Item4: make sure that objects are initialized before they are used

Reading uninitialized values yields undefined behavior.

Non-member objects of built-in types: initialized manually, ex.

int x=0;

const char\* text=” A C-style string”; //manual initialization of a pointer

double d; // “initialization” by reading from an input stream

cin >> d;

make sure that all constructors initialize everything in the object.

Example:

class PhoneNumber { ... };

class ABEntry { // ABEntry = “Address Book Entry”

public:

ABEntry(const std::string& name, const std::string& address,const std::list<PhoneNumber>& phones);

private:

std::string theName;

std::string theAddress;

std::list<PhoneNumber> thePhones;

int numTimesConsulted;

};

ABEntry::ABEntry(const std::string& name, const std::string& address, const std::list<PhoneNumber>& phones){

theName = name; // these are all assignments,

theAddress = address; // not initializations

thePhones = phones;

numTimesConsulted = 0;

}

Default constructors of theName, theAddress, and thePhones were automatically called prior to entering the body of ABEntry constructor except for numTimesConsulted(built-in type)( no guarantee it was initialized prior to assignment)

🡺call default constructors and then call copy assignment operators

🡺initialization list solves the problem

ABEntry::ABEntry(const std::string& name, const std::string& address,

const std::list<PhoneNumber>& phones)

: theName(name), // theName is copy\_constructed from name

theAddress(address), // theAddress is copy\_constructed from address

thePhones(phones), // thePhones is copy\_constructed from phones

numTimesConsulted(0) // no difference in cost between initialization and assignment //for objects of built-in type

{} // the ctor body is now empty

Initialization list for default-contruct

ABEntry::ABEntry()

: theName(), // call theName’s default ctor;

theAddress(), // do the same for theAddress;

thePhones(), // and for thePhones;

numTimesConsulted(0) // but explicitly initialize numTimesConsulted to zero

{}

Initialization order in C++: base class🡺derived class

order within class: data members are initialized in the order they are declared

static object: exist from it is constructed to the end of the program

{local static object: static object inside functions

Non-local static object: others}

Translation unit: the source code leads to a single object file. Basically, a single source file plus #include files

Problem Example:

class FileSystem { // from your library’s header file

public:

std::size\_t numDisks() const; // one of many member functions

};

extern FileSystem tfs; // declare object for clients to use (“tfs” = “the file system” ); //definition is in some .cpp file in your library

class Directory { // created by library client

public:

Directory( params );

};

Directory::Directory( params ){

std::size\_t disks = tfs.numDisks(); // use the tfs object

}

Directory tempDir( params ); // directory for temporary files

🡺tfs should be initialized before tempDir, but tfs and tempDir were created by different people in different source files🡺they are non-local static objects defined in different translation units🡺move each non-local static object into its own function where it is declared static(C++ guarantee that local static objects are initialized when the object’s definition is first encountered during a call to that function)🡺 Example of change:

class FileSystem { ... }; // as before

FileSystem& tfs() // this replaces the tfs object; it could be

{ // static in the FileSystem class

static FileSystem fs; // define and initialize a local static object

return fs; // return a reference to it

}

class Directory { ... }; // as before

Directory::Directory( params ) // as before, except references to tfs are

{ // now to tfs()

std::size\_t disks = tfs().numDisks();

}

Directory& tempDir() // this replaces the tempDir object;

{

static Directory td( params ); // define/initialize local static object

return td; // return reference to it

}

Conclusion1: manually initialize object of built-in type

Conclusion2: In constructor, prefer use of member initialization list to assignment.

Conclusion3: Avoid initialization problems across translation units by replacing non-local static objects with local static objects