Structure

Structure is a group of data elements grouped together under one name. These data elements, known as *members*, can have different types and different lengths. Data structures can be declared in C++ using the following syntax:

```
struct type_name { member_type1
member_name1; member_type2
member_name2; member_type3
member_name3;
.
.
} object_names;
```

Where **type_name** is a name for the structure type, **object_name** can be a set of valid identifiers for objects that have the type of this structure. Within braces **{ }**, there is a list with the data members, each one is specified with a type and a valid identifier as its name.

Example:

```
struct product {   int
weight;   double price;
} ;
product apple; product
banana, melon;
```

This declares a structure type, **called product**, and defines it having two members**: weight and price**, each of a different fundamental type.
This declaration creates a new type (product), which is then used to declare three objects (variables) of this type: **apple, banana, and**

melon. Note how once product is declared, it is used just like any other type.

Right at the end of the struct definition, and before the ending(;), semicolon
the optional field object_names can be used to directly declare objects of the

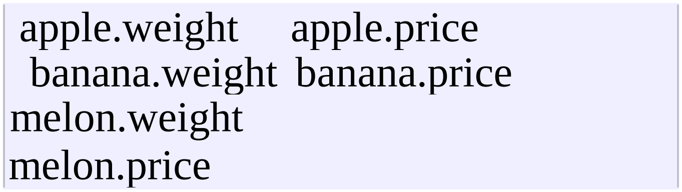
structure type. For example, the structure objects apple, banana, and melon can be declared at the moment the data structure type is defined:

```
struct product {
   int weight;
   double price;
} apple, banana, melon;
```

In this case, where object_names are specified, the type name (product) becomes optional: struct requires either a type_name or at least one name in object_names, but not necessarily both.

Once the three objects of a determined structure type are declared (apple, banana, and melon) its members can be .
accessed directly

The syntax for that is simply todot between the object name
insert a (.) and
the member name. For example, we could operate with any of these elements
as if they were standard variables of their respective types:



apple.weight apple.price
banana.weight banana.price
melon.weight
melon.price

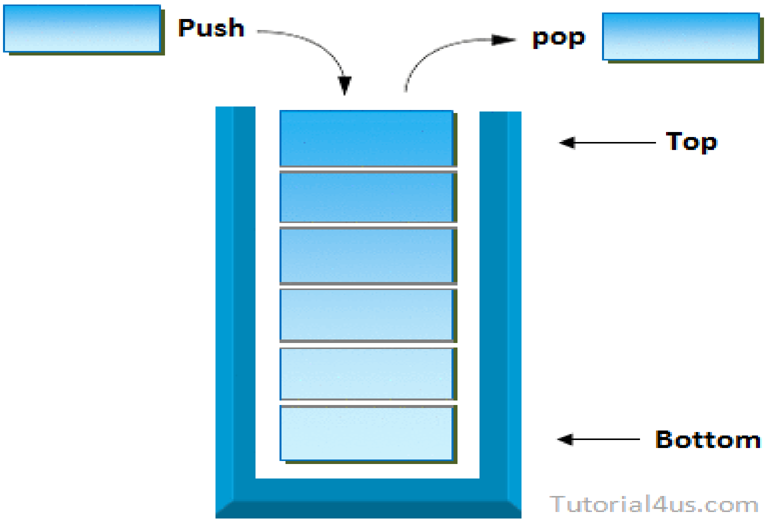
Example: Following is the example to explain usage of structure

```
#include <iostream>

#include <cstring>

using namespace std;

  struct Books {    char

title[50];             char

author[50];          char

subject[100];          int

book_id;

};   int main() {    struct Books Book1;        // Declare Book1

of type Book

   // book 1 specification    strcpy( Book1.title,

"Learn C++ Programming");    strcpy(

Book1.author, "Chand Miyan");     strcpy(

Book1.subject, "C++ Programming");

   Book1.book_id = 6495407;


   // Print Book1 info    cout << "Book 1 title : " <<

Book1.title <<endl;    cout << "Book 1 author : " <<

Book1.author <<endl;    cout << "Book 1 subject : " <<
Book1.subject <<endl;    cout << "Book 1 id : " <<

Book1.book_id <<endl;    return 0;

}
```

Stack in C++

Stack is linear data structure. In stack addition of new data item and deletion of already existing data item is done from only one end, known as **top**. Working of stack on the basis of :

- **Last-in-First-out (LIFO)** principal, it means last entered item remove first.
- **First In Last Out (FILO)** principle, it means first entered item remove last.



Real Life Example of Stack in C++

A most popular example of stack is plates in marriage party. Fresh plates are **pushed** onto to the top and **popped** from the top.

Implementation:

There are two ways to implement a stack:

- Using array
- Using linked list

Applications of stack:

- Balancing of symbols
- Infix to Postfix /Prefix conversion
- Redo-undo features at many places like editors, photoshop.
- Forward and backward feature in web browsers
- Used in many algorithms like Tower of Hanoi, tree traversals, stock span problem, histogram problem.
- Other applications can be Backtracking, Knight tour problem, rat in a maze, N queen problem and sudoku solver
- In Graph Algorithms like Topological Sorting and Strongly Connected Components

Stack Operation

In stack data structure mainly perform two operation; push and pop

- **pop:** In case of stack deletion of any item from stack is called pop. □ **push:** In case of stack Insertion of any item in stack is called push.

1. Insert Item in Stack in C++

- In case of stack Insertion of any item in stack is called **push**.
- In stack any item is inserted from top of the stack, When you insert any item in stack top will be increased by 1 as shown in figure 3 .

Algorithm for push

□ Initialization, set top=-1

□ Repeat step 3 to 5 until top < Max size - 1

□ Read, item

□ Set top=top+1

□ Set stack[top]=item

□ Print "stack overflow"

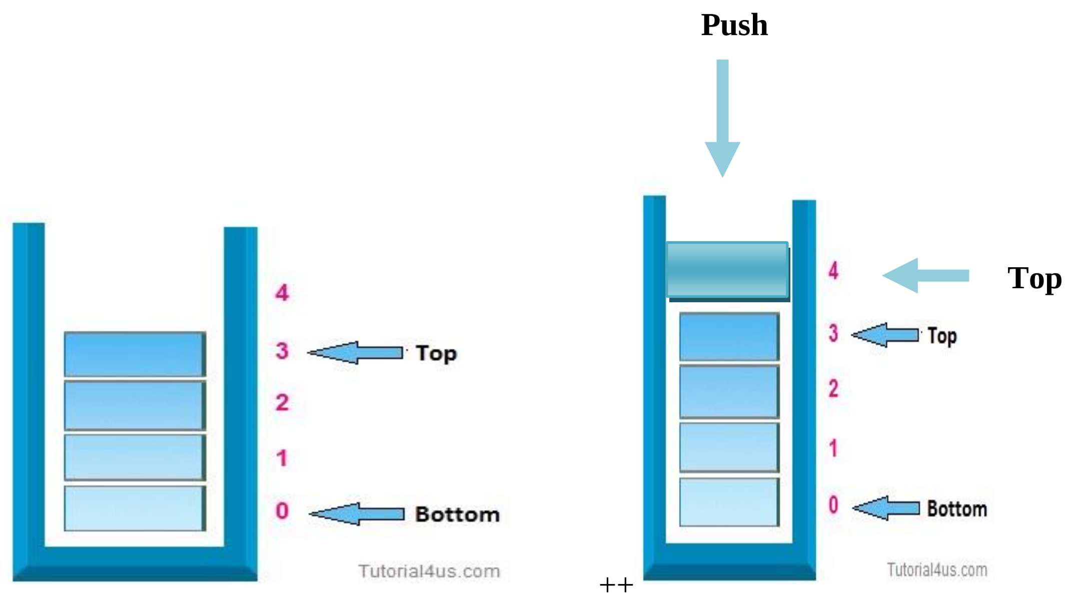


Figure (3): Push operation

Example: Push Item in Stack in C++ using array

```
void push(int  item)
{

if(top==size-1)
{
cout<<"\n Stack is full";
}
else
{
top=top+1;
cout<<"\n\n Enter element in stack: ";
stack[top]= item;
}
}
```

2. Delete data from stack in C++

- In case of stack deletion of any item from stack is called pop.
- In any item is delete from top of the stack, When you delete any item from stack top will be decreased by 1.

Algorithm for pop

- Repeated steps 2 to 4 until top>=0
- Set item=stack[top]
- Set top=top-1
- Print "Item deleted"
- Print "Stack under flow"

Example: Delete data from Stack in C++

```
void pop()
{
if(top==0)
{
cout<<"\nStack is empty: ";

}
else
{
item= stack[top];
top = top-1;
cout<<"deleted data is: " <<item;

}

}
```