References

A reference is an alias for an existing object.  That is, it is an alternative name for the entity defined elsewhere.  Each object defined in source code can have a reference associated with it.  C++17 admits two reference declarations:

* an lvalue reference - denoted by &
* an rvalue reference - denoted by &&

The declaration of an lvalue reference identifies an accessible region of memory.  The declaration of an rvalue reference identifies:

* an object near the end of its lifetime
* a temporary object or subobject
* a value not associated with an object

The C++ language does not support references to references, arrays of references, or pointers to references.

lvalue References

An lvalue reference requires an initializer unless it

* has external linkage
* is a class member within a class definition
* is a function parameter or a function return type

|  |  |
| --- | --- |
| // lvalue References  // lvalue.cpp  #include "A.h"  #include "B.h"  B e(8), f(6);  B& b = e; // LVALUE REQUIRES INITIALIZER  int main() {  A a(f);  foo(a);  } | 6  8 |
| // Module A Header  // A.h  #pragma once  class B;  class A {  B& d; // no initializer required: class member declaration  public:  A(B& bb); // no initializer required: function parameter declaration  };  A& foo(A&); |
| // Module A Implementation  // A.cpp  #include <iostream>  #include "A.h"  #include "B.h"  extern B& b; // no initializer required: external linkage  A::A(B& bb) : d(bb) { // initialization is in constructor definition  d.display();  }  A& foo(A& a) { // no initializers required:  // parameter and return type declarations  A& aa = a; // LVALUE REQUIRES AN INITIALIZER  b.display();  return aa;  } |
| // Module B Header  // B.h  #pragma once  class B {  int b;  public:  B(int n);  void display() const;  }; |
| // Module B Implementation  // B.cpp  #include <iostream>  #include "B.h"  B::B(int n) : b(n) {}  void B::display() const { std::cout << b << std::endl; } |

rvalue Reference

An rvalue reference declaration identifies an object that is less permanent, possibly temporary.  For example,

|  |  |
| --- | --- |
| // rvalue Reference  // rvalue.cpp  #include <iostream>  class A {  int a;  public:  A(int aa) : a(aa) {}  void display(const char\* str) const {  std::cout << str << ' ' << a << std::endl;  }  };  void print(const A& a) { a.display("lvalue "); }  void print(A&& a) { a.display("rvalue "); }  int main() {  A a(10);  print(a);  print(A(20));  } | lvalue 10  rvalue 20 |

Practical uses for rvalue references include the move-constructors and move-assignment operators described in the chapter entitled [Move Operators](https://mwatler.github.io/sep200/week3/MoveOperators.docx).

Standard Library

The standard library provides two functions for specifying the type of reference when that is important:

* std::ref() - returns an lvalue reference to its argument (important of functions in the standard library)
* std::move() - returns an rvalue reference to its argument

Their prototypes are declared in <utility>

|  |  |
| --- | --- |
| // rvalue Reference using std::move()  // std\_move.cpp  #include <iostream>  #include <utility>  class A {  int a;  public:  A(int aa) : a(aa) {}  void display(const char\* str) const {  std::cout << str << ' ' << a << std::endl;  }  };  void print(const A& a) { a.display("lvalue "); }  void print(A&& a) { a.display("rvalue "); }  int main() {  A a(10), b(20);  print(a);  print(std::move(b));  } | lvalue 10  rvalue 20 |

Practical uses of std::ref are discussed in the chapter entitled [Algorithms](https://mwatler.github.io/sep200/Week11/Algorithms.docx).