Intermediate programming(C++)-

Lab 7: { Structures, Enum }

Content

Structures

- Introduction
- Initialization and declaration
- Structures with functions (Passing/ returning)
- Structures and arrays
- Structures and pointers

Enumerations

Structures:: Introduction

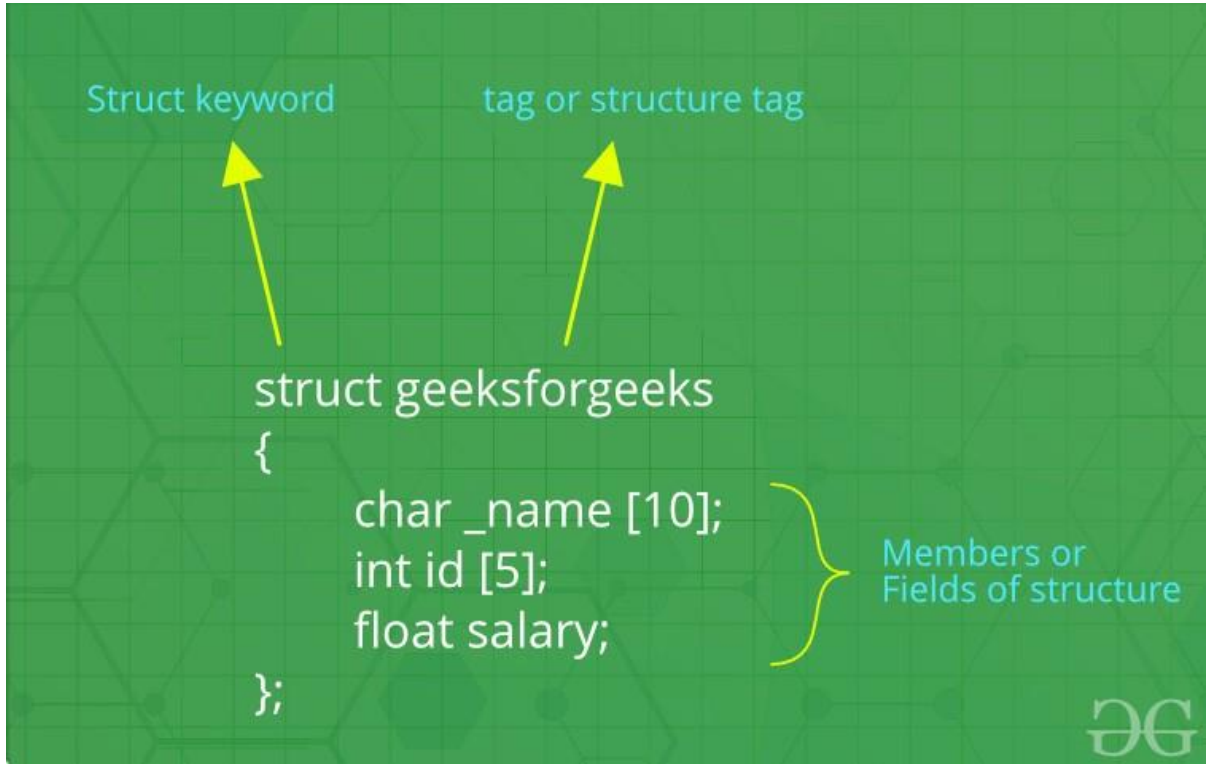
User-defined data type used when we want to store many variables of different data types.

Structure members are:

- Data member
- Member functions

Syntax:

```
Struct structureName{
    member1;
    member2;
    member3;
    .
    .
    .
    memberN;
};
```



Note: Structure itself doesn't allocate in the memory, meaning that it don't have size in memory

Example 1:: Initialization/ declaration

```
// A variable declaration with structure declaration.
struct Point {
   int x, y;
 } p1;   // The variable p1 is declared with 'Point'

// A variable declaration like basic data types; Another way
struct Point {
   int x, y;
};

int main() {
   Point p1; // The variable p1 is declared like a normal variable
   // Initialize variables inside the struct
   p1.x = 20;
   p1.y = 40;

   // or
   p1 = {90, 56};
}
```

Example 2:: Default values

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

// Compilation error before C++11
// After C++11, these are considered as default values
struct Point {
   int x = 0;
   int y = 1;
};

int main() {
   Point p;
   cout << "Default values: \nx = " << p.x << ", y = " << p.y << endl;
   p.x = 10;
   p.y = 11;
   cout << "After changing values: \nx = " << p.x << ", y = " << p.y << endl;
}
```

Output:

```
Default values:
x = 0, y = 1
After changing values:
x = 10, y = 11
```

Example 3:: Structures with functions:: Struct as a Parameter

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

struct Point {
   int x = 0;
   float y = 1.1;
   char c = 'a';
};

void display_x_y(const Point& p) {
   cout << "X: " << p.x << ", y = " << p.y << ", c: " << p.c;
}

int main() {
   Point p2 = { 78, 190.8, 'e'};
   display_x_y(p2);
}
```

Note: We can write the function inside the struct and use it in main in this way:p2.display_x_y();

Example 3:: Function inside the struct

```
struct Student {
   string fname = " ";
   string sname = " ";
   float gpa = 0.0;
   string major = " ";

   void display() {
      cout << "fname: " << fname << "\nsname: " << sname << "\nGPA: " << gpa <<
                  "\nMajor: " << major << endl;
   }
};

int main() {
   Student s;
   s.fname = "Ali";
   s.sname = "Ashraf";
   s.gpa = 3.6;
   s.major = "AI";
   s.display(); }
```

Output:

```
fname: Ali
sname:
Ashraf GPA:
3.6 Major: AI
```

Example 3:: Structures with functions:: Struct as a return value

```
struct Point {
   int x = 0;
   float y = 1.1;
   char c = 'a';
};

void display_struct(const Point& p) {
   cout << "X: " << p.x << ", y = " << p.y << ", c: " << p.c;
}

Point get_struct() {
   int x;
   float y;
   char c;

   cout << "Please enter integer: ";
   cin >> x;
   cout << ",,,float number: ";
   cin >> y;
   cout << ",,,Character: ";
   cin >> c; return Point{ x, y, c };
}
```

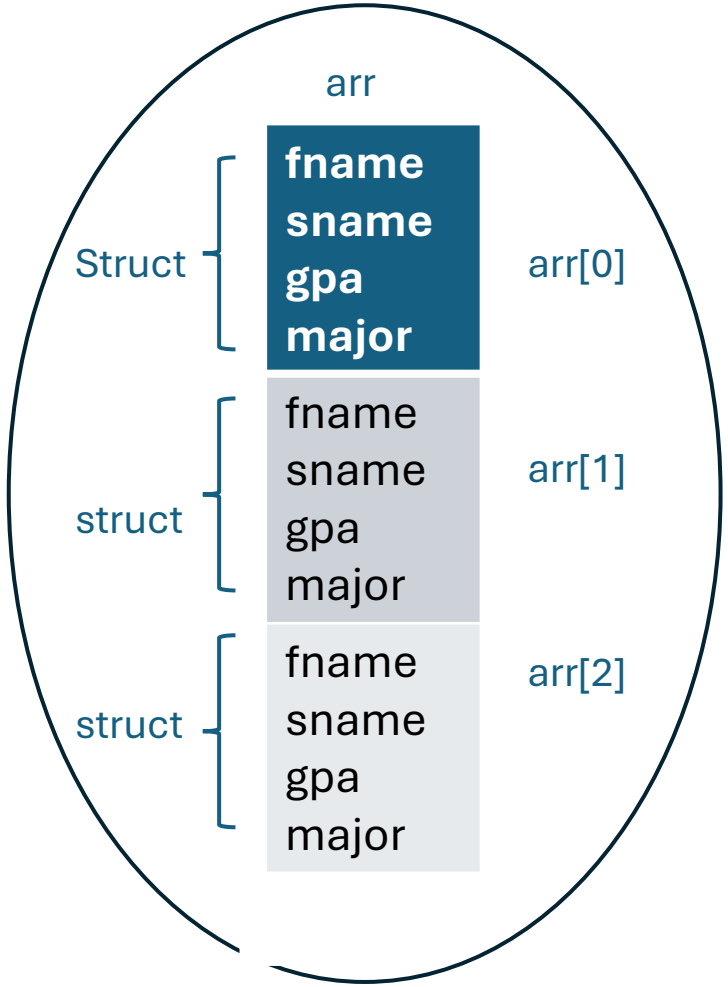
```
int main() {
   Point out = get_struct();
   cout << endl << endl;
   display_struct(out);
}
```

Example 4:: Array of structures

```
struct Student {
   string fname, sname, major;
   float gpa;
};

int main() {
   student arr[3];
   arr[0] = { "Ahmed", "Hesham", "AI", 4 };
   arr[1] = { "Hamed", "Salah", "BMD", 3.5 };
   arr[2] = { "Nada", "Mostafa", "CS", 3.99 };

   for (int i = 0; i < 3; i++) {
      cout << "Student name: " << arr[i].fname + " " + arr[i].sname <<
         "\nHis GPA: " << arr[i].gpa <<
         "\nHis Major: " << arr[i].major << endl;
   }
}
```



Example 5:: Pointer to struct

```
struct Student {
   string fname;
   string sname;
   float gpa;
   string major;
};

int main() {
   Student s;
   Student* p = &s;
   p->fname = "Ali";
   (*p).sname = "Ashraf";
   (*p).sname = 3.6;
   (*p).major = "AI";
   cout << "Student name: " << p->fname + " " + p->sname <<
                ", GPA: " << p->gpa << ", and his major is: " << p->major;
}
```

Output:

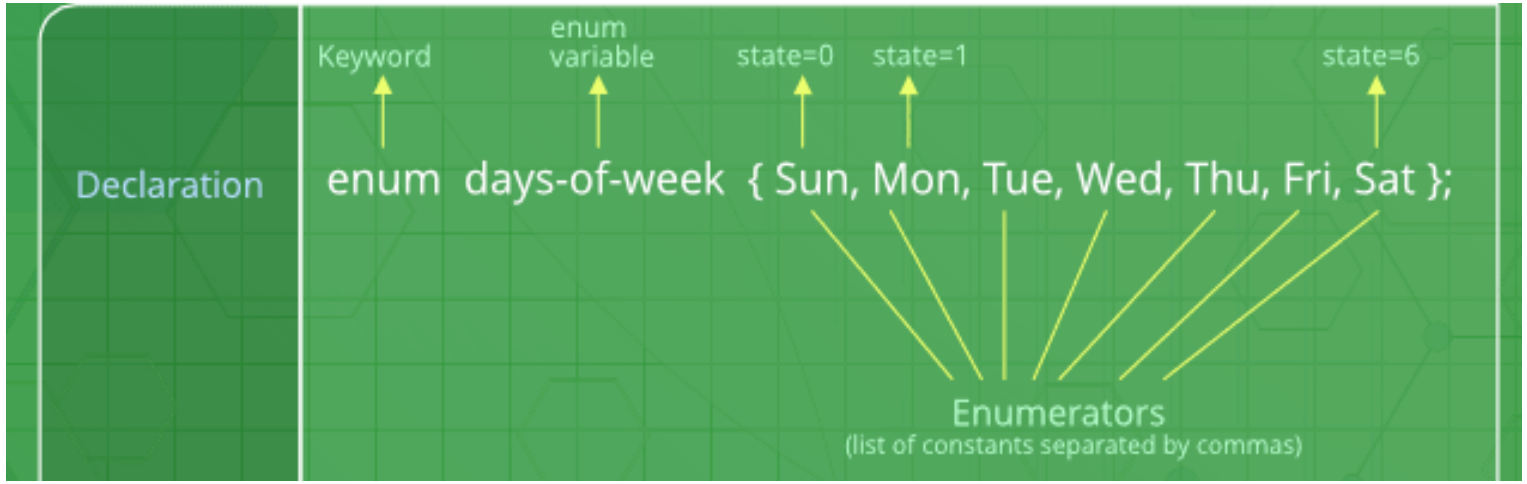
Student name: Ali Ashraf, GPA: 3.6, and his major is: AI

Enum

User-defined data type used to store elements in ordered manner.

Syntax:

```
enum enum_name { element 1, element 2,
element 3, ..............};
```



Example

```
#include <iostream>
using namespace std;
int main() {
   enum State { Working = 1, Failed = 0, Freezed = 0 };
   enum week { Mon, Tue, Wed, Thur, Fri, Sat, Sun };
   enum months { Jan = 1, Feb, March = 3 };
   cout << Working << " " << Failed << " " << Freezed << endl;
   cout << Jan << " " << Feb << " " << March << endl;

   return 0;
}
```

Output:

```
1 0 0
1 2 3
```

Task:

Objective:

Create a simple C++ program that uses structures and enumerations to store and display information about an employee, including their name, age, and employment status.

Task Description:

Create an enumeration for Employee Status: Define an enumeration called Status with three possible values:

- Active
- OnLeave
- Retired

Create a structure for Employee Information: Define a structure called Employee with the following fields:

- name (string): The name of the employee.
- age (int): The age of the employee.
- status (Status): The employment status, which will use the Status enumeration.

Collect Input:

Create variables of type Employee and collect data for one employee, including:

- Name (string)
- Age (integer)
- Status (choose from the enumeration: Active, OnLeave, or Retired)

Display Output:

Display the employee's name, age, and status in a clear format.