CodeWithHarry

HTML CSS JS C C++ JAVA PYTHON PHP

Structures, Unions & Enums in C++ | C++ Tutorials for Beginners #14

Overview Q&A Downloads Announcements

Structures, Unions & Enums in C++ | C++ Tutorials for Beginners #14

In this tutorial, we will discuss structures, unions & enums in C++

Structures in C++

The structure is a user-defined data type that is available in C++. Structures are used to combine different types of data types, just like an array is used to combine the same type of data types. An example program for creating a structure is shown in Code Snippet 1.

```
struct employee
{
    /* data */
    int eId;
    char favChar;
    float salary;
};
```

Code Snippet 1: Creating a Structure Program

As shown in Code Snippet 1, we have created a structure with the name “employee”, in which we have declared three variables of different data types (eId, favchar, salary). As we have created a structure now we can create instances of our structure employee. An example program for creating instances of structure employees is shown in Code Snippet 2.

```
 int main() {
     struct employee harry;
     harry.eId = 1;
     harry.favChar = 'c';
     harry.salary = 120000000;
     cout<<"The value is "<<harry.eId<<endl;
     cout<<"The value is "<<harry.favChar<<endl;
     cout<<"The value is "<<harry.salary<<endl;
     return 0;
}
```

Code Snippet 2: Creating Structure instances

As shown in Code Snippet 2, 1st we have created a structure variable “harry” of type “employee”, 2nd we have assigned values to (eId, favchar, salary) fields of the structure employee and at the end we have printed the value of “salary”.

Another way to create structure variables without using the keyword “struct” and the name of the struct is shown in Code Snippet 3.

```
typedef struct employee
{
    /* data */
    int eId; //4
    char favChar; //1
    float salary; //4
} ep;
```

Code Snippet 3: Creating Structure Program 2

As shown in Code Snippet 3, we have used a keyword “**typedef**” before struct and after the closing bracket of structure, we have written “ep”. Now we can create structure variables without using the keyword “struct” and name of the struct. An example is shown in Code Snippet 4.

```
int main(){
ep harry;
    struct employee shubham;
    struct employee rohanDas;
    harry.eId = 1;
    harry.favChar = 'c';
    harry.salary = 120000000;
    cout<<"The value is "<<harry.eId<<endl;
    cout<<"The value is "<<harry.favChar<<endl;
    cout<<"The value is "<<harry.salary<<endl;
    return 0;
}
```

Code Snippet 4: Creating Structure instance 2

As shown in Code Snippet 4, we have created a structure instance “harry” by just writing “ep” before it.

Unions in C++

Unions are similar to structures but they provide better memory management then structures. Unions use shared memory so only 1 variable can be used at a time. An example program to create unions is shown in Code Snippet 5.

```
union money
{
    /* data */
    int rice; //4
    char car; //1
    float pounds; //4
};
```

Code Snippet 5: Creating Unions Program

As shown in Code Snippet 5, we have created a union with the name “money” in which we have declared three variables of different data types (rice, car, pound). The main thing to note here is that:

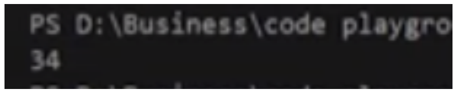
- We can only use 1 variable at a time otherwise the compiler will give us a garbage value
- The compiler chooses the data type which has maximum memory for the allocation.

An example program for creating an instance of union money is shown in Code Snippet 6.

```
int main(){
        union money m1;
        m1.rice = 34;
        cout<<m1.rice;
        return 0;
}
```

Code Snippet 6: Creating a Union Instance

As shown in Code Snippet 6, 1st we have created a union variable “m1” of type “money”, 2nd we have assigned values to (rice) fields of the union money, and in the end, we have printed the value of “rice”. The main thing to note here is that once we have assigned a value to the union field “rice”, now we cannot use other fields of the union otherwise we will get garbage value. The output for the following program is shown in figure 1.



```
PS D:\Business\code playgro
34
```

Figure 1: Creating Union Instance Output

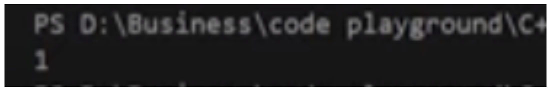
Enums in C++

Enums are user-defined types which consist of named constants. Enums are used to make the program more readable. An example program for enums is shown in Code Snippet 8.

```
int main(){
    enum Meal{ breakfast, lunch, dinner};
    Meal m1 = lunch;
    cout<<m1;
    return 0;
}
```

Code Snippet 7: Enums Program

As shown in Code Snippet 7, 1st we have created an enum “Meal” in which we have stored three named constants (breakfast, lunch, dinner). 2nd we have assigned the value of “lunch” to the variable “m1” and at the end, we have printed “m1”. The main thing to note here is that (breakfast, lunch, dinner) are constants; the value for “breakfast” is “0”, the value for “lunch” is “1” and the value for “dinner” is “2”. The output for the following program is shown in figure 2.



```
PS D:\Business\code playground\C+
1
```

Figure 2: Enums Program Output

Code as described/written in the video

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

typedef struct employee
{
    /* data */
    int eId; //4
    char favChar; //1
    float salary; //4
} ep;

union money
{
    /* data */
    int rice; //4
    char car; //1
    float pounds; //4
};


int main(){
    enum Meal{ breakfast, lunch, dinner};
    Meal m1 = lunch;
    cout<<(m1==2);
    // cout<<breakfast;
    // cout<<lunch;
    // cout<<dinner;
    // union money m1;
    // m1.rice = 34;
    // m1.car = 'c';
    // cout<<m1.car;

    // ep harry;
    // struct employee shubham;
    // struct employee rohanDas;
    // harry.eId = 1;
    // harry.favChar = 'c';
    // harry.salary = 120000000;
    // cout<<"The value is "<<harry.eId<<endl;
    // cout<<"The value is "<<harry.favChar<<endl;
    // cout<<"The value is "<<harry.salary<<endl;
    return 0;
}
```

Previous Next

CodeWithHarry