Engineering Tripos Part IA FIRST YEAR

Paper 4: Mathematical Methods

COMPUTING

Examples Paper 4/1

Straightforward questions are marked †.
Tripos standard (but not necessarily Tripos length) questions are marked ∗.
Hints and answers can be found at the back of the paper.

Questions 1 to 13 are elementary review exercises covering basic C++ principles and control statements. They should take no more than a few minutes each. If you have difficulty with these questions, refer to Sections 1–6 of the Tutorial Guide to C++ Programming, *on the Departmental web pages, and also re-read the online handout of the Michaelmas lab. You are also highly encouraged to type the code fragments that appear in the questions and play with them on your own computer. Instructions on how to obtain compilers for your own computers can be found on the Departmental web pages at* `http://www-h.eng.cam.ac.uk/` *under the C++ section.*

1. † State whether each of the following statements is true or false. If false, explain why.

   (a) When a program is executed, any comments cause the computer to display on the screen the text after the `//` symbol.

   (b) All variables must be declared before they are used.

   (c) All variables must be assigned a type when they are declared.

   (d) C++ considers the variables `temporary` and `TEMPORARY` to be identical.

   (e) The modulus operator (`%`) can be used only with integer operands.

   (f) The arithmetic operators `*`, `/`, `%`, `+` and `-` all have the same level of precedence.

2. † Match the variables (left) with the data items (right).

(a)	`float f;`	(i)	`true`
(b)	`char c;`	(ii)	`4`
(c)	`bool b;`	(iii)	`3.85`
(d)	`int i;`	(iv)	`"hello"`
(e)	`char s[];`	(v)	`'a'`

For Questions 3 to 13, assume the following variable declarations:

```
int i, j, x, y;
bool b1, b2;
```

3. † Identify and correct the bugs in each of the following C++ statements. Explain how each statement would malfunction if the bugs were not corrected.

   (a)
   ```
   if (i > 3);
       cout << "i is greater than 3" << endl;
   ```

   (b)
   ```
   if (i =< 4)
       cout << "i is equal to or less than 4" << endl;
   ```

4. † What, if anything, is displayed on the screen when each of the following C++ statements is executed? Assume `i = 3` and `j = 4`.

   (a) `cout << i;`
   (b) `cout << i + i;`
   (c) `cout << "i = ";`
   (d) `cout << "i = " << i;`
   (e) `cout << endl;`
   (f) `cout << i + j << " = " << j + i;`
   (g) `x = i + j;`
   (h) `// cout << "i + j = " << i + j;`
   (i) `cin >> i >> j;`

5. † What is the value of `x` after each of the following statements is executed?

   (a) `x = 4 + 3 * 8 / 2 - 1;`

   (b) `x = 4 % 4 - 4 / 4 + 4 * 4;`

   (c) `x = (3 * 5 * (3 + (5 * 3 / (3))));`

6. † What is the value of `b2` after each of the following statements is executed? Assume `b1 = false` and `j = 9`.

   (a) `b2 = j > 9 && !b1;`

   (b) `b2 = b1 || j == 9;`

   (c) `b2 = (j != 5) && b1;`

   (d) `if (!b1) b2 = true; else b2 = false;`

7. † Write some C++ code to determine whether an integer variable `i` is odd or even and display the outcome on the screen.

8. † Determine the output for each of the following C++ code segments when `x = 14` and `y = 16` and when `x = 16` and `y = 14`. How would you rewrite the code segments to make their behaviour more readily apparent?

   (a)
   ```
   if (x < 15)
   if (y > 15)
   cout << "hello" << endl;
   else
   cout << "goodbye" << endl;
   cout << "world" << endl;
   ```

(b)
```
if (x < 15) {
   if (y > 15)
   cout << "hello" << endl; }
   else {
   cout << "goodbye" << endl;
   cout << "world" << endl; }
```

9. † The following C++ code should print the numbers 1 to 20. Identify and correct the bug in the code.

```
j = 1;
while (j < 20) {
   cout << j << endl;
   j = j + 1;
}
```

10. † What do the following two code segments print for the cases `x = 1` and `x = 10`?

```
while (x > 1)  {
   cout << x << ' ';
   x--;
}
```

```
do {
   cout << x << ' ';
   x--;
} while (x > 1);
```

11. † What does the following C++ code do if `x` and `y` are both positive? What happens if one or the other of `x` and `y` is zero or negative?

```
for (i = 1; i <= y; i++) {
   for (j = 1; j <= x; j++) cout << '*';
   cout << endl;
}
```

12. † Write three C++ code segments using `for` loops to display the following three patterns separately on the screen.

```
*
**
***
****
*****
******
*******
1
```

```
*******
******
*****
****
***
**
*
2
```

```
*******
*****
***
*

3
```

13. † The following code should print whether `b1` is `true` or `false`. Identify and correct the bug.

```
switch (b1) {
   case true:
      cout << "b1 is true" << endl;
   case false:
      cout << "b1 is false" << endl;
}
```

Questions 14 to 17 address the use of functions in C++, including the two parameter passing mechanisms call by value *and* call by reference. *If you have difficulty with these questions, refer to Sections 7–8 of the* Tutorial Guide to C++ Programming. *Question 16 introduces the concept of recursion.*

14. † Identify and correct the bugs in the following function definitions.

(a)
```
int sum(int a, int b)
{
   int result;
   result = a + b;
}
```

(b)
```
double square(double number)
{
   double number;
   return number * number;
}
```

(c)
```
void times_two(float number)
{
   float result;
   result = 2.0 * number;
   cout << "Two times " << number << " is " << result << endl;
   return result;
}
```

15. Consider the following C++ function definition.

```
int multiply(int n1, int n2)
{
   int result = 0;
   while (n1 > 0) {
      result = result + n2;
      n1--;
```

```
    }
    return result;
}
```

The function is called with the statement `c = multiply(a,b)`, where the integer variables `a` and `b` are initially set to 2 and 3 respectively. What are the contents of `a`, `b` and `c` immediately after the call? Would you change your answer if the first line of the definition were changed to `int multiply(int &n1, int &n2)`? When will the function fail to return the correct answer?

16. * Consider the following function definition.

```
int sum(int n)
{
   if (n == 0) return 0;
   else return n + sum(n-1);
}
```

The function is called with the statement `a = sum(3)`. Draw a diagram showing each subsequent call to the function and the corresponding `return` values. What will be the final value returned to the variable `a`? What would happen if the function were called with the statement `a = sum(-1)`?

17. † Consider the following C++ program.

```
int main()
{
   float l1, l2, angle;
   cout << "Enter two lengths and an angle." << endl;
   cin >> l1 >> l2 >> angle;
   cout << "Area is " << AreaTriangle(l1, l2, angle) << endl;
   return 0;
}

float AreaTriangle(float a, float b, float theta)
{
   return (0.5 * a * b * sin(theta));
}
```

The program will not compile without appropriate header files and function prototypes. Write down the extra lines that must be inserted at the top of the program, and explain their purpose.

Questions 18 to 21 address the use of one- and two-dimensional arrays in C++. If you have difficulty with these questions, refer to Sections 9 and 11 of the Tutorial Guide to C++ Programming.

18. Find the bugs in the following C++ function which performs vector addition.

```
void vector_add(float a[], float b[], float &c[], int n)
   // Adds the n-element vectors a and b, storing the result in c.
{
   for (i = 1; i <= n; i++) c[i] = a[i] + b[i];
   return c;
}
```

Which of the bugs would you expect to be picked up by the compiler?

19. Write a function `vector_mean` that takes two parameters, an array of integers `a` and the length of the array `n`, and returns the mean value of the elements in the array.

20. † Write some code to declare a 3×3 matrix `m1` and initialise it to represent a rotation by an angle θ around the z-axis.

21. The following C++ code segment uses the function `vector_mean` from Question 19 to calculate the average of the elements in the integer matrix `m2`. How many rows and columns must `m2` have?

```
total = 0.0;
for (i = 0; i < n; i++) total = total + vector_mean(m2[i], r);
result = total / n;
```

Questions 22 to 25 address the use of data structures in C++. If you have difficulty with these questions, refer to Section 11 of the Tutorial Guide to C++ Programming.

22. † Consider the following structure definition.

```
struct Date {
   int year;
   int month; // in the range 1 to 12
   int day;   // valid range depends on the month
};
```

Write some code to declare a variable `today` of type `Date` and set it to today's date.

23. * Given the definition of the `Date` structure in Question 22, the following function is supposed to check whether a date is valid. To simplify the exercise, we assume that leap years never happen. The function should return `true` for a valid date and `false` for an invalid date, and reset any invalid date to 1 January 2000. Identify and correct the two bugs in the function definition.

```
bool valid_date(Date today)
{
  bool valid = true;
  const int days[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

  if (today.month < 1 || today.month > 12)
    valid = false;
  else
    if (today.day < 1 || today.day > days[today.month]) valid = false;
  if (!valid) {
    today.year = 2000; today.day = 1; today.month = 1;
  }
  return valid;
}
```

24. * Given the definition of the `Date` structure in Question 22, write a function to work out tomorrow's date given today's date. The function should have the header

    ```
    bool tomorrow(Date today, Date &tomorrow)
    ```

    and should return `true` if `today` is valid and `false` otherwise. Your function may call the (de-bugged) function `valid_date`, which is defined in Question 23.

25. * Write down a definition for a `Time` data type, with separate fields for the hours, minutes and seconds. Use this new data type to extend the `Date` structure in Question 22 to include the time as well as the date. Explain why any high level code that calls the functions `valid_date` and `tomorrow` (see Questions 23 and 24) is not affected by this extension. How would you extend the `valid_date` function to check the validity of the `Time` as well? Discuss the importance of data abstraction for software maintenance.

Suitable past Tripos questions: 2004 Qu. 10, 2006 Qu. 12, 2008 Qu. 11

Hints

2. Remember that single characters are enclosed inside single quotation marks, whereas character arrays (strings) have double quotation marks.

3. (a) Look for a rogue semicolon. (b) Look for an illegal relational operator.

4. Anything inside quotation marks is displayed verbatim. Expressions outside quotation marks are evaluated and then displayed.

5. In the absence of parentheses, `*`, `/` and `%` take precedence over `+` and `-`.

6. Remember that `&&` is the logical AND operator, `||` is the logical OR operator, and `!` is the logical NOT operator. The precedence rules are that relational operators like `>` and `==` are evaluated before logical operators like `&&` and `||`.

7. The modulus operator is very useful here, since `x % 2` gives different answers for odd and even `x`.

8. Remember that only one statement is executed conditionally in an `if` construction, *unless* multiple statements are grouped together using curly brackets. Remember also that constructions like `if (b) s1; else s2;` count as one statement.

9. Look carefully at the termination condition of the `while` loop: what is the last number the loop will print?

10. Remember that a `do` loop always executes at least once, since the termination condition is checked at the end of the loop. `x--` means "decrement `x` by one".

11. Bear in mind that a `for` loop may not execute even once if its termination condition is met immediately: this might happen if one or the other of `x` and `y` is zero or negative.

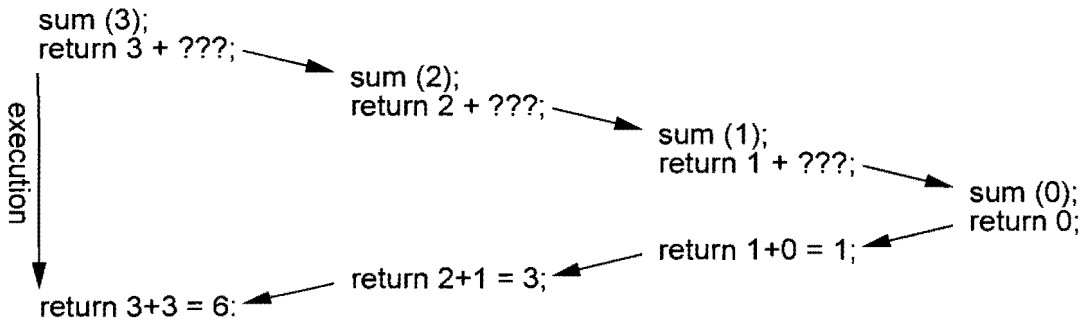
12. Use two loops, with one nested inside the other. Make the outer loop count up for the first pattern and down for the other two. For the third pattern, you'll need to decrement the outer loop counter by two at the end of each iteration.

13. Look for a missing `break` statement.

14. (a) An `int` function should return something. (b) A formal parameter should not be declared as a local variable. (c) A `void` function should not return anything.

15. With *call by value* (no `&`'s in the header), the function makes a private copy of the actual parameters before operating on them. With *call by reference* (`&`'s in the header), the function operates directly on the actual parameters.

16. This is an example of a *recursive* function, ie. one that calls itself. A recursive function is used to solve a problem, in this case calculating $\sum_{i=0}^{n} i$ for $n \geq 0$. The function actually knows how to solve only the simplest case with $n = 0$. To solve harder cases, the function breaks the problem down into two pieces: $\sum_{i=0}^{n} i \equiv n + \sum_{i=0}^{n-1} i$. For recursion to work, the second piece must be a slightly smaller version of the original problem, which indeed it is. So the function calls itself to solve the smaller problem, launching a new version of `sum` while the original call is still open. One call follows another, with no returns, until the `sum(0)` call, which returns `0` without needing to make further calls: this is the simple case we know the answer to. All the other returns now happen, in the reverse order to the calls.



17. The header files, namespace declaration and function prototypes are missing.

18. Check the range of the array indices inside the `for` loop. Are all local variables properly declared? Remember that arrays are *always* passed as reference to functions, with no need for the `&` notation. A `void` function should not return anything.

19. Bear in mind that the mean of a set of integers may not be an integer.

20. A matrix is a two-dimensional array. Individual elements are accessed using two indices: eg. `m1[0][1] = -sin(theta)`.

21. `m2[i]` is the `i`th row of the matrix `m2`. Remember that a row of a matrix is the same as an array.

22. The individual fields of a structure are accessed using the "dot" notation: eg. to set the year to 2001, we'd use the statement `today.year = 2001`.

23. The line beginning `const int days [12]` initialises the array with the twelve numbers inside the curly brackets. Bear in mind that if a function needs to change a parameter, then that parameter must be passed *by reference*. Check the index of the `days` array.

24. The function should immediately return `false` if the date `today` is not valid. Then add one to the day and check it hasn't overshot the end of the month: if so, reset the day to 1 and add one to the month. Then check that the month hasn't overshot the end of the year: if so, reset the month to 1 and add one to the year. You could recycle the `days` array from Question 23 to store the number of days in each month.

25. Try to ensure that the extended `valid_date` function does not know what's inside the `Time` structure, so that the function would not need modifying were we to ever change or extend the definition of `Time`. This sort of careful data abstraction localises the use of data types to particular functions, minimising the impact of any future changes.

Answers

1. (a) False. (b) True. (c) True. (d) False. (e) True. (f) False.

2. (a) (iii); (b) (v); (c) (i); (d) (ii); (e) (iv).

3. (a) Remove the semicolon at the end of the first line. (b) Replace `=<` with `<=`.

4. (a) `3` (b) `6`
   (c) `i =` (d) `i = 3`
   (e) The cursor moves to the next line. (f) `7 = 7`
   (g) Nothing is displayed. (h) Nothing is displayed.
   (i) Whatever the user types is displayed.

5. (a) `15`. (b) `15`. (c) `120`.

6. (a) `false`. (b) `true`. (c) `false`. (d) `true`.

8. For `x = 14` and `y = 16`: (a) `hello world` (b) `hello`

For `x = 16` and `y = 14`: (a) `world` (b) `goodbye` `world`

9. Replace `while (j < 20)` with `while (j <= 20)`.

10. For `x = 1`, the left loop prints nothing, the right loop prints `1`.
For `x = 10`, both loops print `10 9 8 7 6 5 4 3 2`.

11. For `x` and `y` both positive, prints a `y` × `x` rectangle of asterisks.
For `x` non-positive, moves the cursor down `y` lines. For `y` non-positive, does nothing.

13. Insert the line `break;` immediately before the line `case false:`.

14. (a) Add the line `return result;` immediately before the closing brace.
(b) Delete the line `double number;`.
(c) Either delete the line `return result;`
or change the header to `float times_two(float number)`.

15. `a = 2, b = 3, c = 6`. `a = 0, b = 3, c = 6`. When `a` is negative.

16. `sum(3)` returns `6`. `sum(-1)` crashes the program.

17. The missing lines are:

```
#include <iostream>
#include <cmath>
using namespace std;
float AreaTriangle(float a, float b, float theta);
```

18. The corrected function is:

```
void vector_add(float a[], float b[], float c[], int n)
   // Adds the n-element vectors a and b, storing the result in c.
{
   int i;
   for (i = 0; i < n; i++) c[i] = a[i] + b[i];
}
```

21. `m2` must have `n` rows and `r` columns.

23. Two lines need to be corrected as follows:

```
bool valid_date(Date &today)
if (today.day < 1 || today.day > days[today.month-1]) valid = false;
```

Gabor Csanyi

Michaelmas 2013