Answers to 2008/2009 G52CFJ exam

Below are the answers to last year's exam, so that you can check how you did.
In many cases I have given an example answer and an idea of what elements we would look for.

Question 1

1a) (i) only.

1b) (ii) and (v) only

1c) (iv) and (v) OR (v) only
(hindsight showed that the question was more ambiguous than intended)

1d)
Problem 1:
usage of argv[1] before you know it exists, or similar explanation
Solution:
Fix: Easiest solution is to remove arg1 variable and just use atoi(argv[1]) in the printf
Alternatively, split the arg1 into an initial declaration of argv and an assignment of the value after the if block.

Problem 2:
Parameter type token in the printf should be %d not %c.
Solution:
Fix: Change %c to %d

1e) You could run this program and see the answer.
Answer:
3 5 3
6 3 3
6 **3 3**

1f)
4 answers:
Will accept either "a default constructor" or "a no-parameter constructor"
A copy constructor
An assignment operator
A destructor

Question 2

This is a difficult question to mark. Basically we tested the answers given to determine whether the answer was correct, rather than whether it matched a model answer.

2a :
class has private members by default, struct has public members by default
Mark for mentioning private/public.
Mark for mentioning that it is the members.

2b :

```
typedef struct dle
{
        struct dle* pNext;
        struct dle* pPrev;
        long lSaleId;
        long lProductId;
        long lSalesPersonId;
        long lQuantity;
        double dPrice;
} Entry;
```

typedef is optional, and without it the ‘Entry’ will also be missing.
int or possibly short could be used instead of long.
Important part is the forward and backward pointers and the struct definition itself.
Struct name should be propagated forwards into later questions.
Note: If no typedef then they need to specify ‘struct’ later, since this is C not C++ code.

2c :
Things to check:
Need correct function definition, i.e. params, name, brackets, etc
Need malloc for allocating space.
Should assign each element of the new structure
Should check for special case of no items
If no items then set pFirst and set the pPrev of current item to NULL
If existing item: set pTail->pNext to new entry,
and newentry->pPrev to the old last item
Should set pLast/pTail to point to new struct
Should set pNext on new structure to NULL
For full marks return success or failure depending upon outcome of malloc

They need to specify a global head/first pointer, and preferably a tail/last pointer
Entry* g_pFirst = NULL;
or
Entry* g_pHead = NULL;

Entry* g_pLast = NULL;
or
Entry* g_pTail = NULL;

They also need the function itself:
Example answer:

```
int StoreSalesData( long lProductId, long lSalesPersonId, long lQuantity, double dPrice )
{
	Entry* pNewEntry = malloc(sizeof(Entry));
	if ( pNewEntry == NULL )
		return 0; // Failed to allocate memory

	pNewEntry->lSaleId = GenerateUniqueSalesId();
	pNewEntry->lProductId =	lProductId;
	pNewEntry->sSalesPersonId = sSalesPersonId;
	pNewEntry->sQuantity = sQuantity;
	pNewEntry->dPrice = dPrice;
	pNewEntry->pNext = NULL;

	if ( g_pFirst == NULL )
	{
		g_pFirst = pNewEntry;
		pNewEntry->pPrev = NULL;
	}
	else
	{
		g_pLast->pNext = pNewEntry;
		pNewEntry->pPrev = g_pLast;
	}

	g_pLast = pNewEntry;

	return 1; // success
}
```

Note: This year's exam avoids having to write lots of code like this.

2d : Check for the following points in the answer:
Find the relevant entry
- Use a new local variable of appropriate pointer type
- Start at head/first
- Iterate through using pNext on current item
- Check sale id correctly

Upon finding entry:
- Unlink prev
  - special case if first – set pHead
- Unlink next
  - special case if last – set pTail
- They must free() the entry that was unlinked

Example answer:

```
int FindAndRemoveStoreSalesData( long lSaleId )
{
	Entry* pEntry = g_pFirst;
	while ( pEntry != NULL )
	{
		if ( pEntry->lSaleId == lSaleId )
		{
			if ( pEntry->pPrev == NULL )
				g_pFirst = pEntry->pNext;
			else
				pEntry->pPrev->pNext = pEntry->pNext;

			if ( pEntry->pNext == NULL )
				g_pLast = pEntry->pPrev;
			else
				pEntry->pNext->pPrev = pEntry->pPrev;

			free(pEntry);

			return 1;
		}
		pEntry = pEntry->pNext;
	}
	return 0;
}
```

2e: 3 marks total for sensible answers, 1 mark for each point made
e.g.:
Put all of the code into a class as class member functions and member data.
Use new instead of malloc, delete instead of free.
Encapsulation of data
Use exception handling with new instead of checking for NULL.
If returning a value then return bool (true/false) instead of int (1/0).
Use an inner/nested class for the list items.
Hide the head/first and tail/last variables by making them private, accessible only using the class members.

Question 3

3a (iii) and (iv)

3b ONLY (ii)

3c
Mention #define of a UNIQUE label.
Mention #ifndef and #endif.
Mention #endif.
Saying #define and #ifndef at top of file and #endif at bottom of file.

Example answer:
Determine a unique label (called UNIQUE_LABEL below) for the file, such as the capitalized file path with underscores for the /s in it.
Put the following code at the top of the file:
#ifndef UNIQUE_LABEL
#define UNIQUE_LABEL
And this line at the bottom:
#endif

3d
Example answer:

```
long min( long val1, long val2 )
{
        return (val1 < val2 ) ? val1 : val2;
}
```

Optionally they could inline it

Check:
Returns long
Two long parameters
Correct use of ternary operator.
Code is actually correct. i.e. would compile.

3e

```
#define MIN(a,b) ((a < b) ? a : b)
```

Extra brackets may be used and are safer.
Can be split across lines as long as they escape the line breaks.

3f Example answer:
When the evaluation of the argument has a side-effect, because the side-effect will be executed twice on the lesser of the values.

3g
Example answer:

```
template <class myType>
inline myType GetMin (myType a, myType b)
{ return (a<b?a:b); }
```

The inline is optional.
Keyword *class* could be replaced by *typename*

Question 4

4a :
Examples (for 1 mark each, max 4):
malloc merely allocates space, it is unaware of the details of the object and has to be told the size of the object
new calls the constructor, hence initialization is performed
new throws exception on failure, malloc returns NULL
operator new can be overridden easily to change behaviour for individual types
mention the [] form of new and delete

4b :
Example answer:
Changing the meaning of an operator when applied to a type
The meaning of the operator will be determined by the types of the operands
For example, changing the meaning of the assignment operator is a case of operator overloading.
(They do not need to give an example.)

4c
delete should be delete[] (i.e. array delete)
Array length is not big enough for the second usage. (Needs to be 40+)
Note: The const is fine – but they may think that this is the problem.

4d
a comes before b
bb comes before b
ab comes after aa
ab comes before bb
aa is the same as aa
a comes after ab

(ii)
Example answer:
Incorrect lines in output:
b and bb are the wrong way around
a and ab are the wrong way around
Reason:
Cause is that the return 1 and return -1 are the wrong way round, so it incorrectly sorts the values when one string is a prefix of the other.
Solution:
Can be fixed by switching the return values of 1 and -1.

4e
The following are the compilation/linking errors:
=NULL should be ==NULL
FILE should be FILE*
fshut should be fclose
Runtime error: Array not big enough. Runs off the end because pointer is not set back to the beginning of array prior to the read.

Question 5

5a (8 marks total)
1 mark each for the name of each cast.
1 mark each for an explanation of when each case is used.

dynamic_cast
To perform a safe conversion from a base class to a sub-class while checking the type, on polymorphic references or pointers.
Should mention safe or checking the type. Potentially mention returning NULL if conversion fails.
Half mark if none of safety/checking/return NULL are mentioned.

reinterpret_cast
Reinterpret the value of the bits.
Treat the bits in memory as a new type without any conversion at all of the bits

const_cast
Remove or add const-ness or volatile-ness.

static_cast
Normal casts between types which can be performed with only compile-time information, or where run-time checking is not required. Cannot be used to remove const-ness or require knowledge of sub-types. Probably the most commonly used cast.
Explanation will probably be in terms of what it cannot do, or that it covers what the rest of the casts don't.

5b
Base : 3
Derived : 4
Base : 1
Base : 4
Base : 3
Derived : 4
Base : 4
Note: Need to verify both the base/derived is correct and that the value is correct
Give half a mark if they get just base/derived or just the number for each one.

5c
Red Green Orange Indigo Violet

Mark each position individually.
If they put them in a column instead of next to each other then still give marks – it's not marking cout knowledge, but should be marking the exceptions and virtual function knowledge.

Testing knowledge that pointers can be thrown, knowledge of virtual functions and knowledge that base class pointer will match sub-class pointer in a catch clause.
(Note: Warning is given by compiler due to this.)