2. C++ Structures (struct)

Structures (also called structs) are a way to group several related variables into one place. Each variable in the structure is known as a **member** of the structure.

Create a Structure

To create a structure, use the `struct` keyword and declare each of its members inside curly braces.

After the declaration, specify the name of the structure variable (**myStructure** in the example below):

```
struct {             // Structure declaration
  int myNum;         // Member (int variable)
  string myString;   // Member (string variable)
} myStructure;       // Structure variable
```

Access Structure Members

To access members of a structure, use the dot syntax (`.`):

Example

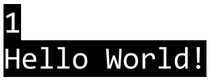
Assign data to members of a structure and print it:

```
// Create a structure variable called myStructure
struct {
  int myNum;
  string myString;
}

myStructure;

// Assign values to members of myStructure
myStructure.myNum = 1;
myStructure.myString = "Hello World!";
```

```
// Print members of myStructure
cout << myStructure.myNum << "\n";
cout << myStructure.myString << "\n";
```

#include <iostream>
#include <string>
using namespace std;
int main() {
  struct {
    int myNum;
    string myString;
  } myStructure;
	myStructure.myNum = 1;
	myStructure.myString = "Hello World!";
	cout << myStructure.myNum << "\n";
	cout << myStructure.myString << "\n";
	return 0;
	}
		
		1
		Hello World!

One Structure in Multiple Variables

You can use a comma (,) to use one structure in many variables:

```
struct {
  int myNum;
  string myString;
} myStruct1, myStruct2, myStruct3; // Multiple structure variables
separated with commas
```

This example shows how to use a structure in two different variables:

Example

Use one structure to represent two cars:

```
struct {
  string brand;
  string model;
  int year;
} myCar1, myCar2; // We can add variables by separating them with a comma
here

// Put data into the first structure
myCar1.brand = "BMW";
myCar1.model = "X5";
myCar1.year = 1999;

// Put data into the second structure
myCar2.brand = "Ford";
myCar2.model = "Mustang";
myCar2.year = 1969;

// Print the structure members
cout << myCar1.brand << " " << myCar1.model << " " << myCar1.year << "\n";
cout << myCar2.brand << " " << myCar2.model << " " << myCar2.year << "\n";
```

Output is:

```
BMW X5 1999
Ford Mustang 1969
```

Named Structures

By giving a name to the structure, you can treat it as a data type. This means that you can create variables with this structure anywhere in the program at any time.

To create a named structure, put the name of the structure right after the `struct` keyword:

```
struct myDataType { // This structure is named "myDataType"
  int myNum;
  string myString;
};
```

To declare a variable that uses the structure, use the name of the structure as the data type of the variable:

```
myDataType myVar;
```

Example

Use one structure to represent two cars:

```
// Declare a structure named "car"
struct car {
  string brand;
  string model;
  int year;
};

int main() {
  // Create a car structure and store it in myCar1;
  car myCar1;
  myCar1.brand = "BMW";
  myCar1.model = "X5";
  myCar1.year = 1999;

  // Create another car structure and store it in myCar2;
  car myCar2;
  myCar2.brand = "Ford";
  myCar2.model = "Mustang";
  myCar2.year = 1969;

  // Print the structure members
  cout << myCar1.brand << " " << myCar1.model << " " <<
myCar1.year << "\n";
  cout << myCar2.brand << " " << myCar2.model << " " <<
myCar2.year << "\n";

  return 0;
}
```

Output is:

```
BMW X5 1999
Ford Mustang 1969
```

Another Example:

Certainly! In C++, you can use a structure to define a custom data type that holds multiple variables. Here's an example of a program that demonstrates how to use a structure to define a student record with multiple variables:

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

// Define a structure for student record
struct Student {
    string name;
    int age;
    float gpa;
};

int main() {
    // Declare multiple variables of type Student
    Student student1, student2;

    // Assign values to the members of student1
    student1.name = "Ahmed";
    student1.age = 20;
    student1.gpa = 3.5;

    // Assign values to the members of student2
    student2.name = "Maha";
    student2.age = 21;
    student2.gpa = 3.8;
    // Print the details of student1
    cout << "Student 1 Details:" << endl;
    cout << "Name: " << student1.name << endl;
    cout << "Age: " << student1.age << endl;
    cout << "GPA: " << student1.gpa << endl;

    cout << endl;

    // Print the details of student2
    cout << "Student 2 Details:" << endl;
    cout << "Name: " << student2.name << endl;
    cout << "Age: " << student2.age << endl;
    cout << "GPA: " << student2.gpa << endl;
    return 0;
}
```

Output is:

```
Student 1 Details:
Name: Ahmed
Age: 20
GPA: 3.5

Student 2 Details:
Name: Maha
Age: 21
GPA: 3.8
```

In this program, we define a structure `Student` with three members: `name`, `age`, and `gpa`. Then, we declare two variables of type `Student`, namely `student1` and `student2`. We assign values to the members of each student using the dot operator (`.`), and finally, we print out the details of each student.

3. C++ References

Creating References

A reference variable is a "reference" to an existing variable, and it is created with the & operator:

```
string food = "Pizza";  // food variable
string &meal = food;    // reference to food
```

Now, we can use either the variable name food or the reference name meal to refer to the food variable:

Example

```
string food = "Pizza";
string &meal = food;

cout << food << "\n";  // Outputs Pizza
cout << meal << "\n";  // Outputs Pizza
```

Another Example:

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

int main() {
    int num = 10;

    // Creating a reference to the variable 'num'
    int &numRef = num;

    // Printing the original value of 'num' and the value through the
reference 'numRef'
    cout << "Original value of num: " << num << endl;
    cout << "Value through reference numRef: " << numRef << endl;

    // Modifying the value of 'num' through the reference 'numRef'
    numRef = 20;

    // Printing the modified value of 'num' and the value through the
reference 'numRef'
```

```
    cout << "Modified value of num: " << num << endl;
    cout << "Value through reference numRef: " << numRef << endl;

    return 0;
}
```

in this program:

- We declare an integer variable `num` and initialize it with the value 10.
- We create a reference `numRef` to the variable `num`. This means that `numRef` refers to the same memory location as `num`.
- We print the original value of `num` and the value of `num` through the reference `numRef`.
- We modify the value of `num` through the reference `numRef`.
- We print the modified value of `num` and the value of `num` through the reference `numRef`.

The output of this program will be:

```
Original value of num: 10

Value through reference numRef: 10

Modified value of num: 20

Value through reference numRef: 20
```

This demonstrates how references in C++ provide an alias or alternative name for a variable, allowing you to manipulate the variable indirectly.

C++ Memory Address

n the example from the previous page, the & operator was used to create a reference variable. But it can also be used to get the memory address of a variable; which is the location of where the variable is stored on the computer.

When a variable is created in C++, a memory address is assigned to the variable. And when we assign a value to the variable, it is stored in this memory address.

To access it, use the & operator, and the result will represent where the variable is stored:

Example

```
string food = "Pizza";

cout << &food; // Outputs 0x6dfed4
```

Note: The memory address is in hexadecimal form (0x..). Note that you may not get the same result in your program.

And why is it useful to know the memory address?

References and **Pointers** (which you will learn about in the next chapter) are important in C++, because they give you the ability to manipulate the data in the computer's memory - **which can reduce the code and improve the performance**.

These two features are one of the things that make C++ stand out from other programming languages, like Python and Java.