**STAGES OF COMPILATION**

+ Preprocessor (inserts the contents of all **#include** header files), (substitutes all**#define** macros)

+ Compiler - compiles each translation unit separately into binary version

+ Linker - assembles binary units along with the system binaries to one complete executable binary

**Function Signature**

A function's *signature* identifies an overloaded function uniquely.  Its signature consists of

+ the function identifier

+ the parameter types (ignoring **const** qualifiers or address of operators as described in references below)

+ the order of the parameter types

The return type and the parameter identifiers are not part of a function's signature. C++ compilers preserve identifier uniqueness by renaming each overloaded function using a combination of its identifier, its parameter types and the order of its parameter types.  We refer to this renaming as name *mangling*.

**Derived class**

* a derived class with a resource requires explicit definitions of its special member functions - constructors, copy assignment operator and destructor
* an explicitly defined derived class copy constructor without a call to the base class' copy constructor calls the base class' no-argument constructor
* the derived class' copy constructor executes the logic in the base class' copy constructor first
* an explicitly defined derived class copy assignment operator does NOT automatically call the base class assignment operator.
* the derived class assignment operator executes the base class assignment operator entirely within the scope of the copy derived class assignment operator
* the destructor of a derived class automatically calls the destructor of the base class

**Student::operator bool() const { return no != 0; }**

#include <iostream>

using namaspace std;

//can be as many as we like <T1, T2, T3, etc>

template<typename T, typename RT>

RT& add(T a, T b) {

//\*\* if RT is a custom type, we must have approriate constructor to support the intinialization

RT result(0); //equivalent to RT result = 0;

//\*\* need an overload of approriate

//in this case we need 2 operator, copy assignmnent for RT and operator+ for T type;

RT = Foo(a + b);

//\*\* We need a copy constructor for RT and a destructor for RT

return result;

}

class Foo {

long val;

long val2;

public:

Foo() { cout << "Foo::Foo()" << endl; val = 10; }

Foo(int) { cout << "Foo::Foo(int)" << endl; val = 20; }

Foo& operator=(const Foo&); //copy assignment

Foo& operator=(short a) { //val = a;

val2 = a;

}

};

int main() {

cout << add<short, float, double>(10, 22.32) << endl;

cout << add<short, float, int>(10.10, 22.32) << endl;

cout << add <short, Foo>(10, 22);

return 0;

}

\*\*Virtual function needs a body even though there are no derived class, only pure virtual fns don’t require a body. Implementation != abstract class

#include <iostream>

using namespace std;

class A {

int m\_num;

public:

A() : m\_num(0) { cout << "A() " << m\_num << endl; }

A(int num) : m\_num(num) {

cout << "A(int) " << m\_num << endl;

}

A(const A& rhs) {

if (&rhs != this) \*this = rhs;

cout << "A copy constructor " << m\_num << endl;

}

~A() { cout << "~A() " << m\_num << endl; }

A& operator= (const A& rhs) {

if (&rhs != this) this->m\_num = rhs.m\_num;

cout << "A copy assignment " << m\_num << endl;

return \*this;

}

int getNum() { return m\_num; }

A& operator+= (const A& rhs) {

this->m\_num += rhs.m\_num;

cout << "A operator+= " << m\_num << endl;

return \*this;

}

A& operator--(int) {

m\_num--;

cout << "A operator-- " << m\_num << endl;

return \*this;

}

};

class B : public A {

public:

B() { cout << "B() " << getNum() << endl; }

B(int num) : A(num) { cout << "B(int) " << getNum() << endl; }

~B() { cout << "~B() " << getNum() << endl; }

B(const B& rhs) {

if (&rhs != this) \*this = rhs;

cout << "B copy constructor " << getNum() << endl;

}

B& operator=(const B& rhs) {

if (&rhs != this)

A::operator= (rhs); //call base class copy assignment

cout << "B copy assignment " << getNum() << endl;

return \*this;

}

B& operator+= (const B& rhs) {

A::operator+= (rhs);

cout << "B operator+= " << getNum() << endl;

return \*this;

}

//The int type in the header distinguishes the post-fix

//operators from their pre-fix counterparts

//which has prototype Type& operator++()

B operator--(int) {

A::operator--(100); //\*\*\*interesting

cout << "B operator-- " << getNum() << endl;

return \*this;

}

};

int main() {

B obj, obj1(5), obj2(3);

//Output:

//A() 0

//B() 0;

//A(int) 5

//B(int) 5

//A(int) 3

//B(int 3

obj = obj1;

//A copy assignment 5

//B copy assignment 5

obj1 += obj2;

//A operator+= 8

//B operator += 8

obj2--;

//A operator-- 2;

//B operator-- 2;

//A() 0 //new object A to build B at B copy constructor

//A copy assignment 2 //B operator= call A's

//B copy assignment 2 //B operator= fisnished

//B copy constructor 2 //new B constructed

//~B() 2 //return \*this go out of scope

//~A() 2

return 0;

//~B() 2

//~A() 2

//~B() 3

//~A() 3

//~B() 8

//~A() 8

//~B() 5

//~A() 5

}