TUTORIAL 4: ARRAY,POINTERS, STRUCTURE IN C++

Concept of ARRAY is same as used in C.

Example: Create and change Array element

```
string cars[4] = {"Volvo", "BMW", "Ford", "Mazda"};

cars[0] = "Opel";

cout << cars[0];
```

OUTPUT:

Now outputs Opel instead of Volvo

Here important thing is **C++** supports for-each loop function for ARRAY. Lets see

Different way to print elements of ARRAY

EXAMPLE: 1. Static way

```
int myNumbers[5] = {10, 20, 30, 40, 50};

for (int i = 0; i < 5; i++) {

  cout << myNumbers[i] << "\n";

}
```

Example: 2. Intelligent way

```
int myNumbers[5] = {10, 20, 30, 40, 50};

for (int i = 0; i < sizeof(myNumbers) / sizeof(int); i++) {

  cout << myNumbers[i] << "\n";

}
```

Note that, in C++ version 11 (2011), you can also use the "for-each" loop:

Example

```
int myNumbers[5] = {10, 20, 30, 40, 50};
for (int i : myNumbers) {
  cout << i << "\n";
}
```

Multi-Dimensional Arrays

A multi-dimensional array is an array of arrays.

Example of two dimension ARRAY

As with ordinary arrays, you can insert values with an array literal - a comma-separated list inside curly braces. In a multi-dimensional array, each element in an array literal is another array literal.

```
string letters[2][4] = {
  { "A", "B", "C", "D" },
  { "E", "F", "G", "H" }
};
```

Accessing array element :

```
cout << letters[0][2];  // Outputs "C"
```

STRUCTURE:

C++ Structures

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.

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.

EXAMPLE:

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

In above example myStructure is variable which is struct datatype. This is just direct declaration of Variable,but structure has no name.

Structure NAME:

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.

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

Here structure name is myDataType.

Create variable : *myDataType myVar;*

EXAMPLE CODE: USE OF STRUCTURE

```
// 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;
}
```

REFERENCE:

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
```

You can change original variable value from reference also.

Example:

```
string food = "Pizza";
string &meal = food;
meal = "Dhokla";
cout << food << "\n";  // Outputs Pizza
cout << meal << "\n";  // Outputs Pizza
```

POINTERS:

Creating Pointers

We can use **"&"** operator for finding address of Variable.

Example

```
string food = "Pizza"; // A food variable of type string
cout << food;  // Outputs the value of food (Pizza)
cout << &food; // Outputs the memory address of food (0x6dfed4)
```

A pointer however, is a variable that stores the memory address as its value.

A pointer variable points to a data type (like int or string) of the same type, and is created with the * operator. The address of the variable you're working with is assigned to the pointer:

Example

```
string food = "Pizza";  // A food variable of type string

string* ptr = &food;    // A pointer variable, with the name ptr, that stores the address of food

// Output the value of food (Pizza)

cout << food << "\n";

// Output the memory address of food (0x6dfed4)

cout << &food << "\n";

// Output the memory address of food with the pointer (0x6dfed4)

cout << ptr << "\n";
```

Example explained

Create a pointer variable with the name ptr, **that points to a string variable**, by using the asterisk sign * (string* ptr). Note that the type of the **pointer has to match the type of the variable you're working with**.

Use the & operator to store the memory address of the variable called food, and assign it to the pointer.

Now, ptr holds the value of food's memory address.

Tip: There are three ways to declare pointer variables, but the first way is preferred:

```
string* mystring; // Preferred

string *mystring;

string * mystring;
```

However, you can also use the pointer to get the value of the variable, by using the * operator (the dereference operator).

Example:

```
string food = "Pizza";  // Variable declaration

string* ptr = &food;    // Pointer declaration

// Reference: Output the memory address of food with the pointer (0x6dfed4)

cout << ptr << "\n";
```

```
// Dereference: Output the value of food with the pointer (Pizza)

cout << *ptr << "\n";
```

We can also change the value using pointer

Example:

```
*ptr = “PANIPURI”;
```

Note that the * sign can be confusing here, as it does two different things in our code:

>>When used in declaration (string* ptr), it creates a pointer variable.

>>When not used in declaration, it act as a dereference operator.