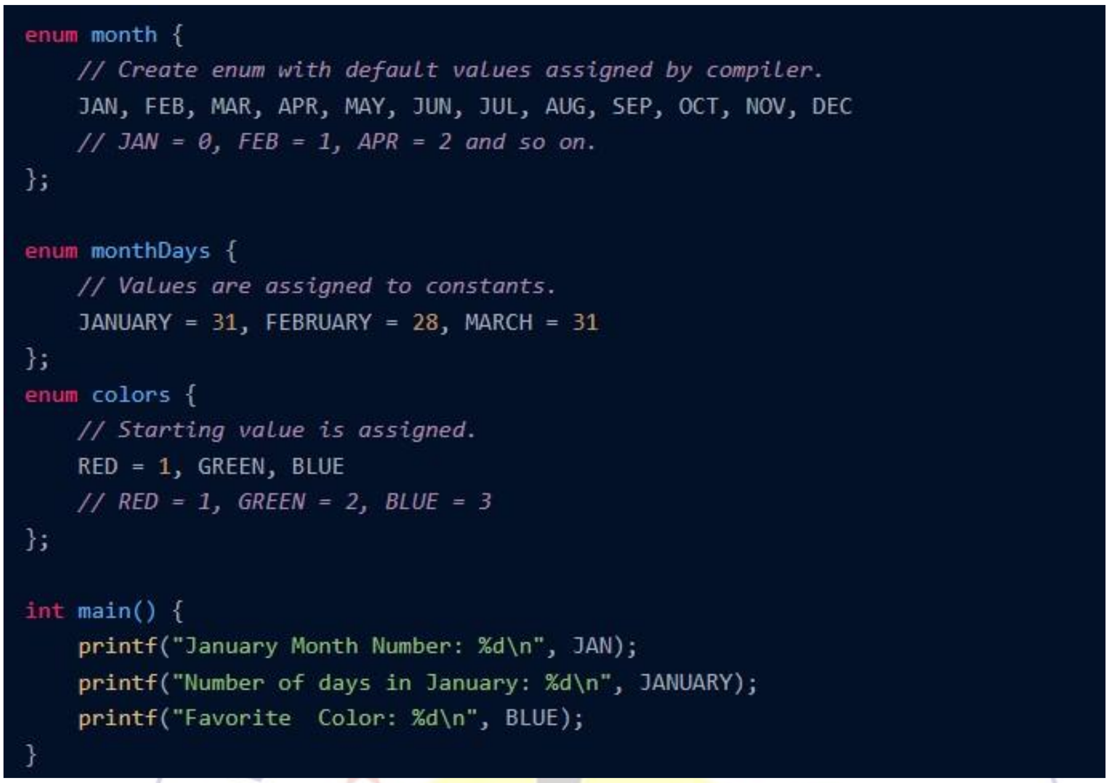
Declaration

Enumerations are declared using the keyword enum.



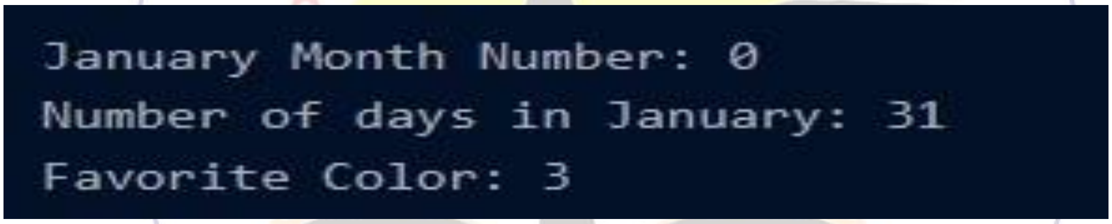
```
enum <identifier> (
    value1,
    value2,
    value3,
    ...
)
```

Example



```c
enum month {
    // Create enum with default values assigned by compiler.
    JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC
    // JAN = 0, FEB = 1, APR = 2 and so on.
};

enum monthDays {
    // Values are assigned to constants.
    JANUARY = 31, FEBRUARY = 28, MARCH = 31
};
enum colors {
    // Starting value is assigned.
    RED = 1, GREEN, BLUE
    // RED = 1, GREEN = 2, BLUE = 3
};

int main() {
    printf("January Month Number: %d\n", JAN);
    printf("Number of days in January: %d\n", JANUARY);
    printf("Favorite  Color: %d\n", BLUE);
}
```

Output



```
January Month Number: 0
Number of days in January: 31
Favorite Color: 3
```

Derived Data Types

- Derived Data Types are derived from fundamental data types, like functions, arrays, and pointers in the c programming language.

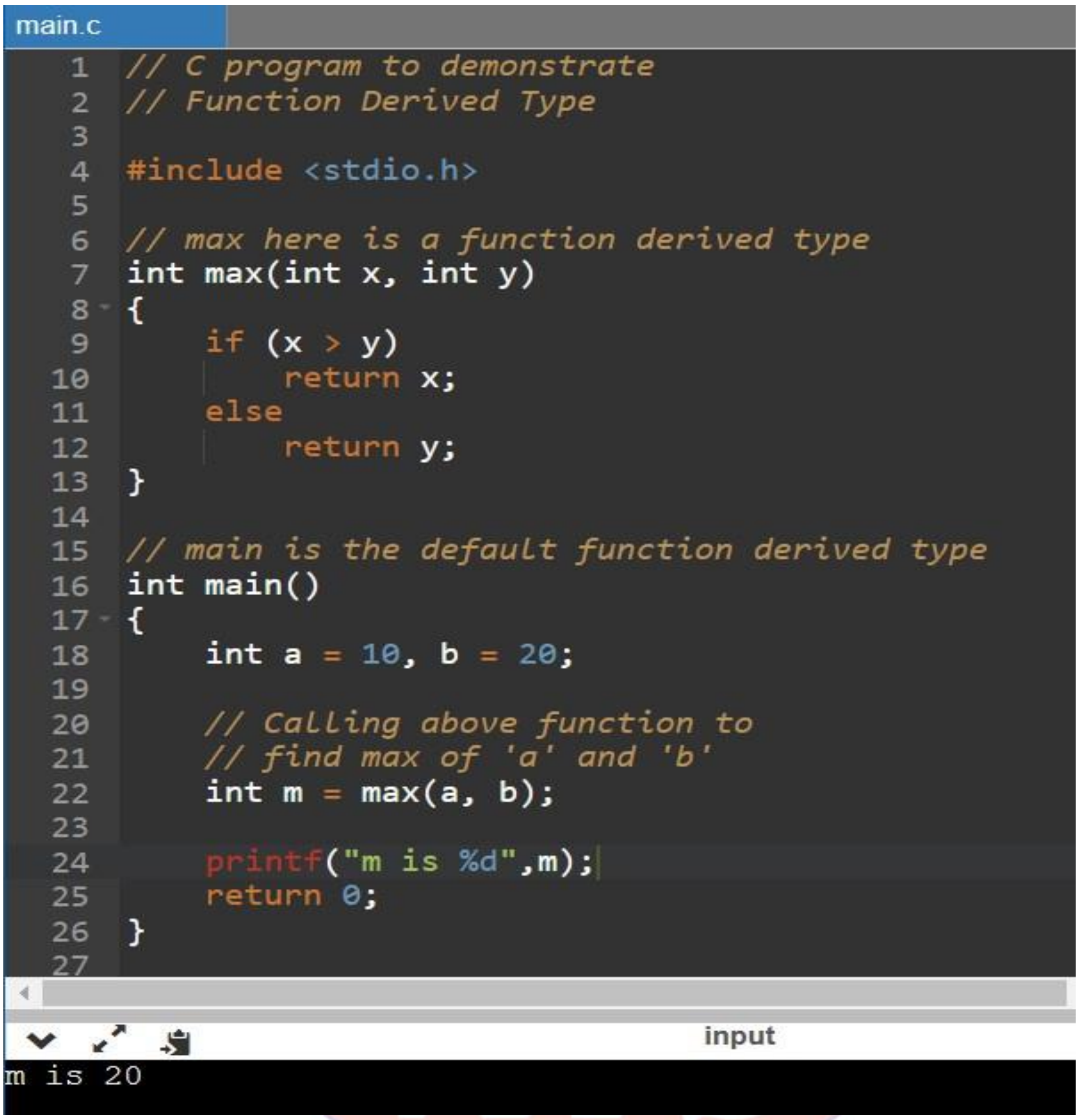
Function

- A function is a block of code or program-segment that is defined to perform a specific well-defined task.
- A function is generally defined to save the user from writing the same lines of code again and again for the same input.
- All the lines of code are put together inside a single function and this can be called anywhere required.
- main() is a default function that is defined in every program of C.

Syntax

```
FunctionType FunctionName(parameters)
```

Ex.



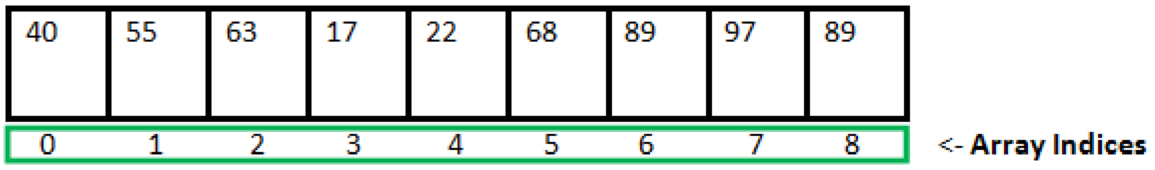
main.c

```c
// C program to demonstrate
// Function Derived Type

#include <stdio.h>

// max here is a function derived type
int max(int x, int y)
{
    if (x > y)
        return x;
    else
        return y;
}

// main is the default function derived type
int main()
{
    int a = 10, b = 20;

    // Calling above function to
    // find max of 'a' and 'b'
    int m = max(a, b);

    printf("m is %d",m);
    return 0;
}
```

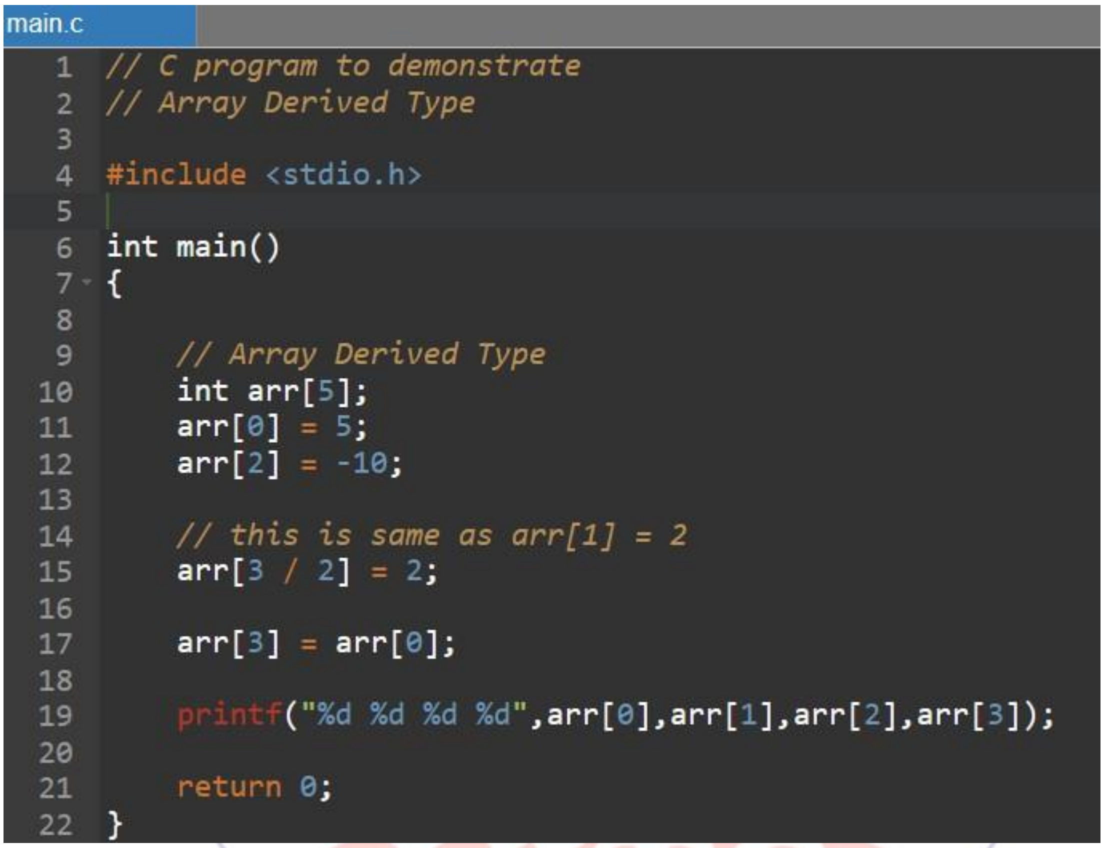
```
m is 20
```

<u>Arrays</u>

An array is a collection of items stored at continuous memory locations. The idea of array is to represent many instances in one variable.

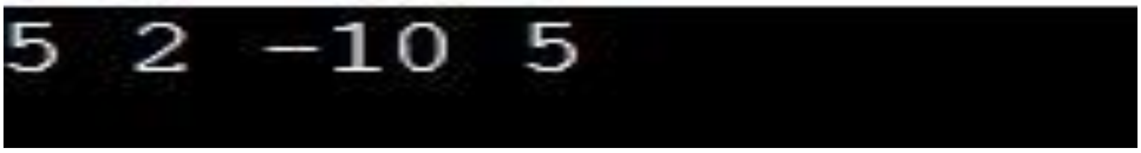


Array Length = 9
First Index = 0
Last Index = 8



```c
// C program to demonstrate
// Array Derived Type

#include <stdio.h>

int main()
{

    // Array Derived Type
    int arr[5];
    arr[0] = 5;
    arr[2] = -10;

    // this is same as arr[1] = 2
    arr[3 / 2] = 2;

    arr[3] = arr[0];

    printf("%d %d %d %d",arr[0],arr[1],arr[2],arr[3]);

    return 0;
}
```

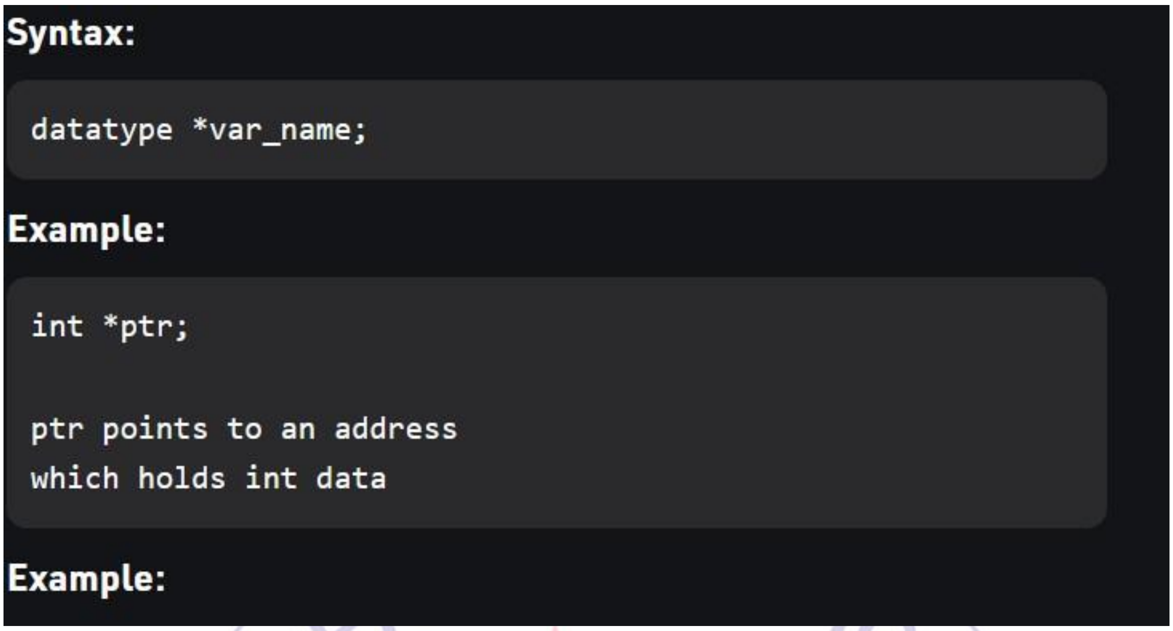
Output



```
5 2 -10 5
```

<u>Pointers</u>

Pointers are symbolic representation of addresses. They enable programs to simulate call-by-refrence as well as to create and manipulate dynamic data structures. It’s general declaration in C/C++ has the format:



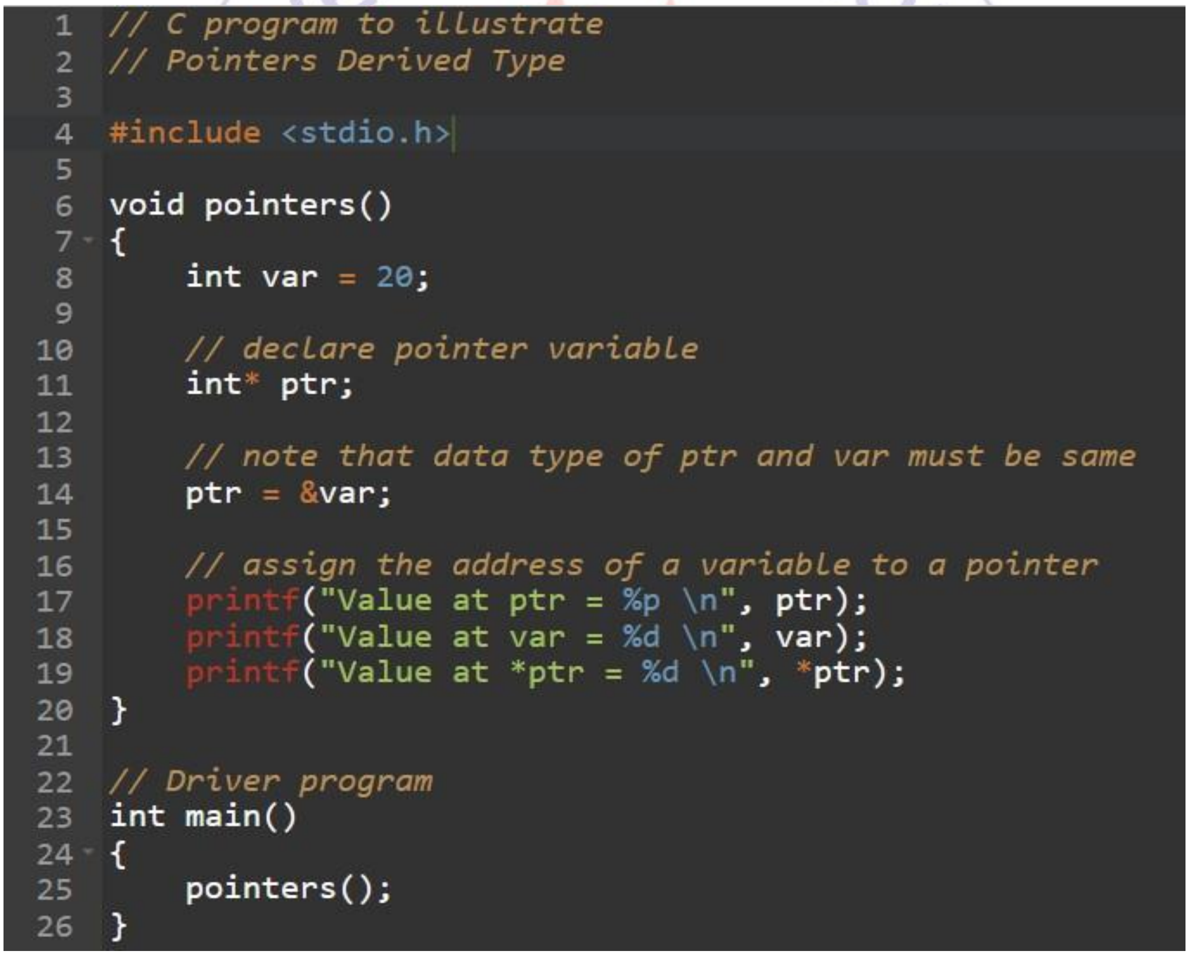
Syntax:

```
datatype *var_name;
```

Example:

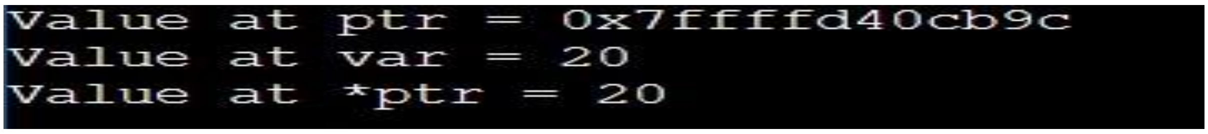
```
int *ptr;

ptr points to an address
which holds int data
```

Example:



```c
// C program to illustrate
// Pointers Derived Type

#include <stdio.h>

void pointers()
{
    int var = 20;

    // declare pointer variable
    int* ptr;

    // note that data type of ptr and var must be same
    ptr = &var;

    // assign the address of a variable to a pointer
    printf("Value at ptr = %p \n", ptr);
    printf("Value at var = %d \n", var);
    printf("Value at *ptr = %d \n", *ptr);
}

// Driver program
int main()
{
    pointers();
}
```

Output:



```
Value at ptr = 0x7ffffd40cb9c
Value at var = 20
Value at *ptr = 20
```

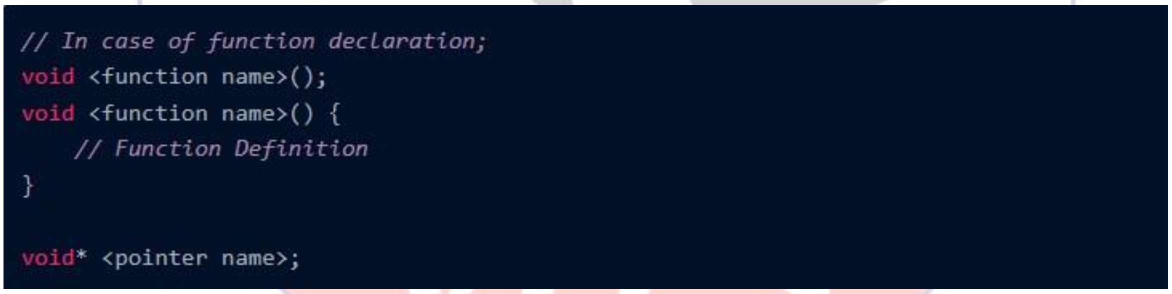
References

- When a variable is declared as reference, it becomes an alternative name for an existing variable.
- A variable can be declared as reference by putting '&' in the declaration.

Empty Data Type

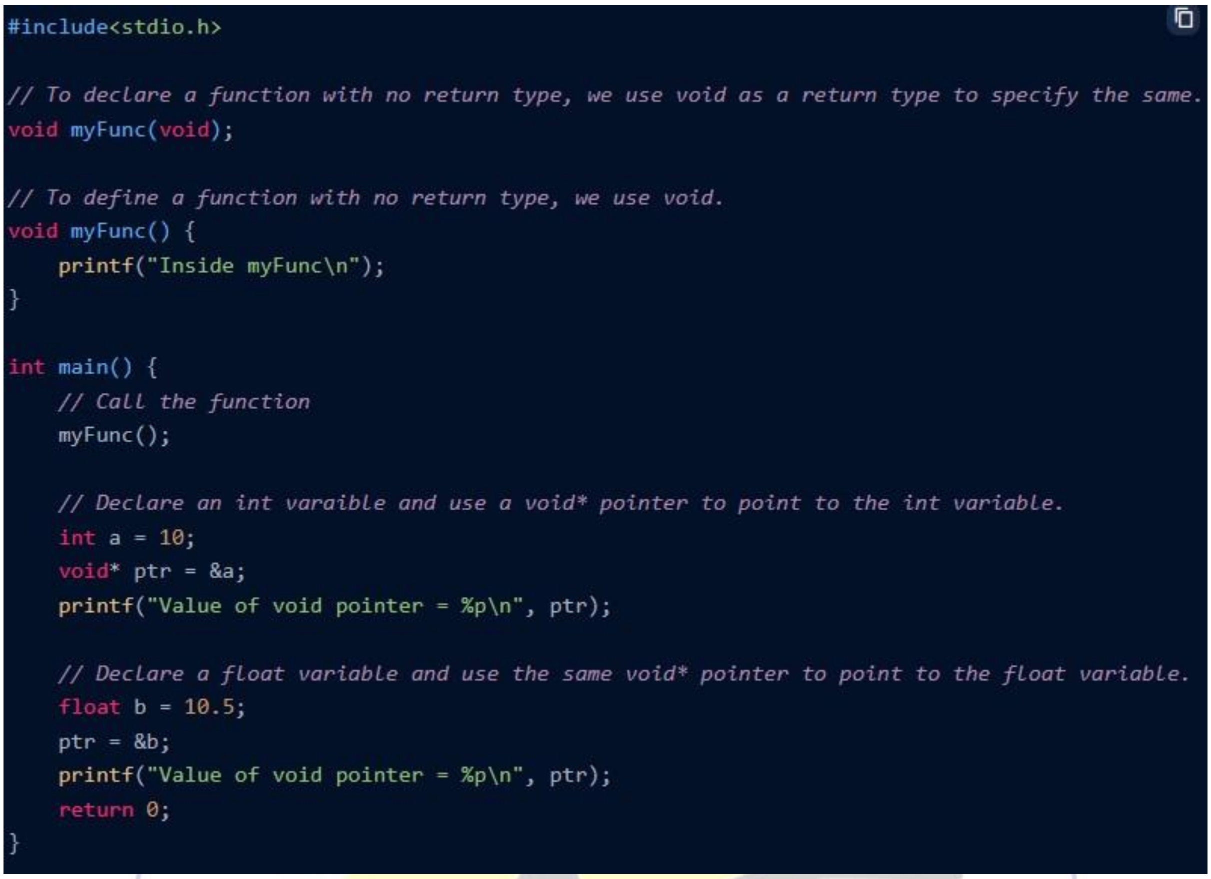
- Void type means no value. This is usually used to specify the type of functions which returns nothing.
- The keyword void is an empty data type that is used in functions and pointers.
- It does not represent any value.
- In the case of functions, it specifies that the function does not return any value and in the case of pointers, it specifies that the pointer is universal.
- C11 standard defines the keyword void as:
  i) The void type comprises an empty set of values; it is an incomplete object type that cannot be completed.
  ii) In the case of functions, it represents the return type of the function. It means the function does not return any value to the calling function.
  iii) In the case of pointers, it represents that the pointer does not specify what it points to. It can point to any data type.

Declaration



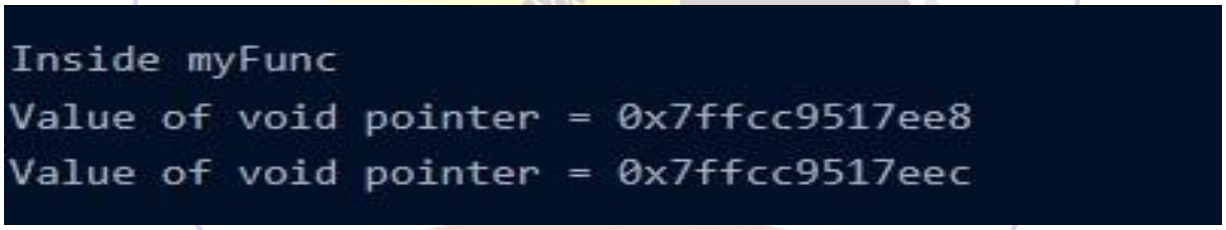
```
// In case of function declaration;
void <function name>();
void <function name>() {
    // Function Definition
}

void* <pointer name>;
```

Example



```c
#include<stdio.h>

// To declare a function with no return type, we use void as a return type to specify the same.
void myFunc(void);

// To define a function with no return type, we use void.
void myFunc() {
    printf("Inside myFunc\n");
}

int main() {
    // Call the function
    myFunc();

    // Declare an int varaible and use a void* pointer to point to the int variable.
    int a = 10;
    void* ptr = &a;
    printf("Value of void pointer = %p\n", ptr);

    // Declare a float variable and use the same void* pointer to point to the float variable.
    float b = 10.5;
    ptr = &b;
    printf("Value of void pointer = %p\n", ptr);
    return 0;
}
```

Output



```
Inside myFunc
Value of void pointer = 0x7ffcc9517ee8
Value of void pointer = 0x7ffcc9517eec
```

Explain

- In this example, we can see the use of void in the case of function and pointers.
- In the case of functions, when we do not return any value, we need to specify that the function has a void return type.
- In the case of pointers, the same void* pointer can be used to point to an int variable or a float variable.

****