UNIT-5

Structures :

Defining a Structure:

Structure in c is a user-defined data type that enables us to store the **collection of different data types.** Each element of a **structure** is called a member.

- To define a structure, you must use the **struct** keyword.
- The struct statement defines a new data type, with more than one member. The format of the struct statement is as follows

Declaration of structure:

We use `struct` keyword to declare a structure.

Syntax:

```
struct   structure_Name

{
 member_type     variablename;
 ...
 member_type    variablename;

}structure variables;
```

Example:

```
struct Books
 {
char title[50];
char author[50];
char subject[100];
int book_id;
} book;
```

How to initialize a structure variable?

C language supports multiple ways to initialize a structure variable. You can use any of the initialization method to initialize your structure.

1. Initialize structure using dot operator :

In C, we initialize or access a structure variable either through **dot .** operator. This is the most easiest way to initialize or access a structure.

Example:

```
// Declare structure variable

struct student stu1;


// Initialize structure members

stu1.name  = "Pankaj";
stu1.roll  = 12;

stu1.marks = 79.5f;
```

2. Value initialized structure variable :

The above method is easy and straightforward to initialize a structure variable. However, C language also supports value initialization for structure variable. Means, you can initialize a structure to some default value during its variable declaration.

Example:

```
// Declare and initialize structure variable

struct student stu1 = { "Pankaj", 12, 79.5f };
```

Note: *The values for the value initialized structure should match the order in which structure members are declared.*

Invalid initialization:

```
// Declare and initialize structure variable

struct student stu1 = { 12, "Pankaj", 79.5f };
```

Accessing Structure Members:

- To access any member of a structure, we use the **member access operator (.)**.

- The member access operator is coded as a **period operator** between the structure variable name and the structure member that we wish to access.
- You would use the keyword **struct** to define variables of structure type.
- The following example shows how to use a structure in a program.

Example program:

```c
/* Example program by to initialize values using dot operator */

#include <stdio.h>

struct employee
{
    char  name[30];
    int   id;
    flaot salary;
} emp;

int main()
{
    Struct employee emp;
    Emp.name=”CSE”;
    Emp.id=201;
    Emp.salary=1500.50;

    printf("Enter name = %s”,emp.name);
    printf("Enter name = %d”,emp.id);
    printf("Enter name = %f”,emp.salary);

return(0);
}
```

Example program:

```
/* Example program by to initialize values */

struct book
{
   char  name[30];
   int   id;
   flaot price;
};

int main()
{
     Struct book bk={“cprogramming”, 150,255.50};

    printf("Enter name = %s”,bk.name);
    printf("Enter name = %d”,bk.id);
    printf("Enter name = %f”,bk.salary);

return(0);
}
```

Example program:

/* program for copy one structure variable to another structure variable.*/

```
struct book
{
    char  bname[30];
    int   id;
    flaot price;
};
int main()
{
    Struct book b1={“cprogramming”, 150,255.50};
    Struct book b2,b3;

    Strcpy(b2.name,b1.name);

    b2.id = b1.id;
    b2.price = b1.price;
    printf(“ to print book details”);
    printf("%s”,b1.name);
    printf("%s”,b2.name);
```

```
    printf("%d”,b1.id);
    printf("%d”,b2.id);
```

```
    printf("%f”,b1.price);
    printf("%f”,b2.price);

return(0);
}
```

Example program using “scanf”:

```
#include <stdio.h>
struct book
{
    char  name[30];
    int   id;
    flaot price;
};

int main()

{
     Struct book b1;
     Printf(“enter book details”);
     printf("enter book name”);
     scanf("%s”,bk.name);

     printf("enter book id”);
     scanf("%d”,&bk.id);

     printf("enter book price”);
     scanf("%d”,&bk.price);
     printf("%s”,bk.name);
    printf("%d”,bk.id);
     printf("%f”,bk.price);

return(0);
 }
```

Nested structures:

Nested structure is also called as **Structure within structure or** A structure inside another structure is called nested structure.

For example:

we have two structures named Address and Employee. To make Address nested to Employee, we have to define Address structure before and outside Employee structure and create an object of Address structure inside Employee structure.

Syntax for structure within structure or nested structure:

```
struct structure1
{
    ----------
    ----------
};

struct structure2
{
    ----------
    ----------
    struct structure1 obj;
};
```

Example :

```c
#include <stdlib.h>
struct time
{
    int   hours;
    int   minutes;
    int   seconds;
};
struct car
{
    int   carnumber;
    struct time ST;        // nested structure
    struct time ET;
};


int main()
{
```

```
Struct car  C;

 Printf(“enter car number”);
 scanf("%d”,&c.carnumber);
 Printf(“enter car starting time i.e ST”);
 scanf("%d %d %d ”,&c.st.hours, &c.st.minutes, &c.st.seconds);
 Printf(“enter car Ending time i.e ET”);
 scanf("%d %d %d ”,&c.et.hours, &c.et.minutes, &c.et.seconds);

// Display car timing details

    printf("%d %d %d ”,c.st.hours, c.st.minutes, c.st.seconds);
    printf("%d %d %d ”,c.et.hours, c.et.minutes, c.et.seconds);
 return(0);
}
```

Array of structures:

- Generally arrays are used to store multiple similar data elements just like that we **can also store structure in array which is called array of structure**.
- In arrays of structure each element is structure type.
- To declare an array of structures, first the structure must be defined and then, an array variable of that type can be defined.

Example: **Struct structure_name structure_variable [array size];**

struct book b[10]; //10 elements in an array of structures of type ‘book’

Syntax:

```
struct   structure_Name
{
 member_type    variablename;
 member_type    variablename;
 ...
 member_type    variablename;
} ;
```

Struct structure_name structure_variable [array size];

Where **Struct** is a keyword, followed by **structure name** and **structure variable with including size of array.**

Example

Given below is the C program for accepting and printing details of 3 students with regards to an array of structures −

```c
#include <stdlib.h>
#include <string.h>
struct student
{
    int id;
    char name[30];
    float percentage;
};
int main()
{
   int i;
    struct student record[2];
   // 1st student's record
   record[0].id=1;
   strcpy(record[0].name, "Bhanu");
   record[0].percentage = 86.5;

// 2nd student's record
   record[1].id=2;
   strcpy(record[1].name, "Priya");
   record[1].percentage = 90.5;

// 3rd student's record
    record[2].id=3;
    strcpy(record[2].name, "Hari");
    record[2].percentage = 81.5;
    for(i=0; i<3; i++)
  {
      printf(" Records of STUDENT : %d \n", i+1);
```

```
        printf(" Id is: %d \n", record[i].id);
        printf(" Name is: %s \n", record[i].name);
        printf(" Percentage is: %f\n\n",record[i].percentage);
    }
    return 0;
}
```

Output

When the above program is executed, it produces the following result −

```
Records of STUDENT : 1
Id is: 1
Name is: Bhanu
Percentage is: 86.500000

Records of STUDENT : 2
Id is: 2
Name is: Priya
Percentage is: 90.500000

Records of STUDENT : 3
Id is: 3
Name is: Hari
Percentage is: 81.500000
```

Pointers and structures:

- Pointers are used to store address of normal variable.
- We can also store address of structure variable .

  struct structurename *struct_pointer;

- Now, you can store the address of a structure variable in the above-defined pointer variable. To find the address of a structure variable, place the '&' operator before the structure's name as follows:
  struct_pointer = &Book1;

- To access the members of a structure using a pointer to that structure, **you must use the ->** operator as follows:
  struct_pointer->title;

Declaration of array of structure:

```
Struct  structure_Name
{
 member_type     variable1;
 member_type     variable2;
 ...
 member_type      variable n;
} ;
```

```
Struct    structure_name    *structure_variable;
```

Here Struct is a keyword, and variable 1,2…n are the structure members.

Generally we use . (dot) operator to access structure members,

But when ever we have a pointer structure we use arrow (-→ operator to access the structure members.

Syntax: **pointer_structure_variable name → structure_member.**

Example : program to declare pointer to structures and display the structure elements.

```
#include <stdio.h>
struct book
{
    char  name[30];
    int   id;
    flaot price;
};

int main()
{
    Struct book b1={“cprog”,123,545.50};

    Struct book *ptr;

    Ptr = &b1;

    Printf(“ to display book details”);


    printf("%s”,ptr->name);
```

```
      printf("%d”,ptr->id);
      printf("%f”,ptr->price);

return(0);

}
```

Example : program to declare structures members as a pointers, and display the structure members, using pointer to structure.

```
#include <stdio.h>
struct book
{
    char  *bname[30];
    int   *id;
    flaot *price;
};

int main()
{
   Struct book *ptr;

    Char   num[30]=”java”;
    int    i=122;
    float  p=50.5;

    Ptr  -> bname=num;
    Ptr  ->id=&i;
    Ptr  ->price=&p;

     Printf(“ to display book details”);
     printf("%s”,ptr->bname);
      printf("%d”,*ptr->id);
      printf("%f”,*ptr->price);

return(0);

}
```

Structures and Functions:

The C Programming allows us to pass the structures as the function parameters.

We can pass the C structures to functions in **3 ways.**

1. **Passing Structure Members as arguments to Function.**
   (It is similar to passing normal values as arguments).
2. **Pass structure variable as argument to the function.**
   (passing whole structure).
3. **Pass the address of the structure (pass by reference i.e ponters) to the function.**

1.Passing Structure Members as arguments to Function:

- In this method we will pass structure variables as arguments to the function.
- We can pass individual members to a function just like ordinary variables.
- The following program shows how to pass structure members as arguments to the function.

```c
/* Passing Structure Members as arguments to Function */

struct student
{
char name[20];
int roll_no;
int marks;
};

void print_struct(char name[], int roll_no, int marks);

int main()
{
struct student stu = {"Tim", 1, 78};
print_struct(stu.name, stu.roll_no, stu.marks);
return 0;
}

void print_struct(char name[], int roll_no, int marks)
{
printf("Name: %s\n", name);
printf("Roll no: %d\n", roll_no);
printf("Marks: %d\n", marks);
printf("\n");
}
```

Expected Output:

```
Name: Tim
Roll no: 1
Marks: 78
```

2. Passing Structure Variable as Argument to a Function:

- how to pass structure members as arguments to a function.

- If a structure contains **two-three members** then we can **easily pass** them to function but what if there are 9-10 or more members ?

- **So in such cases instead of passing members individually, we can pass structure variable itself.**

The following program shows how we can pass structure variable as an argument to the function.

```c
/* Passing Structure Variable as Argument to a Function */

struct student
{
    char name[20];
    int roll_no;
    int marks;
};

void print_struct(struct student stu);

int main()
{
    struct student stu = {"George", 10, 69};
    print_struct(stu);
    return 0;
}

void print_struct(struct student stu)
{
    printf("Name: %s\n", stu.name);
    printf("Roll no: %d\n", stu.roll_no);
    printf("Marks: %d\n", stu.marks);
    printf("\n");}
```

Expected Output:

```
Name: George
Roll no: 10
Marks: 69
```

3.Passing Structure Pointers as Argument to a Function:

Although passing structure variable as an argument allows us to pass all the members of the structure to a function there are some downsides to this operation.

- In this method to pass address of the structure is passed to the formal argument.
- The following program demonstrates how to pass structure pointers as arguments to a function.

```c
struct employee
{
    char name[20];
    int age;
    char doj[10];
    char designation[20];
};

void print_struct(struct employee *ptr);

int main()
{
    struct employee dev = {"Jane", 25, "25/2/2015", "Developer"};
    print_struct(&dev);

    return 0;
}

void print_struct(struct employee *ptr)
{
    printf("Name: %s\n", ptr->name);
    printf("Age: %d\n", ptr->age);
    printf("Date of joining: %s\n", ptr->doj);
    printf("Age: %s\n", ptr->designation);
    printf("\n");
}
```

Expected Output:

```
Name: Jin
Age: 25
Date of joining: 25/2/2015
Age: Developer
```

Self Referential Structures:

Self Referential structures are those structures that have one or more pointers which point to the same type of structure, as their member.

Syntax:

```
struct node
{
   int      data1;
   char    data2;
   struct   node  * link;
};
```

structures pointing to the same type of structures are self-referential in nature.

Example:

```
 struct node
{
   int      data1;
   char    data2;
   struct   node  * link;
};

int main()
{
   struct node ob;

   return 0;
}
```

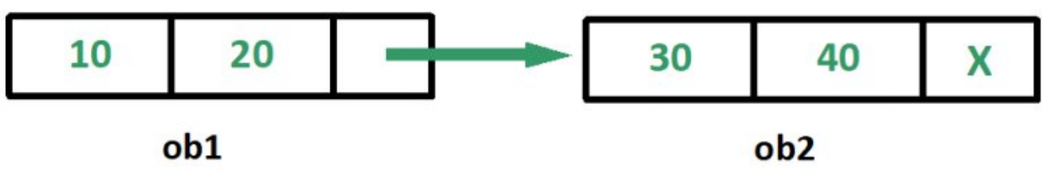
In the above example ‘link’ is a pointer to a structure of type ‘node’. Hence, the structure ‘node’ is a self-referential structure with ‘link’ as the referencing pointer.

Types of Self Referential Structures

1. Self Referential Structure with Single Link
2. Self Referential Structure with Multiple Links

1.Self Referential Structure with Single Link:

- These structures can have only one self-pointer as their member.
- The following example will show us how to connect the objects of a self-referential structure with the single link and access the corresponding data members.
- The connection formed is shown in the following figure.




```
struct node
{
   int data1;
   char data2;
   struct node* link;
};

int main()

{    struct node ob1;                    // Node1

   ob1.link = NULL;
   ob1.data1 = 10;                      // Initialization
   ob1.data2 = 20;

   struct node ob2;                     // Node2

   ob2.link = NULL;
   ob2.data1 = 30;                      // Initialization
   ob2.data2 = 40;

  ob1.link = &ob2;                      // Linking ob1 and ob2

   printf("%d", ob1.link->data1);
   printf("\n%d", ob1.link->data2);
   return 0;
}
```

Output:

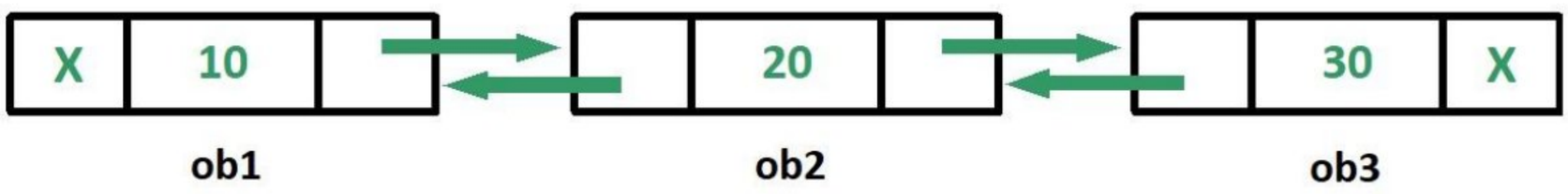
```
30

40
```

2. Self Referential Structure with Multiple Links:

- Self referential structures with multiple links can have more than one self-pointers.
- Many complicated data structures can be easily constructed using these structures.
- Such structures can easily connect to more than one nodes at a time. The following example shows one such structure with more than one links.

The connections made in the above example can be understood using the following figure.




```
struct node
{
   int data;
   struct node  * prev_link;
   struct node  * next_link;
};

int main()
{
   struct node ob1;                          // Node1

   // Initialization
   ob1.prev_link = NULL;
   ob1.next_link = NULL;
   ob1.data = 10;

struct node ob2;                             // Node2

// Initialization
ob2.prev_link = NULL;
ob2.next_link = NULL;
ob2.data = 20;
```

```
struct node ob3;                        // Node3

// Initialization
ob3.prev_link = NULL;
ob3.next_link = NULL;
ob3.data = 30;

// Forward links
   ob1.next_link = &ob2;
   ob2.next_link = &ob3;

   // Backward links
   ob2.prev_link = &ob1;
   ob3.prev_link = &ob2;

   // Accessing  data of ob1, ob2 and ob3 by ob1
   printf("%d\t", ob1.data);
   printf("%d\t", ob1.next_link->data);
   printf("%d\n", ob1.next_link->next_link->data);

   // Accessing data of ob1, ob2 and ob3 by ob2
      printf("%d\t", ob2.prev_link->data);
      printf("%d\t", ob2.data);
      printf("%d\n", ob2.next_link->data);

      // Accessing data of ob1, ob2 and ob3 by ob3
      printf("%d\t", ob3.prev_link->prev_link->data);
      printf("%d\t", ob3.prev_link->data);
      printf("%d", ob3.data);
      return 0;
  }
```

Output:

```
10   20   30
10   20   30
10   20   30
```

Unions

A **union** is a special data type available in C that allows to store different data types in the same memory location. You can define a union with many members, but only one member can contain a value at any given time. Unions provide an efficient way of using the same memory location for multiple-purpose.

Defining a Union:

A **union** is a special data type available in **C** that allows to store different data types in the same memory location. You can **define** a **union** with many members, but only one member can contain a value at any given time. **Unions** provide an efficient way of using the same memory location for multiple-purpose.

Syntax:

```
union union_name
{
   member definition;
   member definition;
   ...
   member definition;
}variable_name;
```

Example:

```
union Data
{
    int i;
    float f;
    char str[20];
} data;
```

- **Arrays and unions.**
- **Pointers with unions.**
- **Unions with functions.**
- **Self Referential Structures:**

All are similar to structures

Difference between Structure and union :

Difference between Structure and Union

STRUCTURE	UNION
Every memory has its own memory sapce.	All the members use the same memory space to store the values.
It can handle all the members(or) a few as required at a time.	It can handle only one member at a time,as all the members use the same space.
Keyword struct is used.	Keyword union is used.
It may be initialized with all its members.	Only its first member may be initialized.
Any member can be accessed at any time without the loss of data.	Only one member can be accessed at any time with the loss of previous data.
Different interpretation for the same memory location are not possible.	Different interpretations for the same
More storage space is required.	Minimum storage space is required.
Syntax: structure strut-name { }var1..varn;	Syntax: union union-name { }var1..varn;

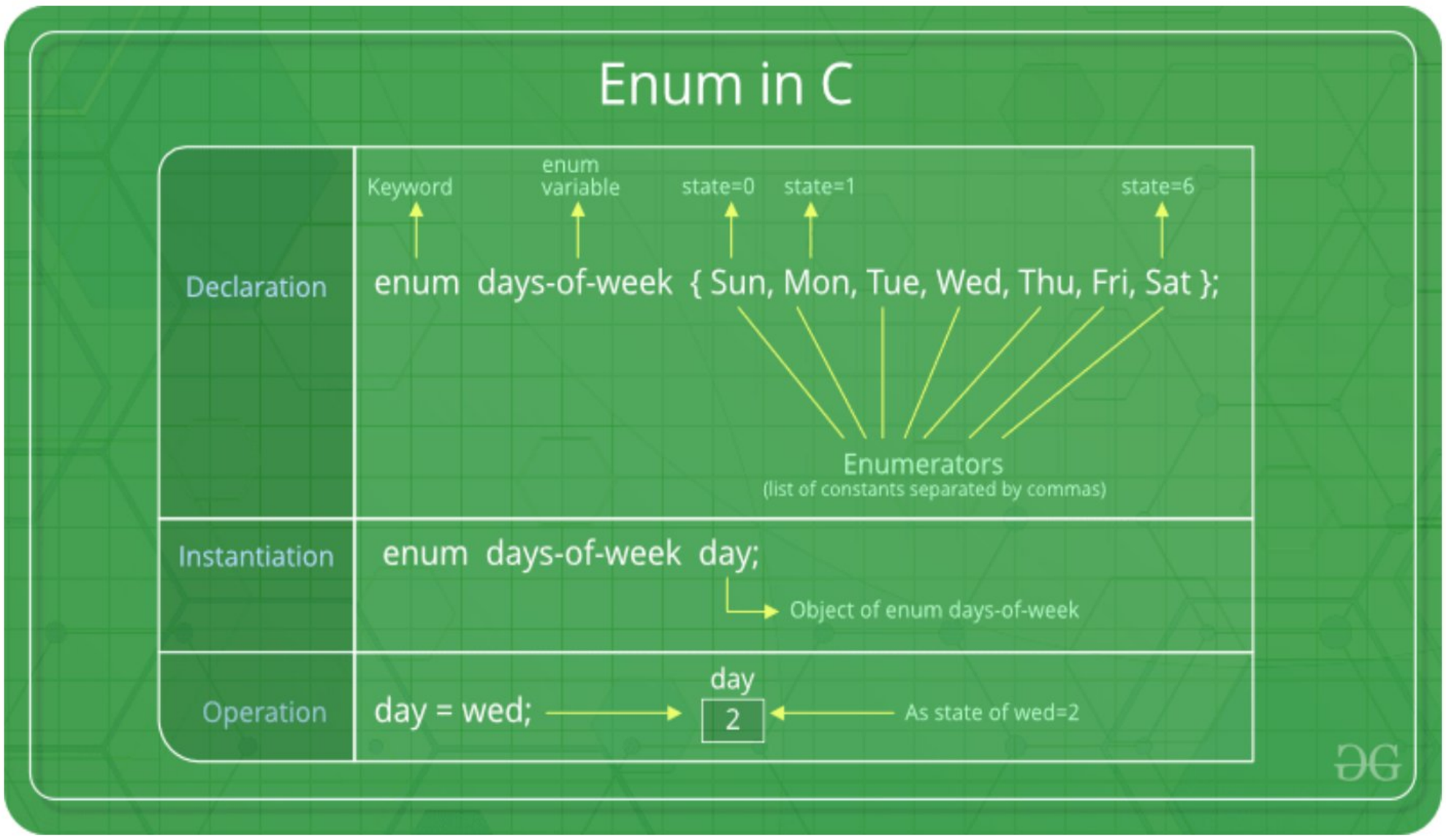
Enumerated in c: (user defined datatype in c)

- Enumeration is a user defined datatype in C language.
- It is used to assign names to the integral constants which makes a program easy to read and maintain. The keyword “enum” is used to declare an enumeration.

Syntax:

```
enum enum_name{const1, const2, ..... };
```

Enumeration (or enum) is a user defined data type in C. It is mainly used to assign names to integral constants, the names make a program easy to read and maintain.



The enum keyword is also used to define the variables of enum type. There are two ways to define the variables of enum type as follows.

```
// An example program to demonstrate working
// of enum in C
#include<stdio.h>

int main()
{
    enum week{sunday, monday, tuesday, wednesday, thursday, friday,
saturday};

printf("%d",sunday);
printf("%d",monday);
printf("%d",tuesday);
printf("%d",wednesday);
printf("%d",thursday);
printf("%d",friday);
printf("%d",saturday);
    return 0;
}
```

Output: 0 1 2 3 4 5 6

Files:

File Mode	Description
R	Open a file for reading. If a file is in reading mode, then no data is deleted if a file is already present on a system.
W	Open a file for writing. If a file is in writing mode, then a new file is created if a file doesn't exist at all. If a file is already present on a system, then all the data inside the file is truncated, and it is opened for writing purposes.
A	Open a file in append mode. If a file is in append mode, then the file is opened. The content within the file doesn't change.
r+	open for reading and writing from beginning
w+	open for reading and writing, overwriting a file
a+	open for reading and writing, appending to file

Getting data using fseek()

If you have many records inside a file and need to access a record at a specific position, you need to loop through all the records before it to get the record.

This will waste a lot of memory and operation time. An easier way to get to the required data can be achieved using `fseek()`.

As the name suggests, `fseek()` seeks the cursor to the given record in the file.

Syntax of fseek():

```
fseek(FILE * stream, long int offset, int whence);
```

The first parameter stream is the pointer to the file. The second parameter is the position of the record to be found, and the third parameter specifies the location where the offset starts.

Different whence in fseek()

Whence	Meaning
SEEK_SET	Starts the offset from the beginning of the file.
SEEK_END	Starts the offset from the end of the file.
SEEK_CUR	Starts the offset from the current location of the cursor in the file.

Example : fseek()

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

struct threeNum
{
   int n1, n2, n3;
};
int main()
{
   int n;
   struct threeNum num;
   FILE *fptr;

   if ((fptr = fopen("C:\\program.bin","rb")) == NULL){
       printf("Error! opening file");

       // Program exits if the file pointer returns NULL.
       exit(1);
 }
   // Moves the cursor to the end of the file
```

```c
    fseek(fptr, -sizeof(struct threeNum), SEEK_END);

    for(n = 1; n < 5; ++n)
    {
        fread(&num, sizeof(struct threeNum), 1, fptr);
        printf("n1: %d\tn2: %d\tn3: %d\n", num.n1, num.n2, num.n3);
        fseek(fptr, -2*sizeof(struct threeNum), SEEK_CUR);
    }
    fclose(fptr);

    return 0;
}
```

This program will start reading the records from the file `program.bin` in the reverse order (last to first) and prints it.

Random Access to File :

There is no need to read each record sequentially, if we want to access a particular record.C supports these functions for random access file processing.

1. **fseek()**
2. **ftell()**
3. **rewind()**

1. fseek():

This function is used for seeking the pointer position in the file at the specified byte.

Syntax: fseek(file pointer, displacement, pointer position);

Where
file pointer ---- It is the pointer which points to the file.
displacement ---- It is positive or negative.This is the number of bytes which are skipped backward (if negative) or forward(if positive) from the current position.This is attached with L because this is a long integer.

Pointer position:
This sets the pointer position in the file.

```
Value          pointer position
0     Beginning of file.
1     Current position
2     End of file
```

Ex:

1) fseek(p,10L,0)

0 means pointer position is on beginning of the file,from this statement pointer position is skipped 10 bytes from the beginning of the file.

2)fseek(p,5L,1)

1 means current position of the pointer position.From this statement pointer position is skipped 5 bytes forward from the current position.

3)fseek(p,-5L,1)

From this statement pointer position is skipped 5 bytes backward from the current position.

ftell():

This function returns the value of the current pointer position in the file.The value is count from the beginning of the file.

Syntax: ftell(fptr);

Where fptr is a file pointer.

rewind():

This function is used to move the file pointer to the beginning of the given file.

Syntax: rewind(fptr);
Where fptr is a file pointer.

Example program for fseek():

Write a program to read last ‘n’ characters of the file using appropriate file functions(Here we need fseek() and fgetc()).

```
void main()
{
      FILE *fp;
      char ch;
clrscr();
fp=fopen("file1.c", "r");
if(fp==NULL)
 printf("file cannot be opened");
else
{
   printf("Enter value of n  to read last ‘n’ characters");
   scanf("%d",&n);
   fseek(fp,-n,2);
   while((ch=fgetc(fp))!=EOF)
   {
        printf("%c\t",ch);
     }
   fclose(fp);
   getch();
}
```

OUTPUT: It depends on the content in the file.