Question 1

1a:
char ptr[] = “Hello World”; char a = ptr[1], b = *(ptr+6);
Creates an array of 12 elements, 11 visible letters and a character value 0 at the end.
i true
ii true
iii false
iv false
v true

1b
only C++ since it has a function in a struct
i.e. (ii) only

1c: tests knowing FILE* and the basic FILE* functions.
Correct code:

```
FILE *file;
file = fopen( argv[1], "w" );
fputs( argv[2], file );
fclose(file);
```

So errors are on the file. lines, and with the missing *

1d: tests knowing about bitwise fields and union/struct differences.

sizeof(A) = 4 = sizeof(int), since sizeof(int) > sizeof(char)
sizeof(B) = 5 = sizeof(int) + sizeof(char)
sizeof(C) = 5 = sizeof(A) + sizeof(char)
sizeof(D) = 5 = sizeof(C) > sizeof(A)
sizeof(E) = 1; bit fields
sizeof(F) = 1 = max of sizeof(E) or 1. If their answer for E was bigger than 1 then accept the bigger answer.

1e tests whether they understand the pointer arithmetic
i) Problem is that the zero terminator on first string gets copied.
ii) Will just output the first string, i.e. “Hello”
iii) Add the text “p--;” just after or just before the text “q = str2;”, to remove the splitting ‘\0’.
Also accept any equivalent answer to remove the ‘0’, presumably by changing p.

1f Mainly tests knowing . vs ->
i) text should be x->a.i == y.a.i
i.e. Left of == is: x->a.i
right of == is y.a.i

ii) Should be comparei(&b1, b2)

Question 2

2a
struct _LE *
(Since it is C not C++, they need the 'struct' in there.)

2b c not c++ so has to use malloc. Needs the +1 for NULL terminator
pEntry->pFirstName = malloc(strlen(pFirstName)+1);
strcpy(pEntry->pFirstName, pFirstName);
pEntry->pLastName = malloc(strlen(pLastName)+1);
strcpy(pEntry->pLastName, pLastName);

2c
i) (Insert at head of list)
pNewEntry->pNext = g_pHead;
g_pHead = pNewEntry;

ii) (Insert elsewhere in list)
pNewEntry->pNext = pPreviousEntry->pNext;
pPreviousEntry->pNext = pNewEntry;

iii) (Iteration through list)
(pPreviousEntry->pNext != NULL)
&&
pPreviousEntry->pNext->iAge > iAge

Order is important. Must check NULL prior to dereferencing.
Note: very similar to the condition for inserting at head of list.

2d
The simple solution:

```
bool operator==( const ListEntry& o1, const ListEntry& o2 )
{
      return strcmp( o1.pLastName, o2.pLastName ) == 0;
}
```

(End of simple solution)

Alternatively, they could make it a member of ListEntry struct if they want. If so, then it needs to at least be declared (even if not defined) within struct declaration, and won't be able to use the name ListEntry.

```
e.g.
typedef struct _LE
{
      char* pFirstName;
```

```
      char* pLastName;
      int iAge;
      struct _LE* pNext;

      bool operator==( const _LE& o2 ) const
      {
            return strcmp( this->pLastName, o2.pLastName ) == 0;
      }
} ListEntry;
```

Also note:
They don't have to use strcmp if they can find an equivalent way.
They could use !strcmp() instead of strcmp()==0

For full marks, const should be specified for both parameters (including implicit this pointer (i.e. const function) if the member function version is used).

Question 3

3a: 2 other methods created implicitly
Copy constructor
Assignment operator

3b
4,7,48,0,2,2,17

3c (only first number is actual answer, rest is explanation)
1 initial (i=2)
2 (i=3, i=10)
11 (i=10 still)
11 (i=10 still, then I = 20)
20 static = 21
21 static = 22, I = 30
30 pi moves on.
1 = diff in positions.

3d
(i) … i.e. elipsis/3 dots.
(ii)
1 : Caught an A
2 : Caught an A
3 : Caught something else
4 : Caught a B*

Question 4

4a
Multiple functions with the same name but different parameter types, where the parameter types are used to determine which function to call.

4b Tests knowing difference between struct and class
Sample 1 won't compile because the constructor is private (by default for class)
Sample 2 will compile.

4c Tests not being scared of function pointers
Note: func a means return 2, func b means return input + 1

f1 = a, f2=a
i = 2
f1 = b, f2=a
j = 21
k = 2
l = 2
m = 3
n = 62

So answer: 2,21,2,2,3,62

4d Copy constructor, assignment operator

1	(a=1,e=1,f=1)
4	(b=4)
3	(c=a+2=3)
6	(d=b+2=6)
9	(e=b+5)
11	(f=b+5=a+5+5)

4e
Dynamic cast is used to safely cast from a base class pointer or reference to a sub-class pointer or reference.
It will verify that the object really is of the sub-class type and if not it will return NULL (if casting a pointer) or throw an exception (if casting a reference).

Example:

```
class A { … };
class B : A { … }

void foo( A* pa )
```

```
{
        B* pb = dynamic_cast<B*>(pa);
        if ( pb == NULL )
        {
                // Handle cannot cast
        }
}
```

Question 5

5a)
Only (iv) is valid.
Note: (iii) has a semi-colon at the end. (i) and (ii) use some kind of return.

i.e. `#define DIFF(x,y) ((x)>(y)) ? ((x)-(y)) : ((y)-(x))`

5b)
```
int diff( int x, int y )
{
      return (x>y) ? (x-y) : (y-x);
}
```

5c)
```
template <class T1, class T2>
T1 diff( T1 x, T2 y )
{
      return (T1)((x>y) ? (x-y) : (y-x));
}
```

For full marks:
should say template
<class …> or <typename …>
Needs two parameter types, with different names
Return type should match param 1 type
Return type should be cast to type of param 1

5d) Tests knowing non-virtual funcs are different from Java.

```
15      Initial: b = 15, s = (base 25, sub 20)
20      Then b = 1
1       Then s = (base 2, sub 20)
20
2
20      Then s = (base 3, sub 20)
3       Then s = (base 3, sub 20)
20
```

5e) Incorrect lines are all due to const.
F G O Q S