SALE!
GeeksforGeeks Courses Upto 25% Off Enroll Now!

References in C++

Difficulty Level : Medium • Last Updated : 30 Mar, 2023

When a variable is declared as a reference, it becomes an alternative name for an existing variable. A variable can be declared as a reference by putting '&' in the declaration.

Also, we can define a reference variable as a type of variable that can act as a reference to another variable. '&' is used for signifying the address of a variable or any memory. Variables associated with reference variables can be accessed either by its name or by the reference variable associated with it.

Prerequisite: Pointers in C++

Syntax:

AD

```
data_type &ref = variable;
```

Example:

C++

```
// C++ Program to demonstrate
// use of references
#include <iostream>
```

```
using namespace std;

int main()
{
    int x = 10;

    // ref is a reference to x.
    int& ref = x;

    // Value of x is now changed to 20
    ref = 20;
    cout << "x = " << x << '\n';

    // Value of x is now changed to 30
    x = 30;
    cout << "ref = " << ref << '\n';

    return 0;
}
```

Output:

```
x = 20
ref = 30
```

Applications of Reference in C++

There are multiple applications for references in C++, a few of them are mentioned below:

1. Modify the passed parameters in a function
2. Avoiding a copy of large structures
3. In For Each Loop to modify all objects
4. For Each Loop to avoid the copy of objects

1. Modify the passed parameters in a function:

If a function receives a reference to a variable, it can modify the value of the variable. For example, the following program variables are swapped using references.

Example:

C++

```
// C++ Program to demonstrate
// Passing of references as parameters
#include <iostream>
using namespace std;
```

```
// Function having parameters as
// references
void swap(int& first, int& second)
{
    int temp = first;
    first = second;
    second = temp;
}

// Driver function
int main()
{
    // Variables declared
    int a = 2, b = 3;

    // function called
    swap(a, b);

    // changes can be seen
    // printing both variables
    cout << a << " " << b;
    return 0;
}
```

Output

```
3 2
```

2. Avoiding a copy of large structures:

Imagine a function that has to receive a large object. If we pass it without reference, a new copy of it is created which causes a waste of CPU time and memory. We can use references to avoid this.

Example:

```
struct Student {
    string name;
    string address;
    int rollNo;
}

// If we remove & in below function, a new
// copy of the student object is created.
// We use const to avoid accidental updates
// in the function as the purpose of the function
// is to print s only.
```

```
void print(const Student &s)
{
    cout << s.name << "  " << s.address << "  " << s.rollNo
         << '\n';
}
```

3. In For Each Loop to modify all objects:

We can use references for each loop to modify all elements.

Example:

C++

```
// C++ Program for changing the
// values of elements while traversing
// using references
#include <iostream>
#include <vector>

using namespace std;

// Driver code
int main()
{
    vector<int> vect{ 10, 20, 30, 40 };

    // We can modify elements if we
    // use reference
    for (int& x : vect) {
        x = x + 5;
    }

    // Printing elements
    for (int x : vect) {
        cout << x << " ";
    }
    cout << '\n';

    return 0;
}
```

Output

```
15 25 35 45
```

4. For Each Loop to avoid the copy of objects:

We can use references in each loop to avoid a copy of individual objects when objects are large.

Example:

C++

```
// C++ Program to use references
// For Each Loop to avoid the
// copy of objects
#include <iostream>
#include <vector>

using namespace std;

// Driver code
int main()
{
    // Declaring vector
    vector<string> vect{ "geeksforgeeks practice",
                         "geeksforgeeks write",
                         "geeksforgeeks ide" };

    // We avoid copy of the whole string
    // object by using reference.
    for (const auto& x : vect) {
        cout << x << '\n';
    }

    return 0;
}
```

Output

```
geeksforgeeks practice
geeksforgeeks write
geeksforgeeks ide
```

References vs Pointers

Both references and pointers can be used to change the local variables of one function inside another function. Both of them can also be used to save copying of big objects when passed as arguments to functions or returned from functions, to get efficiency gain. Despite the above similarities, there are the following differences between references and pointers.

1. A pointer can be declared as void but a reference can never be void For example

```
int a = 10;
void* aa = &a; // it is valid
void& ar = a;  // it is not valid
```

2. The pointer variable has n-levels/multiple levels of indirection i.e. single-pointer, double-pointer, triple-pointer. Whereas, the reference variable has only one/single level of indirection. The following code reveals the mentioned points:

3. Reference variables cannot be updated.

4. Reference variable is an internal pointer.

5. Declaration of a Reference variable is preceded with the '&' symbol (but do not read it as "address of").

Example:

C++

```
// C++ Program to demonstrate
// references and pointers
#include <iostream>
using namespace std;

// Driver Code
int main()
{
    // simple or ordinary variable.
    int i = 10;

    // single pointer
    int* p = &i;

    // double pointer
    int** pt = &p;

    // triple pointer
    int*** ptr = &pt;

    // All the above pointers differ in the value they store
    // or point to.
    cout << "i = " << i << "\t"
         << "p = " << p << "\t"
         << "pt = " << pt << "\t"
         << "ptr = " << ptr << '\n';

    // simple or ordinary variable
    int a = 5;
    int& S = a;
    int& S0 = S;
    int& S1 = S0;
```

```
    // All the references do not differ in their
    // values as they all refer to the same variable.
    cout << "a = " << a << "\t"
         << "S = " << S << "\t"
         << "S0 = " << S0 << "\t"
         << "S1 = " << S1 << '\n';

    return 0;
}
```

Output

```
i = 10    p = 0x7ffecfe7c07c    pt = 0x7ffecfe7c080    ptr = 0x7ffecfe7c088
a = 5    S = 5    S0 = 5    S1 = 5
```

Limitations of References

1. Once a reference is created, it cannot be later made to reference another object; it cannot be reset. This is often done with pointers.
2. References cannot be NULL. Pointers are often made NULL to indicate that they are not pointing to any valid thing.
3. A reference must be initialized when declared. There is no such restriction with pointers.

Due to the above limitations, references in C++ cannot be used for implementing data structures like Linked List, Tree, etc. In Java, references don't have the above restrictions and can be used to implement all data structures. References being more powerful in Java is the main reason Java doesn't need pointers.

Advantages of using References

1. ***Safer****:* Since references must be initialized, wild references like wild pointers are unlikely to exist. It is still possible to have references that don't refer to a valid location (See questions 5 and 6 in the below exercise)
2. ***Easier to use:*** References don't need a dereferencing operator to access the value. They can be used like normal variables. The '&' operator is needed only at the time of declaration. Also, members of an object reference can be accessed with the dot operator ('.'), unlike pointers where the arrow operator (->) is needed to access members.

Together with the above reasons, there are a few places like the copy constructor argument where a pointer cannot be used. Reference must be used to pass the argument in the copy constructor. Similarly, references must be used for overloading some operators like ++.

Exercise with Answers

Question 1 :

C++

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

int& fun()
{
    static int x = 10;
    return x;
}

int main()
{
    fun() = 30;
    cout << fun();
    return 0;
}
```

Output

```
30
```

Question 2

C++

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

int fun(int& x) { return x; }

int main()
{
    cout << fun(10);
    return 0;
}
```

Output:

```
./3337ee98-ae6e-4792-8128-7c879288221f.cpp: In function 'int main()':
./3337ee98-ae6e-4792-8128-7c879288221f.cpp:8:19: error: invalid
initialization of non-const reference of type 'int&' from an rvalue of type
'int'
```

```
      cout << fun(10);
                    ^
./3337ee98-ae6e-4792-8128-7c879288221f.cpp:4:5: note: in passing argument 1
of 'int fun(int&)'
 int fun(int& x) { return x; }
```

Question 3

C++

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

void swap(char*& str1, char*& str2)
{
    char* temp = str1;
    str1 = str2;
    str2 = temp;
}

int main()
{
    char* str1 = "GEEKS";
    char* str2 = "FOR GEEKS";
    swap(str1, str2);
    cout << "str1 is " << str1 << '\n';
    cout << "str2 is " << str2 << '\n';
    return 0;
}
```

Output

```
str1 is FOR GEEKS
str2 is GEEKS
```

Question 4

C++

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

int main()
{
    int x = 10;
    int* ptr = &x;
    int&* ptr1 = ptr;
}
```

Output:

```
./18074365-ebdc-4b13-81f2-cfc42bb4b035.cpp: In function 'int main()':
./18074365-ebdc-4b13-81f2-cfc42bb4b035.cpp:8:11: error: cannot declare
pointer to 'int&'
     int&* ptr1 = ptr;
```

Question 5

C++

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

int main()
{
    int* ptr = NULL;
    int& ref = *ptr;
    cout << ref << '\n';
}
```

Output:

```
timeout: the monitored command dumped core
/bin/bash: line 1:    34 Segmentation fault      timeout 15s ./372da97e-
346c-4594-990f-14edda1f5021 < 372da97e-346c-4594-990f-14edda1f5021.in
```

Question 6

C++

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

int& fun()
{
    int x = 10;
    return x;
}

int main()
{
    fun() = 30;
    cout << fun();
    return 0;
}
```

Output

```
0
```

Related Articles

- Pointers vs References in C++
- When do we pass arguments by reference or pointer?
- Can references refer to an invalid location in C++?
- Passing by pointer Vs Passing by Reference in C++

Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above

291

Related Articles

1. lvalues references and rvalues references in C++ with Examples
2. Can References Refer to Invalid Location in C++?
3. Default Assignment Operator and References in C++
4. Pointers and References in C++
5. Overloads of the Different References in C++
6. Pointers vs References in C++
7. C++ Error - Does not name a type
8. Execution Policy of STL Algorithms in Modern C++
9. C++ Program To Print Pyramid Patterns
10. Jagged Arrays in C++

Previous Next