Tutorial Link https://codequotient.com/tutorials/C++ : Reference Variables/5b419e91f837921993917fb1

TUTORIAL

C++ : Reference Variables

Chapter

Reference Variables

C++ added the so-called reference variables (or references in short). A reference is an alias or an alternate name to an existing variable. For example, suppose you make Aman a reference (alias) to Ajay, you can refer to the person as either Aman or Ajay. The main use of references is acting as function formal parameters to support pass-by-reference. In a reference variable is passed into a function, the function works on the original copy (instead of a clone copy in pass-by-value). Changes inside the function are reflected outside the function. A reference is similar to a pointer. In many cases, a reference can be used as an alternative to a pointer, in particular, for the function parameter.

References (or Aliases) (&)

Recall that C/C++ uses & to denote the address-of operator in an expression. C++ assigns an additional meaning to & in a declaration to declare a reference variable. The meaning of a symbol & is different in an expression and in a declaration. When it is used in an expression, & denotes the address-of operator, which returns the address of a variable, e.g., if the number is an int variable, &number returns the address of the variable number (this has been described in the above section). However, when & is used in a declaration (including function formal parameters), it is part of the type identifier and is used to declare a reference variable (or reference or alias or alternate name). It is used to provide another name, or another reference, or an alias to an existing variable. The syntax is as follow:

```
type &newName = existingName;
// or
type& newName = existingName;
// or
type & newName = existingName; // I shall adopt this convention
```

It shall be read as "newName is a reference to exisitngName", or "newNew is an alias of existingName". You can now refer to the variable as newName or existingName. For example,

C++

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

int main() {
  int number = 88;          // Declare an int variable called number
  int & refNumber = number; // Declare a reference (alias) to the variable number
  // Both refNumber and number refer to the same value

  cout << number << endl;    // Print value of variable number (88)
  cout << refNumber << endl; // Print value of reference (88)

  refNumber = 99;            // Re-assign a new value to refNumber
  cout << refNumber << endl;
  cout << number << endl;    // Value of number also changes (99)

  number = 55;               // Re-assign a new value to number
  cout << number << endl;
  cout << refNumber << endl; // Value of refNumber also changes (55)
  return 0;
}
```

A reference works as a pointer. A reference is declared as an alias of a variable. It stores the address of the variable.

References vs. Pointers

Pointers and references are equivalent, except:

- A reference is a named constant for an address. You need to initialize the reference during the declaration.

```
                            int & iRef;  // Error: 'iRef' declared as reference but not initialized
```

Once a reference is established to a variable, you cannot change the reference to reference another variable.

- To get the value pointed to by a pointer, you need to use the dereferencing operator * (e.g., if pNumber is a int pointer, *pNumber returns the value pointed to by pNumber. It is called dereferencing or indirection). To assign an address of a variable into a pointer, you need to use the address-of operator & (e.g., pNumber = &number).

On the other hand, referencing and dereferencing are done on the references implicitly. For example, if refNumber is a reference (alias) to another int variable, refNumber returns the value of the variable. No explicit dereferencing operator * should be used. Furthermore, to assign an address of a variable to a reference variable, no address-of operator & is needed. For example,

C++

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

int main() {
  int number1 = 88, number2 = 22;

  // Create a pointer pointing to number1
  int * pNumber1 = &number1;  // Explicit referencing
  *pNumber1 = 99;             // Explicit dereferencing
  cout << *pNumber1 << endl;  // 99
  cout << &number1 << endl;   // 0x22ff18
  cout << pNumber1 << endl;   // 0x22ff18 (content of the pointer variable - same as above)
  cout << &pNumber1 << endl;  // 0x22ff10 (address of the pointer variable)
  pNumber1 = &number2;        // Pointer can be reassigned to store another address

  // Create a reference (alias) to number1
  int & refNumber1 = number1;  // Implicit referencing (NOT &number1)
  refNumber1 = 11;             // Implicit dereferencing (NOT *refNumber1)
  cout << refNumber1 << endl;  // 11
  cout << &number1 << endl;    // 0x22ff18
  cout << &refNumber1 << endl; // 0x22ff18
  //refNumber1 = &number2;     // Error! Reference cannot be re-assigned
  // error: invalid conversion from 'int*' to 'int'
  refNumber1 = number2;        // refNumber1 is still an alias to number1.
  // Assign value of number2 (22) to refNumber1 (and number1).
  number2++;
  cout << refNumber1 << endl;  // 22
  cout << number1 << endl;     // 22
  cout << number2 << endl;     // 23
  return 0;
```

```
}
```

A reference variable provides a new name to an existing variable. It is *dereferenced implicitly* and does not need the dereferencing operator * to retrieve the value referenced. On the other hand, a pointer variable stores an address. You can change the address value stored in a pointer. To retrieve the value pointed to by a pointer, you need to use the indirection operator *, which is known as *explicit dereferencing*. Reference can be treated as a `const` pointer. It has to be initialized during declaration, and its content cannot be changed.

Reference is closely related to the pointer. In many cases, it can be used as an alternative to the pointer. A reference allows you to manipulate an object using a pointer, but without the pointer syntax of referencing and dereferencing.

The above example illustrates how reference works but does not show its typical usage, which is used as the function formal parameter for pass-by-reference.

Passing the Function's Return Value

You can also pass the return value as a reference or pointer. For example,

C++

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

int & squareRef(int &);
int * squarePtr(int *);

int main() {
  int number1 = 8;
  cout <<  "In main() &number1: " << &number1 << endl;  // 0x22ff14
  int & result = squareRef(number1);
  cout <<  "In main() &result: " << &result << endl;  // 0x22ff14
  cout << result << endl;   // 64
  cout << number1 << endl;  // 64

  int number2 = 9;
  cout <<  "In main() &number2: " << &number2 << endl;  // 0x22ff10
  int * pResult = squarePtr(&number2);
  cout <<  "In main() pResult: " << pResult << endl;  // 0x22ff10
  cout << *pResult << endl;   // 81
  cout << number2 << endl;    // 81
  return 0;
}

int & squareRef(int & rNumber) {
  cout <<  "In squareRef(): " << &rNumber << endl;  // 0x22ff14
```

```
  rNumber *= rNumber;
  return rNumber;
}

int * squarePtr(int * pNumber) {
  cout <<  "In squarePtr(): " << pNumber << endl;  // 0x22ff10
  *pNumber *= *pNumber;
  return pNumber;
}
```

You should not pass Function's local variable as return value by reference, for example,

C++

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

int * squarePtr(int);
int & squareRef(int);

int main() {
  int number = 8;
  cout << number << endl;  // 8
  cout << *squarePtr(number) << endl;  // ??
  cout << squareRef(number) << endl;   // ??
  return 0;
}

int * squarePtr(int number) {
  int localResult = number * number;
  return &localResult;
  // warning: address of local variable 'localResult' returned
}

int & squareRef(int number) {
  int localResult = number * number;
  return localResult;
  // warning: reference of local variable 'localResult' returned
}
```

This program has a serious logical error, as the local variable of function is passed back as a return value by reference. The local variable has local scope within the function, and its value is destroyed after the function exits. The GCC compiler is kind enough to issue a warning (but no error). It is safe to return a reference that is passed into the function as an argument. See earlier examples.

Passing Dynamically Allocated Memory as Return Value by Reference

Instead, you need to dynamically allocate a variable for the return value, and return it by reference.

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

int * squarePtr(int);
int & squareRef(int);

int main() {
  int number = 8;
  cout << number << endl;  // 8
  cout << *squarePtr(number) << endl;  // 64
  cout << squareRef(number) << endl;   // 64
  return 0;
}

int * squarePtr(int number) {
  int * dynamicAllocatedResult = new int(number * number);
  return dynamicAllocatedResult;
}

int & squareRef(int number) {
  int * dynamicAllocatedResult = new int(number * number);
  return *dynamicAllocatedResult;
}
```