Const & R Value References

In C++, you can use the **const** keyword to declare a reference that cannot be modified, and you can use the **&&** operator to declare an rvalue reference, which is a reference to a temporary object that will be destroyed after the reference goes out of scope.

Here is an example of how to use const references in C++:

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

// function that takes a const reference to an integer as inp
void printNumber(const int &x) {
  cout << x << endl;
}

int main() {
  int a = 3;
  printNumber(a);
  return 0;
}
```

Output

```
3
```

In this example, the function **printNumber()** takes a **const** reference to an integer as input. This means that the value of the integer cannot be modified through the reference. The function is called from the **main()** function with the variable **a**, and the value of **a** is printed to the screen.

Here is an example of how to use rvalue references in C++:

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

// function that takes an rvalue reference to an integer as i
void printNumber(int &&x) {
  cout << x << endl;
}

int main() {
  printNumber(3);
  return 0;
}
```

Output

```
3
```

In this example, the function **printNumber()** takes an rvalue reference to an integer as input. This means that the function receives a reference to a temporary object that will be destroyed after the reference goes out of scope. The function is called from the **main()** function with the value **3**, and the value is printed to the screen.

const references and rvalue references can be useful in certain situations, such as when you want to prevent a function from modifying an object or when you want to pass a temporary object to a function.

One of the main limitations of **const** references is that they cannot be used to modify the original object. This means that if you want to modify an object through a **const** reference, you will need to use another technique, such as pass-by-reference or pass-by-pointer.

Here is an example of how **const** references cannot be used to modify an object:

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

// function that takes a const reference to an integer as inp
void increment(const int &x) {
  x++; // error: x is a const reference and cannot be modifie
}

int main() {
  int a = 3;
  increment(a);
  return 0;
}
```

In this example, the function **increment()** takes a **const** reference to an integer as input and tries to increment the value of the integer. However, this is not allowed, as **x** is a **const** reference and cannot be modified. If you try to compile and run this program, you will get a compiler error.

Another limitation of **const** references is that they cannot be bound to modifiable lvalues. This means that you cannot use a **const** reference to refer to a variable that you can modify, such as a regular variable or a non-**const** reference.