G52CPP
C++ Programming

Some example questions

and

some tactical hints about how to do the exam

Exam Content - repeat

- Most concepts appear somewhere on exam
- Standard class library not needed, except:
  - Recognise that **`cout << v`** means output/print the value of **`v`** and be able to make simple examples of this
  - Know **basics** of the string and vector classes
    - As seen in lectures, i.e. understand lecture samples
- Know the **common C-library functions**
  - What a function does, not parameter details
  - File access, string functions, input and output
- Ensure that you can create a **template function** and **operator overload**
  - And a macro (#define) and understand the difference
- Understand about conversion constructors and operators, copy constructors and assignment operators

Things to know (1)

- You need to know `char*` type C-strings
  - Where the `char*` points to an array of chars with a 0 at the end
- Be aware of array bounds issues
- And pointer arithmetic (e.g. `*p++ = *q++`)

Know about:

- `struct`s, and `class`es
  - member functions
  - `virtual` functions
  - inline functions
  - Scoping and ::
  - constructors (especially default and copy)
  - assignment operators, …
  - Conversion constructors and operators
- `struct` vs `class` vs `union`

Things to know (2)

- `const` members, parameters, references, pointers
- `static` local variables
- `static` member data and functions
- Pass/return by value vs by reference/pointer
- Function pointers
- Exceptions and exception handling
- Ensure that you can create a template function
- Ensure that you can provide operator overloads
  - e.g. for `operator*` and `operator*=`
- Understand the C++-style casts
  - What do they do? What are the differences between them? When would each be used?
  - static vs dynamic, const and reinterpret

Hints

- Pick and choose your questions according to what you are good at
  – Obvious? Why do so many people do Q1,2,3?
- Take some time to work out what each question (part) is asking
  – Is it something that you know how to do?
- Check the rest of the paper if you are stuck – sometimes it may jog your memory
  – e.g. does a code sample answer something?

How to revise

- Try the examples on the lecture slides
- Go through the slides in this presentation
- Try the past papers – ensure that you can do them without having to look up the answers
  - If you get stuck, look up how to do it, don’t look at the answer until you are sure
  - Copy code samples into a compiler and experiment with them
- Go through example code type questions
  - What is the question asking?
  - What do I need to know to answer it?
  - Are there any tricky bits?
  - What is the answer?

One type of short
question to expect:

“What is the output of …”

What is the output of this program?

```
#include <cstdio>
struct Base
{
   int foo()
   {
       static int j = 1;
       return j++;
   }
   virtual int bar()
   { return foo(); }
};
struct Sub : public Base
{
   int foo()
   {
       static int j = 10;
       return j++;
   }
   int bar()
   { return foo(); }
};
```

```
int main()
{
   Base b;
   Sub s;
   Base& br = s;
   Sub& sr = s;

   int i1 = b.foo();
   int i2 = s.foo();
   int i3 = br.foo();
   int i4 = sr.foo();
   int i5 = b.bar();
   int i6 = s.bar();
   int i7 = br.bar();
   int i8 = sr.bar();
   printf( "%d %d %d %d\n",
       i1, i2, i3, i4 );
   printf( "%d %d %d %d\n",
       i5, i6, i7, i8 );
   return 0;
}
```

How to answer…

- **What was the question testing?**
  - ???
- Use the information to avoid mistakes
- This sort of question tests your knowledge of things without asking you about them directly
- Make notes as you go along
- Write down variable values
- **At least give us a chance to give ‘working out marks’ if you make a mistake**

What can we observe?

```
#include <cstdio>
struct Base
{
   int foo()
   {
        static int j = 1;
        return j++;
   }
   virtual int bar()
   { return foo(); }
};
struct Sub : public Base
{
   int foo()
   {
        static int j = 10;
        return j++;
   }
   int bar()
   { return foo(); }
};
```

foo() and bar() differ in being virtual or not

foo() is NOT virtual
The type of the pointer or reference through which it is called matters

bar() IS virtual
The type of the actual object matters

What is the output of this program?

```
#include <cstdio>
struct Base
{
   int foo()
   {
       static int j = 1;
       return j++;
   }
   virtual int bar()
   { return foo(); }
};
struct Sub : public Base
{
   int foo()
   {
       static int j = 10;
       return j++;
   }
   int bar()
   { return foo(); }
};
```

The two foo() functions have separate **j** variables
The values are kept between function calls (static)

Each j increments by 1 AFTER the value is returned (POST increment)

As we work through…
we will need to look at:

- what object foo() or bar() is being called on
- what is the pointer type
- keep a track of the two j values (for the two functions)

Understand what the question tests

Question: What is this question testing?

Understand what the question tests

Question: What is this question testing?

a) Do you know about static local variables?

b) Do you know about virtual and non-virtual functions?

(Note that all foo()s come before all bar()s, so you get some marks for knowing one of these things.)

- Be careful with these questions
- Ask yourself:
  - “What is being tested by this question”
  - Then decide how to apply your knowledge

Now to answer it...

```
#include <cstdio>
struct Base
{
   int foo()
   {
        static int j = 1;
        return j++;
   }
   virtual int bar()
   { return foo(); }
};
struct Sub : public Base
{
   int foo()
   {
        static int j = 10;
        return j++;
   }
   int bar()
   { return foo(); }
};
```

```
int main()
{
   Base b;
   Sub s;
   Base& br = s;
   Sub& sr = s;

   int i1 = b.foo();
   int i2 = s.foo();
   int i3 = br.foo();
   int i4 = sr.foo();
   int i5 = b.bar();
   int i6 = s.bar();
   int i7 = br.bar();
   int i8 = sr.bar();
   printf( "%d %d %d %d\n",
        i1, i2, i3, i4 );
   printf( "%d %d %d %d\n",
        i5, i6, i7, i8 );
   return 0;
}
```

The ‘full’ correct answer

- Assuming that you ‘show your working’ please make the answer particularly clear
- i.e. Markers need to know which is the answer (to mark) and which is your working out, e.g.:
- “The output is:

```
    1 10 2 11
    3 12 13 14”
```

- “Answer:

```
    1 10 2 11
    3 12 13 14”
```

Similar question possibilities

- Many examples could be easily generated:
  - References or pointers vs pass by value
  - Pointer arithmetic (espec. ++) or pointers to pointers
  - static v non-static local variables
  - virtual v non-virtual functions
  - static v non-static member variables
  - Exception throwing and catching
- For questions like: “What is the output of this…”
  - Look for subtleties
  - These test knowledge indirectly
  - Test your ability to apply your knowledge rather than to memorise facts

Another type of question to expect:

“What is wrong with this code?”

Example question

- What is wrong with this program and how would you correct it?

```
#include <cstdio>

int main()
{
  char* str;
  scanf( "%s", str );
  printf("Input string was %s\n“, str);
  return 0;
}
```

Answer:

```
#include <cstdio>

int main()
{
// was char* str;
  char str[1024];
// Was scanf("%s", str);
  scanf( "%1023s", str );
  printf(
"Input string was %s\n“,
  str );
  return 0;
}
```

Two problems:

1. No memory allocated to hold the string
2. No limit on string length read

To fix the problems:

1. Create a char array instead of `char*` (can still be used as a `char*`)
2. Limit the length of the string that `scanf` can read to 1 less than array size

Another big problem in C/C++

- There is no bounds checking
  - You can use classes which will check for you, but they are slower than using native arrays
  - (use container classes – then there is)
- We don’t always know when we have overwritten past the end of an array
- You need to be VERY careful to ensure that:

1. Enough memory is allocated
2. When you use functions you limit the size that they can write

Concept : memory allocation

- There is a difference between a pointer and the thing it points to, e.g.:

1. When can you treat a **`char*`** as a string?
   - ONLY when it points to an array of characters with a zero on the end
2. When can you use a **`char*`** as the place to put a string that you read in (e.g. with **`scanf`**, **`fscanf`**, **`gets`**, **`fgets`**, etc)?
   - ONLY when you allocate memory to accept the string and make the **`char*`** point to the memory that you allocated

Another type of question to expect:

“Write code to …”

Write code to…

1. Give an implementation for a default constructor for this class…

2. Implement a function for class B so that the following code would compile:

```
B a, b, c;
a = b * c;
```

- Work out what the function needs to do first
- i.e. operator overload for * operator in this case

Quick code questions (08/09)

3d: Write a function called min() which takes two long parameters called val1 and val2 and returns the value of the lesser of the two values as a long , using the ternary operator (?:). (i.e. it should return the minimum of the two values.)

3e: Provide the code for a macro called MIN using #define which will perform the same function as your min() function in question 3d.

2010/2011 Q1b

- Provide C/C++ code to define a macro (using #define) called PRODUCT, which takes two parameters and returns the product of the two parameters.
- e.g. PRODUCT(2,3) is 6, PRODUCT(1,5) is 5 and PRODUCT(-3,4) is -12

Answer:

```
#define PRODUCT(a,b) ((a)*(b))
```

Quick ‘knowledge’ questions

Quick questions

- What is the difference between a class and a struct in C++?
- What is the :: operator used for in C++? Give two examples of its use.
- What is the difference between protected and private member data in C++?
- What is meant by the term ‘overloaded function’ in C++?
- Name four methods which may be created implicitly by a C++ compiler for a C++ class if they are needed.
- *“In addition to a default (no-parameter) constructor and a destructor, name two other methods which may be created implicitly by a C++ compiler for a C++ class if they are needed.”*

Example past paper question

2008/2009 Q1b

It is required to call the function **`printf()`** if, and only if, the integer variables **`x`** and **`y`** are both nonzero.

Which (one or more) of the following would have the desired result?

```
I.   if ( x & y )        { printf( "i" ); }
II.  if ( x && y )       { printf( "ii" ); }
III. if ( x | y )        { printf( "iii" ); }
IV.  if ( x || y )       { printf( "iv" ); }
V.   if ( !(!x || !y) ) { printf( "v" ); }
```

What is it testing?

- Do you know what the & operator does?
- What about the && operator?
- What is the difference between & and && ?
- How does ! work with |, ||, & and && ?

struct and union sizes

struct and union sizes

```
#pragma pack(1)

struct S1
{
   short s;
   char c;
};

struct S2
{
   long l1;
   unsigned long l2;
};

struct S3
{
   S1 a1;
   S2 a2;
};
```

```
union U1
{
   char c;
   short s;
   long l;
};

union U2
{
   long l;
   U1 u;
};
```

Q: What is the output assuming that:
`sizeof(short)` is 2
`sizeof(long)` is 4

Reminder: by definition `sizeof(char)` is 1

```
int main()
{
   printf( "S1 size %d\n", sizeof(S1) );
   printf( "S2 size %d\n", sizeof(S2) );
   printf( "S3 size %d\n", sizeof(S3) );
   printf( "U1 size %d\n", sizeof(U1) );
   printf( "U2 size %d\n", sizeof(U2) );
}
```

`struct` and `union` sizes

```
#pragma pack(1)

struct S1
{
   short s;
   char c;
};
```

2+1 = 3

```
struct S2
{
   long l1;
   unsigned long l2;
};
```

4+4 = 8

```
struct S3
{
   S1 a1;
   S2 a2;
};
```

3+8 = 11

```
union U1
{
   char c;
   short s;
   long l;
};
```

max(1,2,4) = 4

```
union U2
{
   long l;
   U1 u;
};
```

max(4,4) = 4

What is the output assuming that:
sizeof(short) is 2
sizeof(long) is 4

```
int main()
{
   printf( "S1 size %d\n", sizeof(S1) );
   printf( "S2 size %d\n", sizeof(S2) );
   printf( "S3 size %d\n", sizeof(S3) );
   printf( "U1 size %d\n", sizeof(U1) );
   printf( "U2 size %d\n", sizeof(U2) );
}
```

References, pointers, parameters and return types

Revision and clarifications

Parameter revision

- Passing a parameter to a function by value makes a copy of it

```
void RefFoo1( int i ) { … }
```

- Passing a pointer to a function by value makes a copy of the pointer

```
void RefFoo1( int* pi ) { … }
```

- Passing a parameter to a function by reference gives a new name for the actual thing (no copy)
  - Acts like passing in a pointer

```
void RefFoo1( int& ri ) { … }
```

Return type revision

- Returning something ‘by value’ makes a copy of the thing returned
  - Copy constructor is used for objects

```
int RefFoo1() { … return j; }
```

- Returning something ‘by reference’ returns the thing itself
  - No copy has to be made, but could be `const &` to **try** to avoid the original being altered

```
int& RefFoo1() { … return j; }
```

- Returning a pointer returns a copy of the pointer, which will point to the same thing

```
int* RefFoo1() { … return pj; }
```

- Note: You can return a reference to a pointer, as shown in the final lecture (not needed for exam!)

Danger: References and pointers

```
// Return pointer to parameter – danger
int* PFoo1( int i ) { return &i; }
// Return reference to parameter - danger
int& RefFoo1( int i )  { return i; }

// Return pointer passed in - fine
int* PFoo2( int* pi ) { return pi; }
// Return reference passed in - fine
int& RefFoo2( int& i ) { return i; }

// Return pointer to local – danger
int* PFoo3( int i ) { int j = 2; return &j; }
// Return reference to local - danger
int& RefFoo3( int i ) { int j = 2; return j; }

// Return pointer to static local - fine
int* PFoo4( int i ) {static int k = 3; return &k;}
// Return reference to static local - fine
int& RefFoo4( int i ) { static int k = 3; return k;}
```

Best wishes for doing well in the exam

Give me a 100% mark to celebrate this year ☺

The remaining slides are just practice questions

We can go through any you wish this afternoon at 2pm,

Or you can ask other questions

template functions

Question

- A function is required which will take two integers as parameters and which will use the ternary operator to determine the minimum of the two parameters passed in
- It should return this minimum value, as an **`int`**
- Provide an implementation of a function called **`mymin()`** which will perform as described above

Answer

- Return type is `int`
- Parameter types are both `int`
- Ternary operator is the `? :` operator

```
int mymin( int i, int j )
{
    return i < j ? i : j;
}
```

- Hint: Look elsewhere in the exam questions if you cannot remember the details for a function definition – there are bound to be a lot of them

Question

- Convert your answer to the previous part into a template function which will accept two parameters of the same type and return a value of the same type

Reminder: template functions

How to make a template function:

1. First generate function for specific type(s)
2. Next replace all copies of the types with template types
3. Finally, add the keyword `template` at the beginning and put the type(s) in the `<>` with keyword `typename` (or `class`)

Answer

1. Add the initial `template <typename T>`
2. Convert all the types to `T`

- Initial version:

```
int mymin( int i, int j )
{
   return i < j ? i : j;
}
```

- Template version:

```
template <typename T>
T mymin( T i, T j )
{
   return i < j ? i : j;
}
```

const

const – a clarification and reminder

- `const` means that the *thing* is constant
- `const` variable
  - Cannot alter the value of the variable. A constant
  - Have to set value on initialisation – otherwise it is too late
- `const` on function parameters:
  - The function guarantees that it will not change the parameter that is passed in – if it does then compiler will give an error
- `const` on member function
  - The `const` member function guarantees not to change the object
  - i.e. the `this` pointer is constant
- `const` reference
  - Cannot alter the thing that is referenced
- `const` pointer (two types)
  - Constant pointer – pointer cannot be changed : `char * const`
  - Pointer to something that cannot be changed : `const char *`

What is wrong with this code?

```
class C
{
public:
   float& get1()
   {
        return f;
   }
   float& get2() const
   {
        return f;
   }
   const float& get3() const
   {
        return f;
   }
private:
   float f;
};
```

- What is wrong with the code on the left?
- i.e. something in this code will prevent it compiling what is it?
- Hint: Something to do with **`const`**

Answer

```
class C
{
public:
   float& get1()
   {
        return f;
   }
   float& get2() const
   {
        return f;
   }
   const float& get3() const
   {
        return f;
   }
private:
   float f;
};
```

- Answer: **`get2()`** will not compile
- It returns a reference to the float in the current object
- This means that the float could be changed, through the reference
- But the function is const, guaranteeing that it cannot be used to change the object
- **`get1()`** will work – it makes no guarantees
- **`get3()`** will work – it returns a const ref, i.e. the reference cannot be used to change the object

Which lines will not compile?

```
class C
{
public:
   float& get1()
   { return f; }

   const float& get3() const
   { return f; }

private:
   float f;
};


int main()
{
   C ob;
   const C& ref = ob;
```

```
   float f1 = ob.get1();
   const float cf1 = ob.get1();
   float& rf1 = ob.get1();
   const float& crf1 = ob.get1();

   float f2 = ref.get1();
   const float cf2 = ref.get1();
   float& rf2 = ref.get1();
   const float& crf2 = ref.get1();

   float f3 = ob.get3();
   const float cf3 = ob.get3();
   float& rf3 = ob.get3();
   const float& rf3 = ob.get3();

   float f4 = ref.get3();
   const float f4 = ref.get3();
   float& rf4 = ref.get3();
   const float& rf4 = ref.get3();
}
```

Consider the first four

```
class C
{
public:
   float& get1()
   { return f; }

   const float& get3() const
   { return f; }

private:
   float f;
};


int main()
{
   C ob;
   const C& ref = ob;
```

```
   float f1 = ob.get1();
   const float cf1 = ob.get1();
   float& rf1 = ob.get1();
   const float& crf1 = ob.get1();

   float f2 = ref.get1();
   const float cf2 = ref.get1();
   float& rf2 = ref.get1();
   const float& crf2 = ref.get1();

   float f3 = ob.get3();
   const float cf3 = ob.get3();
   float& rf3 = ob.get3();
   const float& rf3 = ob.get3();

   float f4 = ref.get3();
   const float f4 = ref.get3();
   float& rf4 = ref.get3();
   const float& rf4 = ref.get3();
}
```

First four

- A non-`const float` reference is returned by `get1()`
- We can use this to initialise:
  - a float – i.e. a copy of the returned float
  - a constant float – i.e. a constand copy of the returned float
  - a float reference – it’s fine by us if it changes it, the returned ref is not const
  - a constant float reference – guarantees not to change it, but we don’t care even if it did

Next four: float from a const ref

```
class C
{
public:
   float& get1()
   { return f; }

   const float& get3() const
   { return f; }

private:
   float f;
};


int main()
{
   C ob;
   const C& ref = ob;
```

```
   float f1 = ob.get1();
   const float cf1 = ob.get1();
   float& rf1 = ob.get1();
   const float& crf1 = ob.get1();

   float f2 = ref.get1();
   const float cf2 = ref.get1();
   float& rf2 = ref.get1();
   const float& crf2 = ref.get1();

   float f3 = ob.get3();
   const float cf3 = ob.get3();
   float& rf3 = ob.get3();
   const float& rf3 = ob.get3();

   float f4 = ref.get3();
   const float f4 = ref.get3();
   float& rf4 = ref.get3();
   const float& rf4 = ref.get3();
}
```

Next four: float from a const ref

- All give compile error:

```
error: passing `const C' as `this'
argument of `float& C::get1()' discards
qualifiers
```

- What does this mean?
  - The **`this`** argument is the pointer to the current object and it is constant
  - But **`get1()`** is **not** a **`const`** function, so you cannot call it on a constant reference – it would allow the ref to be altered
- All of these four lines will fail to compile
  - the **`get1()`** fails – it’s irrelevant what we try to do with it

Third four

```
class C
{
public:
   float& get1()
   { return f; }

   const float& get3() const
   { return f; }

private:
   float f;
};


int main()
{
   C ob;
   const C& ref = ob;
```

```
   float f1 = ob.get1();
   const float cf1 = ob.get1();
   float& rf1 = ob.get1();
   const float& crf1 = ob.get1();

   float f2 = ref.get1();
   const float cf2 = ref.get1();
   float& rf2 = ref.get1();
   const float& crf2 = ref.get1();

   float f3 = ob.get3();
   const float cf3 = ob.get3();
   float& rf3 = ob.get3();
   const float& rf3 = ob.get3();

   float f4 = ref.get3();
   const float f4 = ref.get3();
   float& rf4 = ref.get3();
   const float& rf4 = ref.get3();
}
```

Third four

- **`get3()`** returns a constant reference
- You can use it to initialise a float (a copy)
- You can store it in another constant reference
- You CANNOT just make a non-constant reference refer to it – would allow it to be altered
- This is the only line which fails:

```
    float& rf3 = ob.get3();
```

- However, you could use **`const_cast`** to remove the **`const`**-ness

```
float& rf3 = const_cast<float&>(ob.get3());
```

Last four

```
class C
{
public:
   float& get1()
   { return f; }

   const float& get3() const
   { return f; }

private:
   float f;
};


int main()
{
   C ob;
   const C& ref = ob;
```

```
   float f1 = ob.get1();
   const float cf1 = ob.get1();
   float& rf1 = ob.get1();
   const float& crf1 = ob.get1();

   float f2 = ref.get1();
   const float cf2 = ref.get1();
   float& rf2 = ref.get1();
   const float& crf2 = ref.get1();

   float f3 = ob.get3();
   const float cf3 = ob.get3();
   float& rf3 = ob.get3();
   const float& rf3 = ob.get3();

   float f4 = ref.get3();
   const float f4 = ref.get3();
   float& rf4 = ref.get3();
   const float& rf4 = ref.get3();
}
```

Last four

- `get3()` returns a constant reference
  - This is safe, even when called on a constant object
    - We are guaranteeing not to change it anyway
  - Previously it failed in trying to return a `float&` from a constant object (using `get1()`)
  - Now it succeeds in getting a `const float&`, using `get3()`, since we are guaranteeing not to change anything – the reference that we get is `const`
- So, this is the only line which fails:

```
float& rf4 = ref.get3();
```

  - We are trying to assign a `const` reference (from `get3()`) to a non-`const` reference – which would lose the `const` property
  - Could use a `const_cast` to remove the `const`-ness

Question

You have been provided with a simple class, C, and some functions which use it. However, the code will not compile. Correct the class definition.

```
class C
{
public:
   // Constructor
   C(int i = 0) : i(i)
   {}
   // Get value
   int get()
   { return i; }
   // Set value
   void set(int i)
   { this->i = i; }
private:
   int i;
};
```

```
void output( const C& c )
{
   using namespace std;
   cout << c.get()
        << endl;
}

int main()
{
   C c1(2), c2;
   output( c1 );
   output( c2 );
}
```

Just use namespace in this function

Answer

The parameter **`c`** of **`output()`** is passed by *constant* reference. It is used to call **`get()`**. **`get()`** was not **`const`**, so didn't guarantee to not change **`c`**

```
class C
{
public:
   // Constructor
   C(int i = 0) : i(i)
   {}
   // Get value
   int get() const
   { return i; }
   // Set value
   void set(int i)
   { this->i = i; }
private:
   int i;
};
```

```
void output( const C& c )
{
   using namespace std;
   cout << c.get()
        << endl;
}

int main()
{
   C c1(2), c2;
   output( c1 );
   output( c2 );
}
```

Function pointers

Question

- What is the output of the following code?

```
#include <iostream>

int mult2( int i )
{
   return i*2;
}

int square( int i )
{
   return i*i;
}

int main()
{
   using namespace std;
   int (*f)(int) = &mult2;
   int (*g)(int) = &square;

   for (int i = 0; i < 5 ; i++)
       cout << (*f)(i)
              << endl;

   for (int i = 0; i < 5 ; i++)
       cout << (*g)(i)
              << endl;

   for (int i = 0; i < 5 ; i++)
       cout << (*g)((*f)(i))
              << endl;

   f = g;

   for (int i = 0; i < 5 ; i++)
       cout << (*f)(i) << endl;
}
```

Answer: function pointers

- What is the output of the following code?

```
#include <iostream>

int mult2( int i )
{
   return i*2;
}

int square( int i )
{
   return i*i;
}

int main()
{
   using namespace std;
```

```
Output:
0
2
4
6
0
1
4
9
0
4
16
36
0
1
4
9
```

```
int (*f)(int) = &mult2;
int (*g)(int) = &square;

for (int i = 0; i < 4 ; i++)
     cout << (*f)(i)
           << endl;
```

f=mult2

```
for (int i = 0; i < 4 ; i++)
     cout << (*g)(i)
           << endl;
```

g=square

```
for (int i = 0; i < 4 ; i++)
     cout << (*g)((*f)(i))
           << endl;
```

g=square
f=mult2
square(mult2(i))

```
f = g;

for (int i = 0; i < 4 ; i++)
     cout << (*f)(i) << endl;
```

f=square

static

static – a reminder

- Static applies to three things:

1. Local variables
   - Maintain their value beyond function calls
   - Not stored on the stack
2. Global functions and variables
   - Hides them within a file, file access only
   - Does not show them to the linker
3. Member functions/data (in a class)
   - Associated with the class rather than instances of the class (next week)
   - i.e. functions have no **`this`** pointer
     - So cannot access non-static member data – they would not know which object to affect

Question

- What is the output of the following code?

```
int f1( int i )
{
  int j = ++i;
  return j;
}

int f2( int i )
{
  static int j = ++i;
  return j;
}
```

```
int main()
{
  using namespace std;
  int i;

  for (i=0 ; i<5 ; i++)
     cout << f1(i)
          << endl;

  for (i=0 ; i<5 ; i++)
     cout << f2(i)
          << endl;
}
```

Answer: static local variables

- What is the output of the following code?

```
int f1( int i )
{
   int j = ++i;
   return j;
}

int f2( int i )
{
   static int j = ++i;
   return j;
}
```

```
Output:
1
2
3
4
5
1
1
1
1
1
```

```
int main()
{
  using namespace std;

  for ( int i = 0
          ; i < 5
          ; i++ )
     cout << f1(i)
          << endl;

  for ( int i = 0
          ; i < 5
          ; i++ )
     cout << f2(i)
          << endl;
}
```

Question

- What is the output of the following code?

```
int j;
static int k;

int f1( int i )
{
  j = ++i;
  return j;
}

int f2( int i )
{
  k = ++i;
  return k;
}
```

```
int main()
{
  using namespace std;

  for ( int i = 0
           ; i < 5
           ; i++ )
     cout << f1(i)
          << endl;

  for ( int i = 0
           ; i < 5
           ; i++ )
     cout << f2(i)
          << endl;
}
```

Answer: static global variables

- Static globals are just not accessible outside of the file. Within the file they make no difference!

```
int j;
static int k;

int f1( int i )
{
   j = ++i;
   return j;
}

int f2( int i )
{
   k = ++i;
   return k;
}
```

```
Output:
1
2
3
4
5
1
2
3
4
5
```

```
int main()
{
  using namespace std;

  for ( int i = 0
           ; i < 5
           ; i++ )
     cout << f1(i)
          << endl;

  for ( int i = 0
           ; i < 5
           ; i++ )
     cout << f2(i)
          << endl;
}
```