# Move Semantics and PerfectForwarding

Move semantics

Rvalue,lvalue,xvalue,pvalue

#include <vector>  
#include <iostream>  
  
  
using namespace std;  
  
struct Foo{  
  
  
 Foo()= default;  
 Foo(int i):i{new int(i)}{}  
 Foo(Foo const& other){  
 if(other.i!=nullptr){  
 this->i=new int(\*(other.i));  
 }  
 }  
 int\* i=nullptr;  
 ~Foo(){  
 if(i!=nullptr){  
 delete i;  
 i=nullptr;  
 }  
 }  
  
 Foo&operator=(const Foo& rhs){// lvalue reference  
 if(rhs.i!= nullptr){  
 this->i=new int(\*(rhs.i));  
 }  
 return \*this;  
  
 }  
  
 Foo&operator=(Foo&& rhs){ // rvalue reference  
 if(rhs.i!= nullptr){  
 this->i=rhs.i;  
 rhs.i=nullptr;  
 }  
 return \*this;  
  
 }  
  
};  
  
void FuncByValue(Foo other); //value semantics->copy  
void FuncByRef(Foo& other);// reference semantics  
  
  
// xvalue -> eXpiring value  
// pvalue -> pure value  
int ReturnPRvalue(){  
 return 42;  
}  
  
  
  
  
  
int main(){  
  
 42;// complies fine  
 int i=42;// prvalue  
  
 i= ReturnPRvalue();// prvalue  
 Foo f1;  
 {  
 Foo f2;  
 f2.i=new int (42); // heap allocation -> expensive  
 f1=move(f2);// xvalue  
 f1=Foo(42);// prvalue  
 }  
  
  
 return 0;  
}

Special member function: complier sẽ tự động generate ra 6 function này.

Nếu ta tự định nghĩa một vài hàm trong số này thì complier sẽ không tự động tạo ra nữa.

#include <vector>  
#include <iostream>  
  
  
using namespace std;  
  
struct Foo{  
  
 // giving v's ownership for Foo constructor,then foo constructor give ownership to vector constructor of member V  
 Foo(vector<int>&& v):v{move(v)}{  
 // neu khong dung v{move(v)} thi se khong thao tac duoc vector v:  
 v.front();  
 //v.erase(..)  
  
 }  
 vector<int > v;  
 /\* a lot of expensive other data \*/  
 // bar b;  
  
  
};  
void FillVector(vector<int>& v){ }  
  
  
  
  
  
int main(){  
  
 vector<int>v;  
 FillVector(v);  
  
 if(!v.empty()){  
 // do nothing  
 Foo f1{move(v)};  
 //valid but unspecified state  
 // v.front() // not safe!  
  
 v.clear(); // this is fine.... because it do not depend on state  
  
 }else{  
  
 }  
  
  
  
}

Perfect forwarding

#include <vector>  
#include <iostream>  
  
using namespace std;  
struct Foo{  
 // alot of member variables  
 Foo()=default;  
 Foo(int i):m\_i{i}{}  
 Foo(int i,bool b):m\_i{i},m\_b{b}{}  
 Foo(int i,bool b,float f):m\_i{i},m\_b{b},m\_f{f}{}  
 int m\_i;  
 bool m\_b;  
 float m\_f;  
};  
struct Bar{  
 template <typename ... Args>  
 void AddFoo(Args ... args){  
 v.emplace\_back(forward<Args>(args)...);  
 }  
private:  
 vector<Foo> v;  
};  
  
int main(){  
  
 Bar b;  
 Foo f(1,true,2.f);  
 b.AddFoo(1,true,2.f);  
 return 0;  
}