PROGRAMMING FUNDAMENTALS

Let's explore technology together to live in the future

Structures in C++

Structure is a combination of multiple variables. A collection of variables that can be used with a single name.

```
        ┌──────────────→ Keyword
struct  student
               └──→ Structure Name
{
    int rollNo, marks;  ─┐
    float avg;          ─┼→ ]→ Elements,
    char grade;         ─┘     Members,
                               Fields of
};                             Structure.
```

Structure helps us to store multiple datatypes values together.

=> Structures are used to define new data types that may contain different types of data.

When a structure is declared, memory cells are not reserved in your own memory. You defined a datatype named 'Student'. When you declare new variabl with 'student' datatype, then memory cells are reserved, for sturcture variables.

Defining Structure Variables:

The structure variables can be defined after decleration of structure
The process of defining a structure variable is same as defining a var. basic types such as int, char.
The definition tells the compiler to allocate memory space for variable.
The compiler automatically allocates sufficient memory according to elements of structure.

Syntax:

$$\underbrace{\text{Struct\_Name}}_{\text{Name of Structure.}}\ \text{Identifier;} \xrightarrow{} \text{Name of variable to be defined.}$$

Example:

Student Rutab;
The statement allocates the memory space for all four members of structure.

Rutab

Roll no. Marks Average Grade

The structure var. Rutab will occupy 9 bytes in the memory.

The member variable RollNo occupy 2 bytes.
Marks - 2 bytes
Average - 4 bytes
Grade - 1 byte.

Accessing Members of Structure Var.

=> Any member of a structure variable can be accessed by using dot operator.
=> The name of the structure variable is written on the left side of dot.
=> The name of member variable is written on right side of dot.

```
Struct_Var.Mem_Var;
```

```
Student Rutab;
Rutab.RollNo = 10;
Rutab.Marks = 25;
Rutab.Average = 40;
Rutab. Grade = 'A';
```

* It uses four assignment statements to store different values in member variables of Rutab.

=> The same method can be used to input value in member variables and display these values.

```
cout << Rutab.RollNo;
cout << Rutab.Marks;
cout << Rutab.Average;
cout << Rutab. Grade;
```

```
cin >> Rutab.RollNo;
cin >> Rutab.Marks;
cin >> Rutab.Average;
cin >> Rutab.Grade;
```

```
#include <iostream>
 using namespace std;
  struct student
 {
     int rno;
     int marks;
     float avg;
     char grade;
   }; int main()
   {
      Student s;
      Cout << "Enter roll no.: ";
       cin>> s.ro;
      cout << "Enter marks";
       cin>> s.marks;
      cout << "Enter average:";
       cin >> s.avg;
      cout << "Enter grade:";
       Cin>> s.grade;

     cout<< "You entered following details:";
      Cout << "Roll no.: " << s.rno <<endl;
      Cout << "Marks: " << s.marks<<endl;
      cout << "Average:" << s.avg <<endl;
      cout << "Grade:" << s.grade <<endl;
    }
```

Scope of variables

- Lifetime of variable
- Area of usage.

$$\begin{tikzcd}
 & \text{Scope} \arrow[dl] \arrow[d] \arrow[dr] & \\
\text{global variable (i)} & \text{local variable (ii)} & \text{static variable (iii)}
\end{tikzcd}$$

i) starts and ends with program.
It can be used anywhere in program.

ii) starts and ends with function.
It can be used only in that function in which it is declared.

iii) Lifetime - existance as global
Usage -> local

```
   int sum=0 ——————→ accessed by all functions
void sum (int a , int b)     so it is global.
{
      sum = a+b;
}
   int main ()
   {
      Sum (42, 10);
      cout <<sum;
   }
```

local variable:

```
void sum (int a, intb)
{
    int sum =0;          → as it is declared
                           in function so it
     sum = a+b;             is local variable.
     return sum;            not available to
}                             others.
 int main()
 {
   cout << sum (42,10);
```

static variable:

```
 void sum (int a, intb)
 {
     static  int add = a+b;   → you can
 }                              display it
  int main()                    throughout the
  {                             Program.
     cout << sum (42,10);
      cout <<add;
```

Functions and Arrays

Array by default is "Pass by Reference."

```
void display (int arr [5])
{
   for (int i=0 ; i<=4 ;i++)
      cout << arr[i] <<"  ";
}
```

```
int main()
{
    int arr [5] = { 42, 36, 12, 43, 24 };

    display(arr);   // by default array is
                       pass by reference.
}
```

=> The address of index [0] is passed to display func. Variable '42' is passed. Only name is used while calling.

```
void disp-mid (int index)
{                    └→ we declare integer
    cout << index;      datatype as the
}                        value which is
 int main()              passed is of
 {                        integer datatype
    int arr[5] = { 42, 36, 12, 43, 24 };

    int mid = 2;

    disp-mid (arr [mid]);
 }                 └→ In this case it
                      is passed by
                      value the value
                      of array at mid
                      is passed as
                      variable.
```

Function Overloading:

↳ same name but different parameters number or sequence.

1)
```
int sum (int a, int b);
int sum (int a, int b , int c);
int sum (int a , int b, int c , int d );

int main()
{
  sum (12, 24);
  sum ( 13, 24, 66 , 89);
  sum ( 13, 14, 15);
}
```

2)
```
int sum (float a , int b);
int sum (int a , float b, int c);
int sum (int a , float b);

int main()
{
   sum (43.6 , 28)
   sum (29, 43, 82);
   sum (43, 28);
}

int sum (int a , float b)
{
   // code
}
```

Local Variable:

A variable declared inside a function is known as local variable.

Scope: Local variable can be used only in the function in which it is declared. If a statement accesses a local variable that is not in scope, it is an error.

Lifetime: The lifetime of local variable starts when the control enters the function in which it is declared. Local variable is automatically destroyed when control exits from the function and its lifetime ends. When the lifetime of local variable ends, the value stored in this variable also become inaccessible.

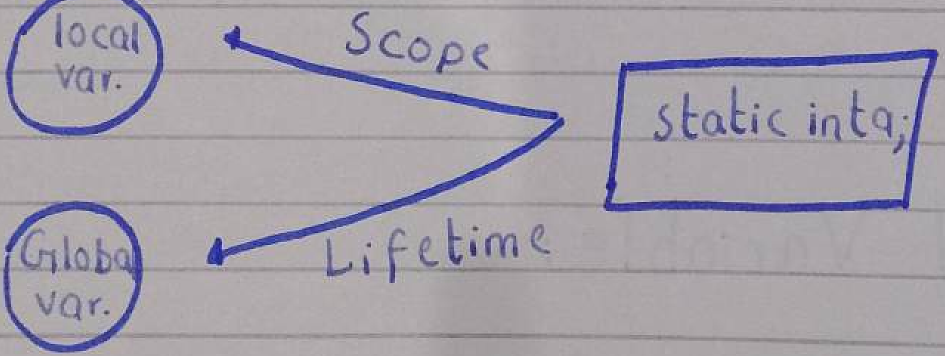
Global Variable:

A variable declared outside any function is known as global variable. It can be used by all functions in program. The values of these variables are share among different functions. If one function changes the value of global variable, this change is also available to other functions.

Scope: Global variables can be used by all functions in the program. It means that these variables are globally accessed from any part of program.

Lifetime:
Global variables exist in memory as long as the program is running. These variables are destroyed from memory when program terminates. Occupy memory longer than local variables.

Static Variable



Functions created with static variables perform better.
Static variables are declared and initialized only even once if the function is called repeatedly.

Initializing Structure Variable:

$$\text{Student Rutab} = \{\underbrace{10, 585, 65.5, \text{'B}}_{\substack{\text{Values should be} \\ \text{according to member} \\ \text{variables of structu}}}\};$$

Assigning one structure variable to other

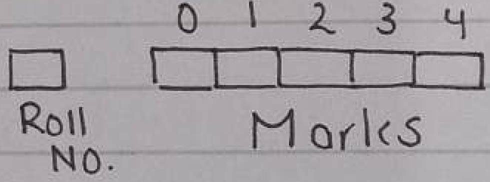
```
struct student
{
    int RollNo , Marks;
    float Average;
     char Grade;
  };

        Student Usman = { 10, 560, 65.5, 'B'
        Student Abdullah = Usman,
```

```
# include <iostream>
using namespace std;
int main ()
 struct Phone
{
    int ncode;
    int acode;
    long number;
 };
```

```
int main()
{
   Phone p1, p2 = {92, 41, 9220083};
   cout << "Enter national code:";
   cin >> p1.ncode;
   cout << "Enter area code:";
   cin >> p1.acode;
   cout << "Enter phone number:";
   cin >> p1.number;
   cout << "Phone Number 1: +";

 Cout << p1.ncode << "-" << p1.acode << "-" << p1.number

  Cout << "Phone Number 2: +";

 cout << p2.ncode << "-" << p2.acode << "-" << p2.number
}
#include <iostream>
using namespace std;
struct Marks
{
   int m;
   char g;
};
int main()
{
     Marks a, b;
cout << "Enter marks:";
 cin>> a.m ;
cout<< "Enter grade:";
cin>> a.g;
 b=a;

cout<<"The first record is as follows:"\n";
cout <<"Marks ="<<a.m <<endl;
cout << "Grade=" << a.g <<endl;

cout<<"The second record is as follows: \n";
 cout<<"Marks:"<< b.m <<endl;
 cout<< "Grade:"<< b.g <<endl;

}
```

Array as Member of Structure:

```
struct Student
{
    int RollNo;
    int Marks[5];
}
```



Accessing array elements:

```
Student Usman;
Usman.RollNo=10;
Usman.Marks[0]=85;
Usman.Marks[1]=90;
Usman.Marks[2]=87;
Usman.Marks[3]=90;
Usman.Marks[4]=78;
```

Initializing a Structure with array as member.

$$\text{Student Usman} = \{10, \underbrace{\{85, 95, 78, 81, 90\}}_{\text{Array}}\};$$

```
#include <iostream>
 using namespace std;

  struct Test
 {    int  rno;
      int  m[5];
   };
   int main()
   {  Test r;
      int i , t=0;
      float avg =0.0;
      cout << " Enter roll no. ";
     cin >> r.rno;

     for(i=0 ; i<5, i++)
     {  cout << "Enter marks: ";
          cin>> r.m[i] ;
           t  =  t + r.m [i];
     }
        avg = t / 5.0;
cout << "Roll No." << r.rno << endl;
cout << "Total marks : " << t << endl;
cout << "Average: " << avg << endl;

}
```

Array of Structures:

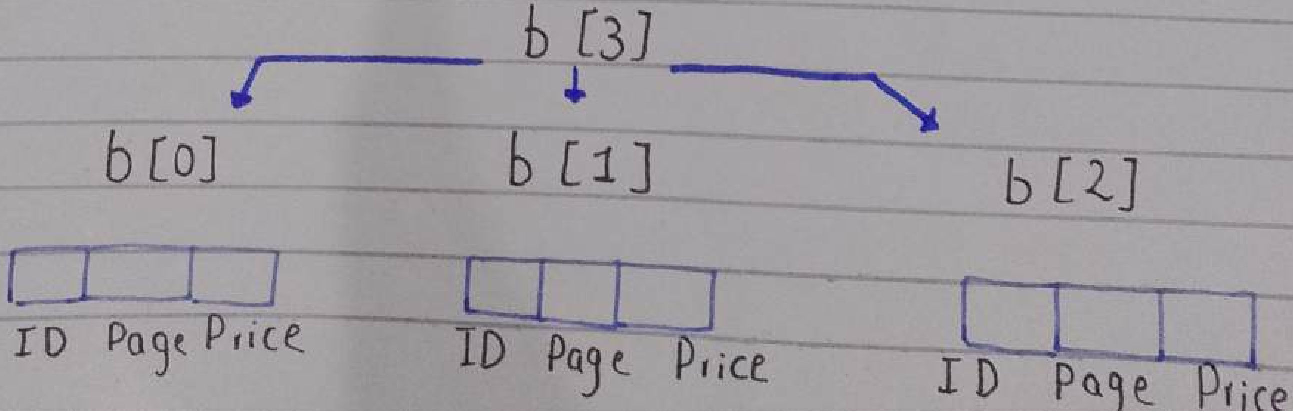
Array is a collection of same datatype.

The Structures are used to define new datatypes.

Suppose you have some books and you want to store information about books. such as ID, No. of Pages, Price.

```
struct Book
{
    int BookID;
    int pages;
    float price;
}
```

Book b[3]; ⟶ Array of Structures



Accessing array of Structures:

```
b [0] . BookID = 1;
b [0] . Pages  = 230;
b [0] . Price  = 250;
```

Initializing array of Structures:

```
struct Book
{
  int BookID;
  int Pages;
  float Price;
};
 Book b[3] = {{1,250,150.0},{2,50,165.0},
                  {3,350,210.50}};
```

```
#include <iostream>
using namespace std;

struct Book
{
     int id;
     int pages;
     int price;
 };
```

```
int  main()
{
     Book b[5];
      int i ,max ,m;

     for (int i=0; i<5;i++)
      {
        cout<< "Enter Book ID:";
         cin>> b[i].id;
        cout << "Enter Pages:";
         cin>> b[i].pages;
         cout << "Enter Price:";
          cin>> b[i].price;
      }
        max = b[0].price;
         m =0;

     for (int i=0 ;i<5; i++)
          {
          if (b[i].price >max)
          {
             max = b[i].price;
              m = i;
          }
       }
    cout<<"The most costly Book:";
    cout << "Book ID:" <<b[m].id <<endl;
    cout << "Pages:" <<b[m].pages<<end
    cout << "Price:" << b[m].price<<endl;
}
```

Nested Structure:

A structure within a structure is called nested structure.
A nested structure is created when the member of a structure is itself a structure

```
struct A
{
    int n;
    float b;
};
struct b
{
    char c;  ← B rec;
    A x;
};
```

★ The member variables of nested structure can be accessed using multiple dot op.

Suppose we created a structure variable 'B rec'. which involves 'c' of struct b and also 'b' and 'n' of struct A, because struct A is itself a member of struct b.

```
rec.c~~.c~~ = 'A';
rec.x.n = 10;
rec.x.b = 15.5;
```

Initializing Nested Structure

```
Brec = {'A', {10, 50.25}};
```

```
#include <iostream>
 using namespace std;
  struct date
  {
     int day;
     int month;
     int year;
  };
    struct phonebook
    {
      char name [40];
      char city [40];
      long tel;
      date birthday;
    };
      int main()
    {
       phonebook a1;
       cout <<"Enter name:";
        cin>>a1.name;
        cout <<"Enter city:";
         cin>>a1.city;
        cout << "Enter phone number:";
         cin>>a1.tel;
     cout <<"Enter date of birth dd-mm-yy:";
     cin>>a1.birthday.day;
     cin>>a1.birthdg.month;
     cin>>a1.birthday.year;
    cout <<"Size of structure var. is:"
                 << sizeof(a1);
```

```
cout<<"The entry made is:";
cout<<a1.name<<"-"<<a1.city<<"-"<<a1.tel;

cout<<"Birthdg is on:"<<a1.birthday.day
      <<"-"<<a1.birthday.month<<"-"<<
      a1.birthday.year;
```

Unions:

Unions are used to group together variables of different datatype. Unions can be used to create user defined datatypes just like struc.

```
union student
{
  int RollNo;
  int Marks;
  char Grade;
};
  Student a;
```

Union

a. RollNo
4 bytes

a. Marks
a. Grade

Recursion

A function calling itself is called recursion and such function is known as ~~r~~ recursive function.
Counter part of loop.

$4! \rightarrow$ factorial
$\downarrow 4 \times 3 \times 2 \times 1 = 24$

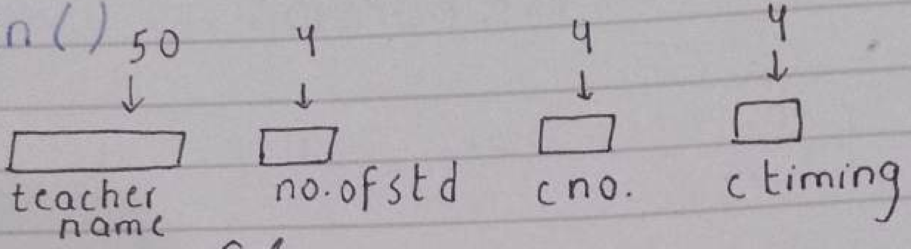
```
int factorial ( int n
{   int fact =0;
    if (n>1)
    { fact = factorial(n-1);      ← Function call (Recursion)
          return n* fact ;}
}                           ↳ else
                                 return 1;
 int main ()
 {
    int n;
    cout << "Enter no.:";
     cin >>n;
   cout << "Factorial of n=" <<factorial(n);    ← Function call
```

Stack
call - 4 -> 1
call - 3 -> 2
call - 2 -> 3
call - 1 -> 4

Structure

Structure is a user defined datatypes.
It can store multiple datatypes.

```
 struct class
 {
      char teacher_name[50];   }
      int  no_of_stud;         }  Structure
      int  c_no;               }  members
      int  c_timing;           }
 };
```

```
int main()
{
```


// Example

Class

```
Datytype var_name;   → new datatype
class   c1, c2, c3;  → 62 bytes

class c1 = {{'H','U','M'}, {15}, {13}, {100}};
```

Accessing C1 members :

```
int main()
{
   class c1;
   cout << "Enter teaacher name!";
   cin.getline (c1.teacher_name, 15);
```
. → member access operator.

```
   cout << "Enter class timing!";
   cin >> c1.c_teming;

   cout << "Enter no. of students!";
   cin >> c1.no_of_stud;

   cout << "Enter class number!";
   cin >> c1.c_no;

   cout << "Teacher Name:";
   cout << c1.teacher_name;
```

```
        cout << "No. of students";
         cout << cl.no-of-stud;

        cout << "Class room no. ";
         cout << cl.cno;

        cout << "Class timing :";
        cout << cl.c-timing ;

    }


 struct student
{
 char s-name;
  int s-id;
  int  marks;
 };

   int main ()
  {
     student s1, s2, s3, s4, s5;

     cout<<"Enter student name:";
      cin:getline (s1.name ,10);

      cout << "Enter Student ID:";
      cin >> s1.id ;
      cout<<"Enter student marks:";
      cin>>s1.marks;

     cout << "Enter student name:";
     cin.getline (s2.name ,10);
```

```
cout<< "Enter Student ID:";
cin>> S1.id;
cout << "Enter Student marks:";
cin>> S1.marks;

cout <<"Enter student name:";
cin.getline ( S3.name , 10);
cout<< "Enter student ID:";
cin>> S3.id;
cout << "Enter student marks:";
cin>> S3.marks;

cout<<"Enter student name:";
cin.getline (S4.name, 10);
Cout <<"Enter student ID:";
cin>>S4.id;
cout <<"Enter student marks:";
cin>>S4.marks;

cout <<"Enter student name:";
cin.getline (S5.name, 10);
Cout <<"Enter student ID: ";
```