Structure and Pointers

In this lesson, you will learn how to define a pointer to the user-defined data type.

We'll cover the following

C++ structure pointer

We have already learned that there are pointers to built-in data types such as `int`, `char`, `double`, etc.

Like we have a pointer to `int`, we can also have pointers to the user-defined data types such as structure.

---

*The pointer that stores the address of the structure variable is known as **structure pointer**.*

Declaring structure pointer

The basic syntax for declaring a structure pointer is given below:

```
struct  structure_name  *ptr ;
```

Example program

See the program given below:

```
#include <iostream>

using namespace std;

// Student structure
struct Student {
  string name;
  int roll_number;
  int marks;
};

// main function
int main() {
  // Declare structure variable
  struct Student s1;
  // Declare structure pointer
  struct Student *ptrs1;
  // Store address of structure variable in structure pointer
  ptrs1 = &s1;
  return 0;
}
```

Explanation

Line No. 15: Declares a variable `s1` of type `Student`

Line No. 17: Declares a pointer variable `ptrs1` of type `Student`

Line No. 19: Stores the address of `s1` in `ptrs1`

Accessing structure members through a structure pointer

We can access members of structure through a structure pointer in two ways:

- Indirection and the dot operator
- Arrow operator

Indirection and dot operator

The basic syntax for accessing the structure members through the structure pointer is given below:

(*StructurePointer) . StructureMember ;

To access the members of the structure variable to which the structure pointer is pointing, we will first use the dereference operator with a structure pointer, which is followed by the dot operator and the member whose value you want to access.

Example program

RUN the program given below and see the output!

```
#include <iostream>

using namespace std;

// Student structure
struct Student {
  string name;
  int roll_number;
  int marks;
};

// main function
int main() {
  // Declare structure variable
  struct Student s1;
  // Declare structure pointer
  struct Student *ptrs1;
  // Store address of structure variable in structure pointer
  ptrs1 = &s1;

  // Set value of name
  (*ptrs1).name = "John";
  // Set value of roll_number
  (*ptrs1).roll_number = 1;
  // Set value of marks
  (*ptrs1).marks = 50;

  // Print value of structure member
  cout << "s1 Information:" << endl;
  cout << "Name = " << (*ptrs1).name << endl;
  cout << "Roll Number = " << (*ptrs1).roll_number << endl;
  cout << "Marks = " << (*ptrs1).marks << endl;


  return 0;
}
```

Explanation

In the code above, `ptrs1` is pointing to `s1`. So, by dereferencing the `ptrs1`, we get the content of `s1`. It means `*ptrs1` is equivalent to `s1`. Therefore, to access the

members of the structure variable, we write `*ptr`, which is followed by a dot operator `.` and a structure member.

> ℹ The precedence of dot operator `.` is greater than the precedence of indirection operator `->`. Therefore, it is necessary to put brackets around the asterisk, which is followed by a pointer name.

Arrow operator

You must be thinking the above method of accessing the structure members using a pointer is quite confusing! So, can't we just access the structure members using one simple operator?

Yes, we can. Here is where the arrow operator comes in!

StructurePointer -> StructureMember ;

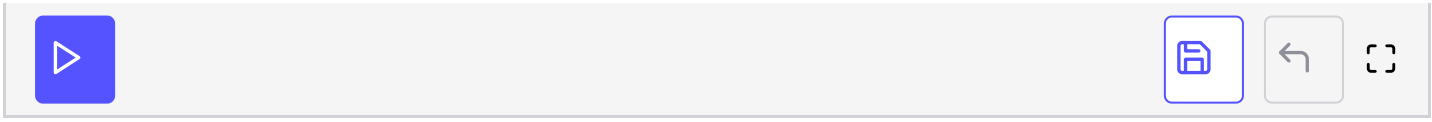
To access the structure members using the arrow operator, we have to write the name of the structure pointer followed by an arrow operator `->`, which is further followed by structure member and semicolon.

Example program

Let's see the use of the arrow operator!

```
#include <iostream>

using namespace std;

// Student structure
struct Student {
  string name;
  int roll_number;
  int marks;
};

// main function
int main() {
  // Declare structure variable
  struct Student s1;
  // Declare structure pointer
  struct Student *ptrs1;
  // Store address of structure variable in structure pointer
  ptrs1 = &s1;
```

```
  // Set value of name
   ptrs1->name = "John";
  // Set value of roll_number
   ptrs1->roll_number = 1;
  // Set value of marks
   ptrs1->marks = 50;

  // Print value of structure member
  cout << "s1 Information:" << endl;
  cout << "Name = " << ptrs1->name << endl;
  cout << "Roll Number = " << ptrs1->roll_number << endl;
  cout << "Marks = " << ptrs1->marks << endl;


  return 0;
}
```



> `(*ptrs1).name` and `ptrs1->name` are functionally equivalent.

What is the output of the following code?

```
struct Student {
  string name;
  int roll_number;
  int marks;
};

int main() {
  struct Student s1;
  struct Student *ptrs1;
  ptrs1 = &s1;
  *ptrs1.name = "John";
  cout << "Name = " << *ptrs1.name << endl;
```

Retake Quiz

This sums up our discussion of structures. Let’s dive right in and solve some challenges related to structures.