‘STATIC’ IN C IS A STORAGE CLASS

We can create variables with different settings

Static storage class

Where the variable would be stored	Memory
default value for the variable	0
scope of the variable	Local to the block in which it is defined
the life of the variable	Value of the variable persists between different function calls

We can create variables with different settings

Static storage class

Where the variable would be stored	Memory
default value for the variable	0
scope of the variable	Local to the block in which it is defined
the life of the variable	Value of the variable persists between different function calls

What about life of the variable?

RECAP: STATIC IN C

Static storage class

```
 main( )
 {
  power( ) ;
  power( ) ;
  power( ) ;
 }
power( )
 {
 int i = 2 ;
 printf ( "%d\n", i ) ;
 i = i *2 ;
}
```

Output:

```
2
2
2
```

```
main( )
 {
  power( ) ;
  power( ) ;
  power( ) ;
 }
power( )
 {
 static int i = 2 ;
  printf ( "%d\n", i ) ;
  i = i *2 ;
  }
```

Output:

```
2
4
8
```

RECAP: STATIC IN C

We can create variables with different settings

Static storage class

Where the variable would be stored	Memory
default value for the variable	0
scope of the variable	Local to the block in which it is defined
the life of the variable	Value of the variable persists between different function calls

What about life of the variable?

Static variables don't disappear when the function is no longer active. Their values persist.

STATIC IN C++

WE HAVE DISCUSSED HOW CLASSES ARE USER-DEFINED TYPES THAT INCLUDE BOTH DATA AND FUNCTIONS

THE DATA ITEMS ARE CALLED MEMBER VARIABLES

THE FUNCTIONS ARE CALLED MEMBER FUNCTIONS, OR METHODS

RECAP: OBJECTS HAVE THEIR OWN COPIES

STATIC IN C++

WE HAVE DISCUSSED HOW CLASSES ARE USER-DEFINED TYPES THAT INCLUDE BOTH DATA AND FUNCTIONS

THE DATA ITEMS ARE CALLED MEMBER VARIABLES

THE FUNCTIONS ARE CALLED MEMBER FUNCTIONS, OR METHODS

VARIABLES OF A CLASS ARE CALLED OBJECTS

RECAP: OBJECTS HAVE THEIR OWN COPIES

VARIABLES OF A CLASS ARE CALLED OBJECTS

WE HAVE DISCUSSED HOW CLASSES ARE USER-DEFINED TYPES THAT INCLUDE BOTH DATA AND FUNCTIONS

THE DATA ITEMS ARE CALLED MEMBER VARIABLES

THE FUNCTIONS ARE CALLED MEMBER FUNCTIONS, OR METHODS

EACH OBJECT HAS ITS OWN COPY OF THE MEMBER VARIABLES, AND THE MEMBER FUNCTIONS ACT ON THOSE MEMBER VARIABLES

RECAP: OBJECTS HAVE THEIR OWN COPIES

EACH OBJECT HAS ITS OWN COPY OF THE MEMBER VARIABLES, AND THE MEMBER FUNCTIONS ACT ON THOSE MEMBER VARIABLES

ERR..ACTUALLY, WHEN WE SAID THIS

EACH OBJECT HAS ITS OWN COPY OF THE MEMBER VARIABLES, AND THE MEMBER FUNCTIONS ACT ON THOSE MEMBER VARIABLES

WE..LIED A LITTLE BIT

ERR..ACTUALLY, WHEN WE SAID THIS

EACH OBJECT HAS ITS OWN COPY OF THE MEMBER VARIABLES, AND THE MEMBER FUNCTIONS ACT ON THOSE MEMBER VARIABLES

WE..LIED A LITTLE BIT

MEMBER VARIABLES OR MEMBER FUNCTIONS MARKED `static` ARE SHARED BY ALL OBJECTS OF A CLASS

MEMBER VARIABLES OR MEMBER FUNCTIONS MARKED `static` ARE SHARED BY ALL OBJECTS OF A CLASS

WEIRD C++ RULE:

`static` MEMBER VARIABLES MUST BE DEFINED OUTSIDE THE CLASS BODY, (EVEN IF `const`)

EXAMPLE 38

ADD A STATIC MEMBER VARIABLE TO A CLASS, AND USE IT OUTSIDE THE CLASS

EXAMPLE 38 ADD A STATIC MEMBER VARIABLE TO A CLASS, AND USE IT OUTSIDE THE CLASS

THIS INVOLVES 4 STEPS

1. DECLARE THE VARIABLE INSIDE THE CLASS

2. DEFINE THE VARIABLE OUTSIDE THE CLASS

3. USE THE VARIABLE INSIDE THE CLASS

4. USE THE VARIABLE OUTSIDE THE CLASS

EXAMPLE 38 ADD A STATIC MEMBER VARIABLE TO A CLASS, AND USE IT OUTSIDE THE CLASS

THIS INVOLVES 4 STEPS

1. DECLARE THE VARIABLE INSIDE THE CLASS

2. DEFINE THE VARIABLE OUTSIDE THE CLASS

3. USE THE VARIABLE INSIDE THE CLASS

4. USE THE VARIABLE OUTSIDE THE CLASS

EXAMPLE 38 ADD A STATIC MEMBER VARIABLE TO A CLASS, AND USE IT OUTSIDE THE CLASS

THIS INVOLVES 4 STEPS

1. DECLARE THE VARIABLE INSIDE THE CLASS
2. DEFINE THE VARIABLE OUTSIDE THE CLASS
3. USE THE VARIABLE INSIDE THE CLASS
4. USE THE VARIABLE OUTSIDE THE CLASS

1. DECLARE THE VARIABLE INSIDE THE CLASS

```
class ComplexNumber
{
private:
  float realPart;
  float complexPart;
public:
  static int numObjectsCreated;
```

THIS IS VERY SIMPLE, JUST TAG THE VARIABLE WITH THE STATIC KEYWORD

ASIDE: I'VE MARKED THIS VARIABLE AS PUBLIC, ONLY BECAUSE WE HAVE YET TO TALK ABOUT STATIC METHODS - MEMBER DATA SHOULD BE PRIVATE

EXAMPLE 38 ADD A STATIC MEMBER VARIABLE TO A CLASS, AND USE IT OUTSIDE THE CLASS

THIS INVOLVES 4 STEPS

1. DECLARE THE VARIABLE INSIDE THE CLASS

2. DEFINE THE VARIABLE OUTSIDE THE CLASS

3. USE THE VARIABLE INSIDE THE CLASS

4. USE THE VARIABLE OUTSIDE THE CLASS

EXAMPLE 38 ADD A STATIC MEMBER VARIABLE TO A CLASS, AND USE IT OUTSIDE THE CLASS

THIS INVOLVES 4 STEPS

✓ 1. DECLARE THE VARIABLE INSIDE THE CLASS

2. DEFINE THE VARIABLE OUTSIDE THE CLASS

3. USE THE VARIABLE INSIDE THE CLASS

4. USE THE VARIABLE OUTSIDE THE CLASS

2. DEFINE THE VARIABLE OUTSIDE THE CLASS

THIS IS TRICKY. SOMEWHERE OUTSIDE THE CLASS,
YOU NEED A LINE DEFINING THE VARIABLE, LIKE THIS

```
int ComplexNumber::numObjectsCreated = 0; // define the static variable
```

NOTICE HOW WE USE THE SCOPE RESOLUTION OPERATOR, PREFIXED BY THE CLASS NAME

2. DEFINE THE VARIABLE OUTSIDE THE CLASS

THIS IS TRICKY. SOMEWHERE OUTSIDE THE CLASS,
YOU NEED A LINE DEFINING THE VARIABLE, LIKE THIS

```
int ComplexNumber::numObjectsCreated = 0; // define the static variable
```

NOTICE HOW WE USE THE SCOPE RESOLUTION
OPERATOR, PREFIXED BY THE CLASS NAME

THIS IS THE STANDARD WAY TO REFER TO ANY
MEMBER OR METHOD OF A CLASS OUTSIDE THAT CLASS

2. DEFINE THE VARIABLE OUTSIDE THE CLASS

THIS IS TRICKY. SOMEWHERE OUTSIDE THE CLASS,
YOU NEED A LINE DEFINING THE VARIABLE, LIKE THIS

```
int ComplexNumber::numObjectsCreated = 0; // define the static variable
```

"SOMEWHERE OUTSIDE THE CLASS"?

USUALLY, C++ CLASSES ARE SPLIT INTO .CPP
(SOURCE CODE) AND .H (HEADER FILES).

2. DEFINE THE VARIABLE OUTSIDE THE CLASS

THIS IS TRICKY. SOMEWHERE OUTSIDE THE CLASS,
YOU NEED A LINE DEFINING THE VARIABLE, LIKE THIS

```
int ComplexNumber::numObjectsCreated = 0; // define the static variable
```

"SOMEWHERE OUTSIDE THE CLASS"?

USUALLY, C++ CLASSES ARE SPLIT INTO .CPP
(SOURCE CODE) AND .H (HEADER FILES).

THERE, THE CLASS WOULD BE DECLARED IN THE .H FILE, AND THIS LINE WOULD BE IN THE .CPP FILE

2. DEFINE THE VARIABLE OUTSIDE THE CLASS

THIS IS TRICKY. SOMEWHERE OUTSIDE THE CLASS,
YOU NEED A LINE DEFINING THE VARIABLE, LIKE THIS

```
int ComplexNumber::numObjectsCreated = 0; // define the static variable
```

THE CLASS WOULD BE DECLARED IN THE .H FILE,
AND THIS LINE WOULD BE IN THE .CPP FILE

2. DEFINE THE VARIABLE OUTSIDE THE CLASS

```
int ComplexNumber::numObjectsCreated = 0; // define the static variable
```

THE CLASS WOULD BE DECLARED IN THE .H FILE, AND
THIS LINE WOULD BE IN THE .CPP FILE

THIS IS TRICKY.

IF YOU FORGET TO INCLUDE THE DEFINITION, THE COMPILER WILL THROW A LINKER ERROR! IN SOME OLDER COMPILERS, THE ERROR WILL POP UP AT RUNTIME

EXAMPLE 38 ADD A STATIC MEMBER VARIABLE TO A CLASS, AND USE IT OUTSIDE THE CLASS

THIS INVOLVES 4 STEPS

✓ 1. DECLARE THE VARIABLE INSIDE THE CLASS

2. DEFINE THE VARIABLE OUTSIDE THE CLASS

3. USE THE VARIABLE INSIDE THE CLASS

4. USE THE VARIABLE OUTSIDE THE CLASS

EXAMPLE 38 ADD A STATIC MEMBER VARIABLE TO A CLASS, AND USE IT OUTSIDE THE CLASS

THIS INVOLVES 4 STEPS

✓ 1. DECLARE THE VARIABLE INSIDE THE CLASS

✓ 2. DEFINE THE VARIABLE OUTSIDE THE CLASS

3. USE THE VARIABLE INSIDE THE CLASS

4. USE THE VARIABLE OUTSIDE THE CLASS

3. USE THE VARIABLE INSIDE THE CLASS

```
ComplexNumber() : realPart(0.0),complexPart(0.0)
  {
    // increment the static variable keeping track of objects created
    numObjectsCreated++;
    cout << "No arg-constructor called" << endl;
  }
  ComplexNumber(double c, double r) : realPart(r) , complexPart(c)
  {
    // increment the static variable keeping track of objects created
    numObjectsCreated++;
    cout << "Inside the 2-argument constructor" << endl;
  }

  ComplexNumber(const ComplexNumber& rhs) :
     realPart(rhs.realPart), complexPart(rhs.complexPart)
  {
    // increment the static variable keeping track of objects created
    numObjectsCreated++;
    cout << "Inside the copy constructor" << endl;
  }
```

THIS IS VERY SIMPLE, JUST USE IT INSIDE THE CLASS LIKE ANY OTHER MEMBER VARIABLE!

3. USE THE VARIABLE INSIDE THE CLASS

```
ComplexNumber() : realPart(0.0),complexPart(0.0)
  {
    // increment the static variable keeping track of objects created
    numObjectsCreated++;
    cout << "No arg-constructor called" << endl;
  }
  ComplexNumber(double c, double r) : realPart(r) , complexPart(c)
  {
    // increment the static variable keeping track of objects created
    numObjectsCreated++;
    cout << "Inside the 2-argument constructor" << endl;
  }

  ComplexNumber(const ComplexNumber& rhs) :
     realPart(rhs.realPart), complexPart(rhs.complexPart)
  {
    // increment the static variable keeping track of objects created
    numObjectsCreated++;
    cout << "Inside the copy constructor" << endl;
  }
```

THIS IS VERY SIMPLE, JUST USE IT INSIDE THE CLASS LIKE ANY OTHER MEMBER VARIABLE!

EXAMPLE 38 ADD A STATIC MEMBER VARIABLE TO A CLASS, AND USE IT OUTSIDE THE CLASS

THIS INVOLVES 4 STEPS

✓ 1. DECLARE THE VARIABLE INSIDE THE CLASS

✓ 2. DEFINE THE VARIABLE OUTSIDE THE CLASS

3. USE THE VARIABLE INSIDE THE CLASS

4. USE THE VARIABLE OUTSIDE THE CLASS

EXAMPLE 38 ADD A STATIC MEMBER VARIABLE TO A CLASS, AND USE IT OUTSIDE THE CLASS

THIS INVOLVES 4 STEPS

✓ 1. DECLARE THE VARIABLE INSIDE THE CLASS

✓ 2. DEFINE THE VARIABLE OUTSIDE THE CLASS

✓ 3. USE THE VARIABLE INSIDE THE CLASS

4. USE THE VARIABLE OUTSIDE THE CLASS

4. USE THE VARIABLE OUTSIDE THE CLASS

IF THE MEMBER VARIABLE IS PRIVATE, THEN OF COURSE YOU CAN'T USE IT OUTSIDE THE CLASS AT ALL

4. USE THE VARIABLE OUTSIDE THE CLASS

IF THE MEMBER VARIABLE IS PRIVATE, THEN OF COURSE YOU CAN'T USE IT OUTSIDE THE CLASS AT ALL

IF THE MEMBER VARIABLE IS PUBLIC, THEN YOU NEED TO USE THE SCOPE RESOLUTION OPERATOR, PRECEDED BY THE CLASS NAME

```
ComplexNumber::numObjectsCreated
```

THIS IS THE STANDARD WAY TO REFER TO ANY MEMBER OR METHOD OF A CLASS OUTSIDE THAT CLASS

4. USE THE VARIABLE OUTSIDE THE CLASS

```
ComplexNumber::numObjectsCreated
```

NOTE THAT WE USE THE CLASS NAME TO REFER TO THIS, NOT A SPECIFIC OBJECT VARIABLE!

THE VARIABLE IS SHARED ACROSS ALL OBJECTS OF A CLASS, REFERRING TO IT BY CLASS NAME MAKES IT CLEAR THAT IT'S A CLASS VARIABLE AND NOT AN OBJECT VARIABLE

4. USE THE VARIABLE OUTSIDE THE CLASS

```
int main()
{
  cout << "Number of ComplexNumber objects created so far: " << ComplexNumber::numObjectsCreated <<
endl;
  cout << "Create one object " << endl;
  ComplexNumber c1(1,2);

  cout << "Number of ComplexNumber objects created so far: " << ComplexNumber::numObjectsCreated <<
endl;
  cout << "Create one object " << endl;
  ComplexNumber c2(3,3);

  cout << "Number of ComplexNumber objects created so far: " << ComplexNumber::numObjectsCreated <<
endl;
  cout << "Create one object " << endl;
  ComplexNumber c3(4.5,5.3);

}
```

IF THE MEMBER VARIABLE IS PUBLIC, THEN YOU NEED TO USE THE SCOPE RESOLUTION OPERATOR, PRECEDED BY THE CLASS NAME

4. USE THE VARIABLE OUTSIDE THE CLASS

```
int main()
{
  cout << "Number of ComplexNumber objects created so far: " << ComplexNumber::numObjectsCreated <<
endl;
  cout << "Create one object " << endl;
  ComplexNumber c1(1,2);

  cout << "Number of ComplexNumber objects created so far: " << ComplexNumber::numObjectsCreated <<
endl;
  cout << "Create one object " << endl;
  ComplexNumber c2(3,3);

  cout << "Number of ComplexNumber objects created so far: " << ComplexNumber::numObjectsCreated <<
endl;
  cout << "Create one object " << endl;
  ComplexNumber c3(4.5,5.3);

}
```

IF THE MEMBER VARIABLE IS PUBLIC, THEN YOU NEED TO USE THE SCOPE RESOLUTION OPERATOR, PRECEDED BY THE CLASS NAME

EXAMPLE 38 ADD A STATIC MEMBER VARIABLE TO A CLASS, AND USE IT OUTSIDE THE CLASS

THIS INVOLVES 4 STEPS

✓ 1. DECLARE THE VARIABLE INSIDE THE CLASS

✓ 2. DEFINE THE VARIABLE OUTSIDE THE CLASS

✓ 3. USE THE VARIABLE INSIDE THE CLASS

4. USE THE VARIABLE OUTSIDE THE CLASS

EXAMPLE 38 ADD A STATIC MEMBER VARIABLE TO A CLASS, AND USE IT OUTSIDE THE CLASS

THIS INVOLVES 4 STEPS

✓ 1. DECLARE THE VARIABLE INSIDE THE CLASS

✓ 2. DEFINE THE VARIABLE OUTSIDE THE CLASS

✓ 3. USE THE VARIABLE INSIDE THE CLASS

✓ 4. USE THE VARIABLE OUTSIDE THE CLASS

EXAMPLE 39

ADD A STATIC MEMBER FUNCTION TO A CLASS, AND USE IT OUTSIDE THE CLASS

EXAMPLE 39 ADD A STATIC MEMBER FUNCTION TO A CLASS, AND USE IT OUTSIDE THE CLASS

THIS IS VERY SIMILAR TO USING A STATIC MEMBER VARIABLE, BUT SIMPLER -

ITS NOT MANDATORY TO DEFINE THE FUNCTION OUTSIDE THE CLASS

EXAMPLE 39 ADD A STATIC MEMBER FUNCTION TO A CLASS, AND USE IT OUTSIDE THE CLASS

THIS IS VERY SIMILAR TO USING A STATIC MEMBER VARIABLE, BUT SIMPLER -

ITS NOT MANDATORY TO DEFINE THE FUNCTION OUTSIDE THE CLASS

```
class ComplexNumber
{
private:
  float realPart;
  float complexPart;
  static int numObjectsCreated;
public:
  static int getNumObjectsCreated()
  {
    cout << "Inside the static method " << endl;
    return numObjectsCreated;
  }
```

EXAMPLE 39 ADD A STATIC MEMBER FUNCTION TO A CLASS, AND USE IT OUTSIDE THE CLASS

THIS IS VERY SIMILAR TO USING A STATIC MEMBER VARIABLE, BUT SIMPLER -

ITS NOT MANDATORY TO DEFINE THE FUNCTION OUTSIDE THE CLASS

JUST REMEMBER THE STATIC METHOD CAN ONLY ACCESS STATIC MEMBER VARIABLES, NOT OBJECT-SPECIFIC MEMBER VARIABLES

```
class ComplexNumber
{
private:
  float realPart;
  float complexPart;
  static int numObjectsCreated;
public:
  static int getNumObjectsCreated()
  {
    cout << "Inside the static method " << endl;
    return numObjectsCreated;
  }
```

EXAMPLE 39 ADD A STATIC MEMBER FUNCTION TO A CLASS, AND USE IT OUTSIDE THE CLASS

THIS IS VERY SIMILAR TO USING A STATIC MEMBER VARIABLE, BUT SIMPLER -

ITS NOT MANDATORY TO DEFINE THE FUNCTION OUTSIDE THE CLASS

```
cout << "Number of ComplexNumber objects created so far: "
     << ComplexNumber::getNumObjectsCreated() << endl;
cout << "Create one object " << endl;
ComplexNumber c1(1,2);
```

CALLING THE STATIC MEMBER FUNCTION INVOLVES THE SCOPE RESOLUTION OPERATOR AGAIN

EXAMPLE 39 ADD A STATIC MEMBER FUNCTION TO A CLASS, AND USE IT OUTSIDE THE CLASS

THIS IS VERY SIMILAR TO USING A STATIC MEMBER VARIABLE, BUT SIMPLER -

ITS NOT MANDATORY TO DEFINE THE FUNCTION OUTSIDE THE CLASS

EXAMPLE 39 ADD A STATIC MEMBER FUNCTION TO A CLASS, AND USE IT OUTSIDE THE CLASS

THIS IS VERY SIMILAR TO USING A STATIC MEMBER VARIABLE, BUT SIMPLER -

ITS NOT MANDATORY TO DEFINE THE FUNCTION OUTSIDE THE CLASS

YOU COULD ALWAYS CHOOSE TO DECLARE IN THE .H FILE, AND DEFINE IN THE .CPP FILE IF YOU SO CHOOSE THOUGH

EXAMPLE 40

UNDERSTAND WHAT WILL HAPPEN IF YOU FORGET TO DEFINE A STATIC MEMBER VARIABLE

EXAMPLE 40

UNDERSTAND WHAT WILL HAPPEN IF YOU FORGET TO DEFINE A STATIC MEMBER VARIABLE

USING A STATIC MEMBER VARIABLE INVOLVES 4 STEPS

1. DECLARE THE VARIABLE INSIDE THE CLASS

2. DEFINE THE VARIABLE OUTSIDE THE CLASS

3. USE THE VARIABLE INSIDE THE CLASS

4. USE THE VARIABLE OUTSIDE THE CLASS

RECAP

1. DECLARE THE VARIABLE INSIDE THE CLASS

```
class ComplexNumber
{
private:
  float realPart;
  float complexPart;
public:
  static int numObjectsCreated;
```

RECAP

EXAMPLE 40

UNDERSTAND WHAT WILL HAPPEN IF YOU FORGET TO DEFINE A STATIC MEMBER VARIABLE

USING A STATIC MEMBER VARIABLE INVOLVES 4 STEPS

✓ 1. DECLARE THE VARIABLE INSIDE THE CLASS

2. DEFINE THE VARIABLE OUTSIDE THE CLASS

3. USE THE VARIABLE INSIDE THE CLASS

4. USE THE VARIABLE OUTSIDE THE CLASS

RECAP

2. DEFINE THE VARIABLE OUTSIDE THE CLASS

THIS IS TRICKY. SOMEWHERE OUTSIDE THE CLASS, YOU NEED A LINE DEFINING THE VARIABLE, LIKE THIS

```
int ComplexNumber::numObjectsCreated = 0; // define the static variable
```

NOTICE HOW WE USE THE SCOPE RESOLUTION OPERATOR, PREFIXED BY THE CLASS NAME

2. DEFINE THE VARIABLE OUTSIDE THE CLASS

```
int ComplexNumber::numObjectsCreated = 0; // define the static variable
```

THIS IS TRICKY.

IF YOU FORGOT TO INCLUDE THE DEFINITION, THE COMPILER WILL THROW A LINKER ERROR! IN SOME OLDER COMPILERS, THE ERROR WILL POP UP AT RUNTIME

2. DEFINE THE VARIABLE OUTSIDE THE CLASS

```
int ComplexNumber::numObjectsCreated = 0; // define the static variable
```

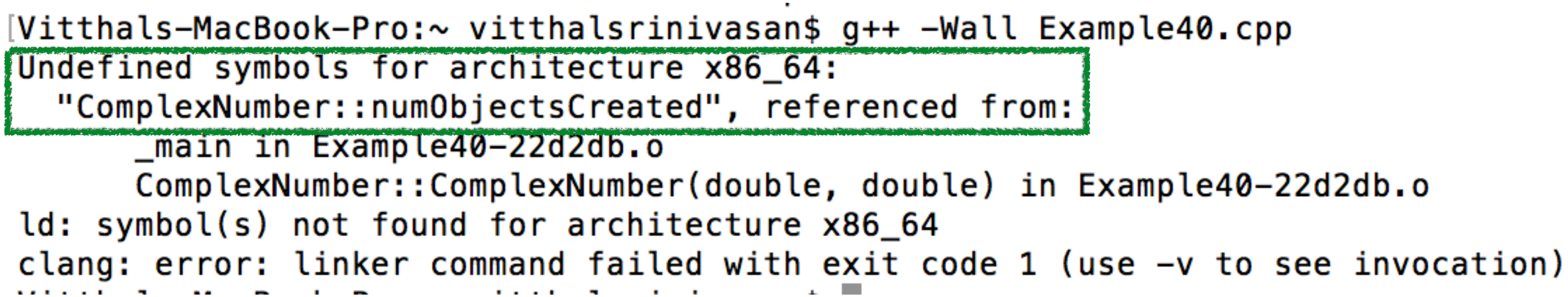
THIS IS TRICKY.

IF YOU FORGOT TO INCLUDE THE DEFINITION, THE COMPILER WILL THROW A LINKER ERROR! IN SOME OLDER COMPILERS, THE ERROR WILL POP UP AT RUNTIME

LET US INTENTIONALLY "FORGET" (COMMENT OUT THIS LINE) AND SEE WHAT HAPPENS

RECAP

LET US INTENTIONALLY "FORGET" (COMMENT OUT THIS LINE) AND SEE WHAT HAPPENS

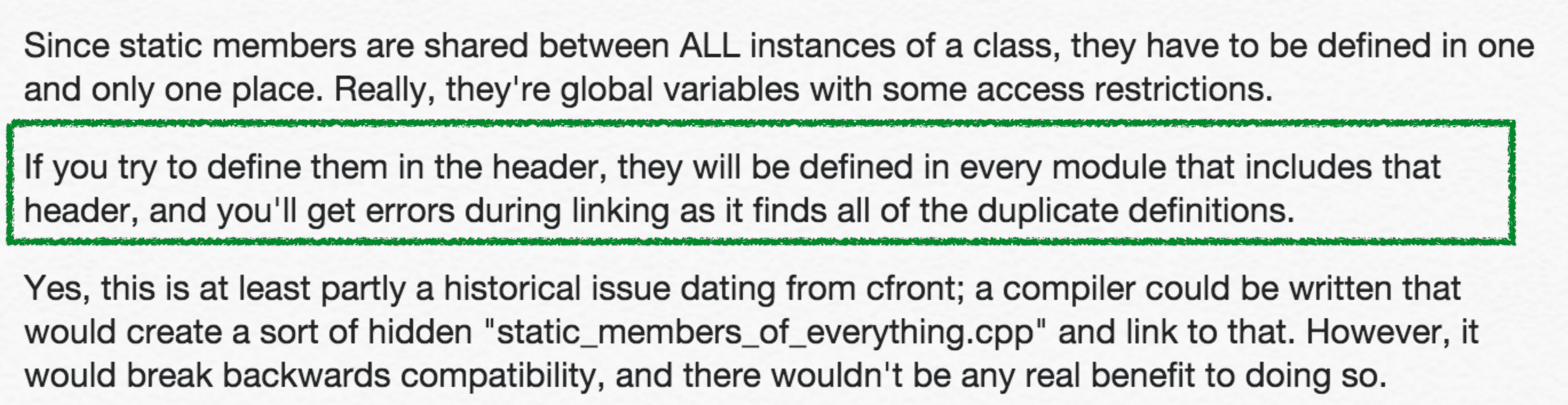


```
[Vitthals-MacBook-Pro:~ vitthalsrinivasan$ g++ -Wall Example40.cpp
Undefined symbols for architecture x86_64:
  "ComplexNumber::numObjectsCreated", referenced from:
      _main in Example40-22d2db.o
      ComplexNumber::ComplexNumber(double, double) in Example40-22d2db.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see invocation)
```

WHY DO WE NEED TO DO THIS STRANGE DEFINITION OUTSIDE THE CLASS?

WHY DO WE NEED TO DO THIS STRANGE DEFINITION OUTSIDE THE CLASS?

ITS A BIT ARCANE, BUT IF YOU'D REALLY LIKE TO KNOW - HERE'S WHY :-)



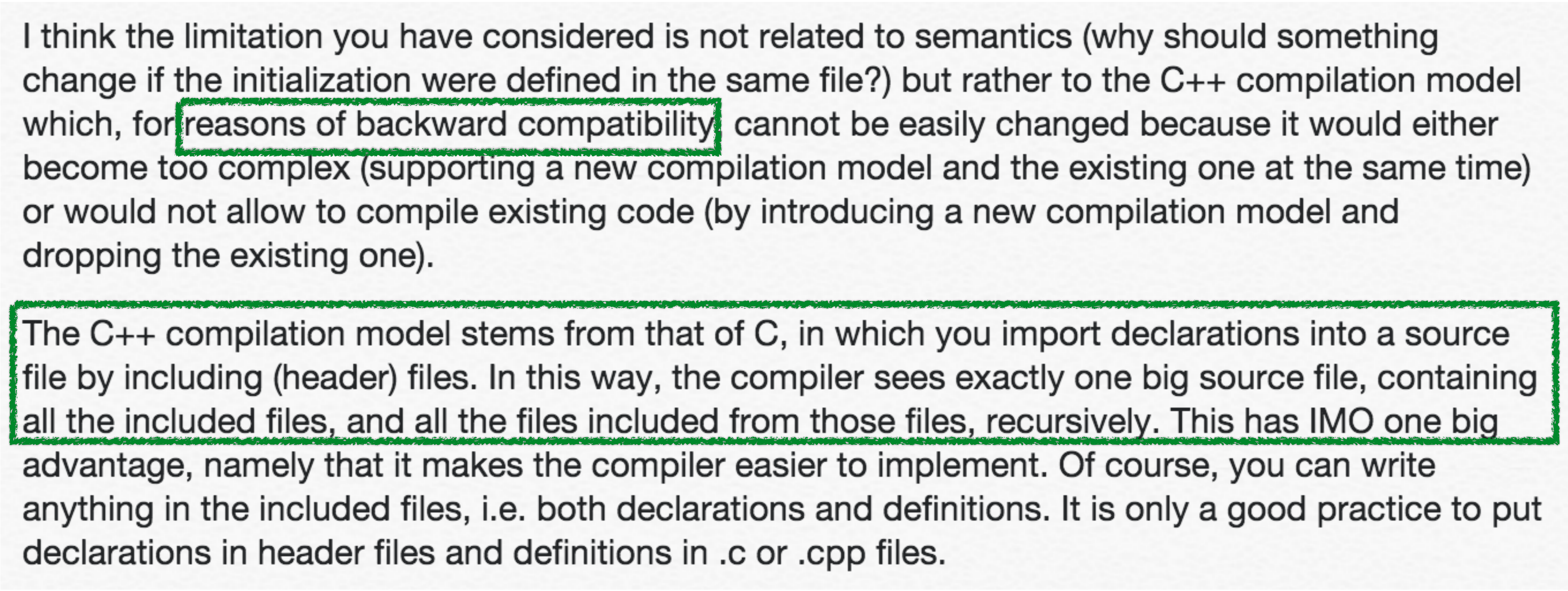
Since static members are shared between ALL instances of a class, they have to be defined in one and only one place. Really, they're global variables with some access restrictions.

If you try to define them in the header, they will be defined in every module that includes that header, and you'll get errors during linking as it finds all of the duplicate definitions.

Yes, this is at least partly a historical issue dating from cfront; a compiler could be written that would create a sort of hidden "static_members_of_everything.cpp" and link to that. However, it would break backwards compatibility, and there wouldn't be any real benefit to doing so.

WHY DO WE NEED TO DO THIS STRANGE DEFINITION OUTSIDE THE CLASS?

ITS A BIT ARCANE, BUT IF YOU'D REALLY LIKE TO KNOW - HERE'S WHY :-)



I think the limitation you have considered is not related to semantics (why should something change if the initialization were defined in the same file?) but rather to the C++ compilation model which, for reasons of backward compatibility, cannot be easily changed because it would either become too complex (supporting a new compilation model and the existing one at the same time) or would not allow to compile existing code (by introducing a new compilation model and dropping the existing one).

The C++ compilation model stems from that of C, in which you import declarations into a source file by including (header) files. In this way, the compiler sees exactly one big source file, containing all the included files, and all the files included from those files, recursively. This has IMO one big advantage, namely that it makes the compiler easier to implement. Of course, you can write anything in the included files, i.e. both declarations and definitions. It is only a good practice to put declarations in header files and definitions in .c or .cpp files.

EXAMPLE 41

UNDERSTAND HOW TO PROPERLY INITIALISE A CONST STATIC MEMBER VARIABLE

CONST

C++ HAS AN INTERESTING NEW KEYWORD CALLED `const`.

A VARIABLE CAN BE MARKED `const`, AND THEN ATTEMPT TO CHANGE ITS VALUE WILL THROW AN ERROR

RECAP

CONST

C++ HAS AN INTERESTING NEW KEYWORD CALLED `const`.

A VARIABLE CAN BE MARKED `const`, AND THEN ATTEMPT TO CHANGE ITS VALUE WILL THROW AN ERROR

A MEMBER FUNCTION OF AN OBJECT CAN BE MARKED `const`, WHICH MEANS THAT IT WILL NOT CHANGE ANY MEMBER VARIABLE OF THAT OBJECT

RECAP

EXAMPLE 30

DEFINE AND USE `const` VARIABLES

RECAP

EXAMPLE 30

DEFINE AND USE const VARIABLES

```
// create a const int variable - the declaration and initialisation must be in the same sentence
const int x = 5;
// Why must the initialisation occur in the definition itself?
// Because re-assignment to a const will not work
// x = 10;

// create a const string
const string firstName("Vitthal");
// Any attempt to modify this string will throw some ferocious errors!
//firstName.insert(0,"Mr. ");

return 0;
```

RECAP

EXAMPLE 30

DEFINE AND USE const VARIABLES

```
// create a const int variable - the declaration and initialisation must be in the same sentence
const int x = 5;
// Why must the initialisation occur in the definition itself?
// Because re-assignment to a const will not work
// x = 10;

// create a const string
const string firstName("Vitthal");
// Any attempt to modify this string will throw some ferocious errors!
//firstName.insert(0,"Mr. ");

return 0;
```

RECAP

EXAMPLE 30

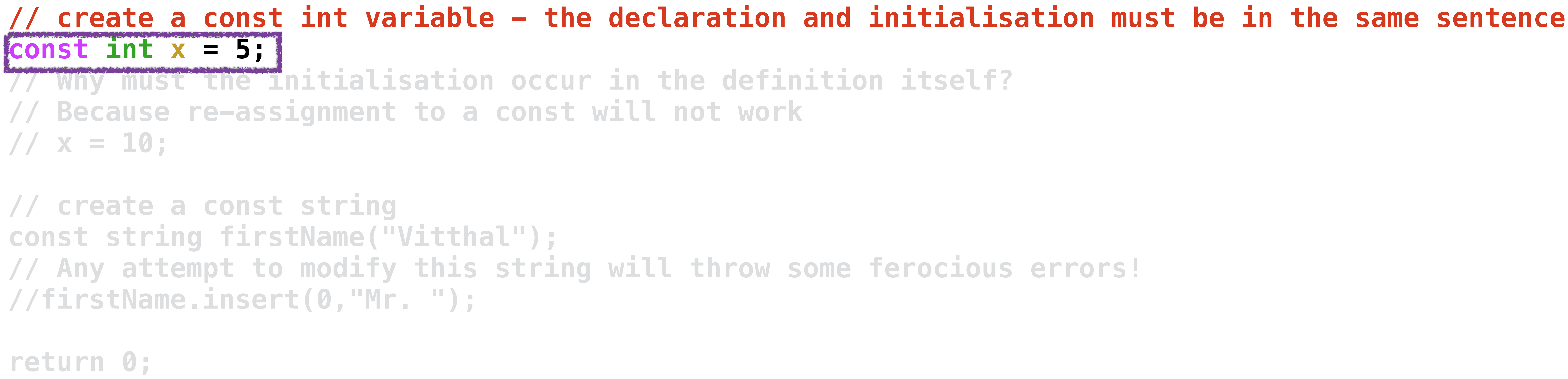
DEFINE AND USE const VARIABLES

```
// create a const int variable - the declaration and initialisation must be in the same sentence
const int x = 5;
// Why must the initialisation occur in the definition itself?
// Because re-assignment to a const will not work
   x = 10;

// create a const string
const string firstName("Vitthal");
// Any attempt to modify this string will throw some ferocious errors!
//firstName.insert(0,"Mr. ");

return 0;
```

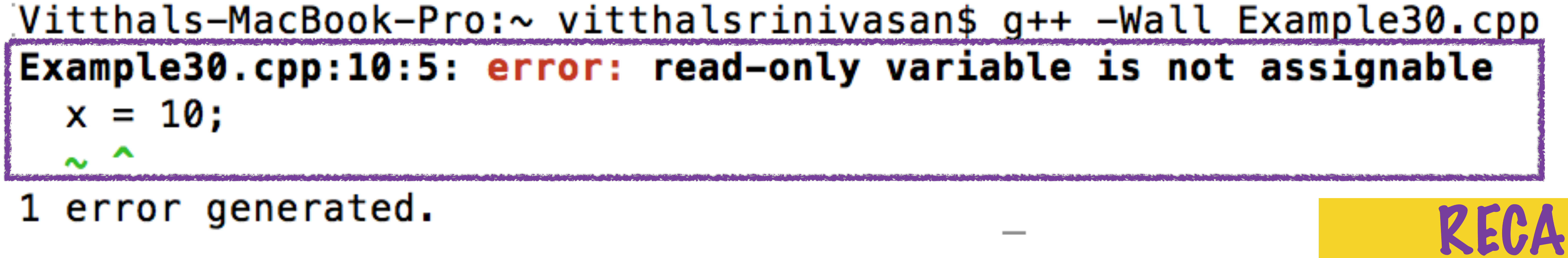
RECAP

EXAMPLE 30

DEFINE AND USE const VARIABLES



```
// create a const int variable – the declaration and initialisation must be in the same sentence
const int x = 5;
// Why must the initialisation occur in the definition itself?
// Because re-assignment to a const will not work
// x = 10;

// create a const string
const string firstName("Vitthal");
// Any attempt to modify this string will throw some ferocious errors!
//firstName.insert(0,"Mr. ");

return 0;
```



```
Vitthals-MacBook-Pro:~ vitthalsrinivasan$ g++ -Wall Example30.cpp
Example30.cpp:10:5: error: read-only variable is not assignable
  x = 10;
  ~ ^
1 error generated.
```

RECAP

BUT - WE JUST SAID THAT THE DECLARATION AND DEFINITION OF A STATIC MEMBER VARIABLE MUST BE SEPARATE!!

EXAMPLE 40

UNDERSTAND WHAT WILL HAPPEN IF YOU FORGET TO DEFINE A STATIC MEMBER VARIABLE

USING A STATIC MEMBER VARIABLE INVOLVES 4 STEPS

1. DECLARE THE VARIABLE INSIDE THE CLASS

2. DEFINE THE VARIABLE OUTSIDE THE CLASS

3. USE THE VARIABLE INSIDE THE CLASS

4. USE THE VARIABLE OUTSIDE THE CLASS

RECAP

1. DECLARE THE VARIABLE INSIDE THE CLASS

```
class ComplexNumber
{
private:
  float realPart;
  float complexPart;
public:
  static int numObjectsCreated;
```

RECAP

EXAMPLE 40

UNDERSTAND WHAT WILL HAPPEN IF YOU FORGET TO DEFINE A STATIC MEMBER VARIABLE

USING A STATIC MEMBER VARIABLE INVOLVES 4 STEPS

✓ 1. DECLARE THE VARIABLE INSIDE THE CLASS

2. DEFINE THE VARIABLE OUTSIDE THE CLASS

3. USE THE VARIABLE INSIDE THE CLASS

4. USE THE VARIABLE OUTSIDE THE CLASS

RECAP

2. DEFINE THE VARIABLE OUTSIDE THE CLASS

THIS IS TRICKY. SOMEWHERE OUTSIDE THE CLASS,
YOU NEED A LINE DEFINING THE VARIABLE, LIKE THIS

```
int ComplexNumber::numObjectsCreated = 0; // define the static variable
```

NOTICE HOW WE USE THE SCOPE RESOLUTION OPERATOR, PREFIXED BY THE CLASS NAME

EXAMPLE 41

UNDERSTAND HOW TO PROPERLY INITIALISE A CONST STATIC MEMBER VARIABLE

EXAMPLE 41

UNDERSTAND HOW TO PROPERLY INITIALISE A CONST STATIC MEMBER VARIABLE

APPROACH #1: DEFINE INSIDE, ALSO DUMMY-DEFINE OUTSIDE

APPROACH #2: DECLARE INSIDE, DEFINE OUTSIDE (EXACTLY LIKE NON-CONST)

APPROACH #2: DECLARE INSIDE, DEFINE OUTSIDE (EXACTLY LIKE NON-CONST)

INSIDE THE CLASS: (.H FILE)

```
class ComplexNumber
{
 const static double PI;
```

OUTSIDE THE CLASS: (.CPP FILE)

```
const double ComplexNumber::PI = 3.1415;
```

EXAMPLE 41

UNDERSTAND HOW TO PROPERLY INITIALISE A CONST STATIC MEMBER VARIABLE

APPROACH #1: DEFINE INSIDE, ALSO DUMMY-DEFINE OUTSIDE

APPROACH #2: DECLARE INSIDE, DEFINE OUTSIDE (EXACTLY LIKE NON-CONST)

APPROACH #1: DEFINE INSIDE, ALSO DUMMY-DEFINE OUTSIDE

INSIDE THE CLASS: (.H FILE)

```
class ComplexNumber
{
 const static double e = 2.71828 ;
```

THE DEFINITION AND DECLARATION ARE BOTH INSIDE THE CLASS..

OUTSIDE THE CLASS: (.CPP FILE)

```
const double ComplexNumber::e;
```

APPROACH #1: DEFINE INSIDE, ALSO DUMMY-DEFINE OUTSIDE

INSIDE THE CLASS: (.H FILE)

```
class ComplexNumber
{
 const static double e = 2.71828 ;
```

OUTSIDE THE CLASS: (.CPP FILE)

```
 const double ComplexNumber::e;
```

A DUMMY DECLARATION WITHOUT A VALUE SITS OUTSIDE THE CLASS

EXAMPLE 41

UNDERSTAND HOW TO PROPERLY INITIALISE A CONST STATIC MEMBER VARIABLE

APPROACH #1: DEFINE INSIDE, ALSO DUMMY-DEFINE OUTSIDE

APPROACH #1 WORKS ON ALL NEW COMPILERS

APPROACH #2: DECLARE INSIDE, DEFINE OUTSIDE (EXACTLY LIKE NON-CONST)

APPROACH #2 WORKS ON ALL COMPILERS