UNIVERSITY OF EDINBURGH

FACULTY OF SCIENCE AND ENGINEERING

COMPUTER PROGRAMMING SKILLS AND CONCEPTS 1h

Tuesday 17 September 2002,
2.00pm – 3.30pm

Examiners
D.K. Arvind (Chair), R. Dyckhoff (External).

INSTRUCTIONS TO CANDIDATES

1. Check that the question paper contains pages 1–12. If it does not, inform the invigilator.
2. You should attempt as many questions as possible.
3. The questions in this paper vary widely in difficulty. During your first pass through the paper, you are advised not to dwell on questions to which the answer is not readily apparent.
4. Write your answer immediately below each question. Where a box or table is provided, write your answer within it. If you wish to write more in answer to a question, continue on the blank page opposite, indicating that you have done so.
5. Please write *legible* and *concise* answers.
6. The marks allocated to each part of a question are indicated in the margin. There are 100 marks in total.
7. If you are in the third or later year of your degree course write 3 here. $\boxed{}$

1. Consider the following `if`-statement, where x,y,z are of type float.

```
if ( (x / 2.0) < (y - z) ) {
    y = x + 1.0;
    z = y + 1.0;
  }
  else {
    if ( z > y ) {
      x++;
      y++;
    }
    if ( z++ < 0 ){
      y++;
      z++;
    }
  }
```

Fill in the values that the variables `x,y,z` would have after executing the statement above, assuming their initial values are as given. *(10 marks)*

Initial x	Initial y	Initial z	Final x	Final y	Final z
-2.0	1.5	2.0			
0.0	1.5	2.0			
0.0	-1.5	2.0			
10.0	1.5	-3.0			
1	2	3			

2. (a) Below is a template for character-by-character processing of an input file. Complete the program by filling in the boxes so that the program counts the number of occurrences of the letter `e` (both upper and lower case) in the input and then prints the value out at the end.

```
int main(void) {
    int c, count;
```


(1 mark)

```
    while ((c = getchar()) != EOF) {
```


(3 marks)

```
    }
```


(2 marks)

```
    return EXIT_SUCCESS;
}
```

(b) Below is a template for character-by-character processing of an input file. Complete the program by filling in the boxes so that the program works out the ratio of upper case letters to lower case letters in the input as a fraction and then prints the value out at the end. [*Recall that the functions* `isupper` *and* `islower` *test whether a character is upper case or lower case, respectively.*]

```
int main(void) {
    int c, countu = 0, countl = 0;

    while ((c = getchar()) != EOF) {
```


(2 marks)

```
    }
```


(2 marks)

```
    return EXIT_SUCCESS;
}
```

3. Consider the following program fragment.

```
int m, n, count;

 m = 28;
 count = 0;

 for( n = m; n != 0; n-- ){
   if ( m % n == 0 ) { count++ ; }
 }

 printf( "count=%d\n", count);

 return EXIT_SUCCESS;
```

(a) What is the output when this code is executed?

(3 marks)

(b) Supply an initial value for `m` that will cause the `for` loop to fail to terminate.

(2 marks)

(c) Suppose the statement `n != 0` is replaced with `n > 0` in this loop. Does your answer to part (3b) still cause the loop to fail to terminate?

(2 marks)

(d) Explain why the two versions of the code will produce the same observable behaviour for any *positive* initial value of `m`.

(3 marks)

4. Write the output of the following program in the box underneath it.

```
void function1(int i, int j) {
   printf("i = %d, j = %d\n", i, j);
}

void function2(int i, int *j) {
   *j = i * 2;
   i = *j * 2;
   printf("i = %d, *j = %d\n", i, *j);
}

int *function3(int *i) {
  *i = 99;
  return i;
}

int main(void) {
   int a, b, *c;

   a = 1;
   b = 2;
   function1(a++, ++b);
   printf("a = %d, b = %d\n", a, b);

   a = 3;
   b = 4;
   function2(a, &b);
   printf("a = %d, b = %d\n", a, b);

   a = 5;
   c = function3(&a);
   b = *c;
   printf("a = %d, b = %d\n", a, b);
}
```

(10 marks)

5. For each code fragment below, write a tick in the left box if the code terminates, and a cross otherwise. In the cases that terminate, write the final value of `n` in the centre box, and of `m` in the right-hand box. In all cases assume a declaration `int n, m;` is given.

(a)
```
n = 10; m = 0;
while ( !(n < m)) {
    ++n;
}
```

terminates? ☐ value of `n`? ☐ value of `m`? ☐ *(2 marks)*

(b)
```
n = 5; m = 22;
while (n < m) {
    n++;
    m =- n;
}
```

terminates? ☐ value of `n`? ☐ value of `m`? ☐ *(2 marks)*

(c)
```
n = -5; m = 2;
while (n < m) {
    m = m * n;
}
```

terminates? ☐ value of `n`? ☐ value of `m`? ☐ *(2 marks)*

(d)
```
n = 10; m = 1;
while (n != m) {
    m = n * m;
    n = 10 * m;
}
```

terminates? ☐ value of `n`? ☐ value of `m`? ☐ *(2 marks)*

(e)
```
n = 1000; m = -500;
while (n != m) {
    m--;
    n++;
}
```

terminates? ☐ value of `n`? ☐ value of `m`? ☐ *(2 marks)*

6. Write the output of the following program in the box underneath it. [*Recall that the function* `malloc(n)` *allocates* `n` *bytes of memory from the heap and returns a pointer to the first byte.*]

```
#define N 6

int main(void) {
   char name1[N], *name2;

   strcpy(name1, "Tony");
   name2 = name1;
   printf("name1 = %s, name2 = %s\n", name1, name2);

   strcpy(name1, "Blair");
   printf("name1 = %s, name2 = %s\n", name1, name2);

   name1[1] = '\0';
   name2 = malloc(2);
   strcpy(name2, "T");
   printf("name1 = %s, name2 = %s\n", name1, name2);

   printf("name1 = %d, name2 = %d\n", strlen(name1), strlen(name2));

   if (!strcmp(name1, name2)) printf("Yes\n"); else printf("No\n");
}
```

(10 marks)

7. This question concerns the use of a data-type recording infomation about the state of a game played on a 3×3 board. Each square is marked as one of **B, W**, indicating black or white. The state of the game is given by a marked board.

   Some initial code is provided as shown.

```
#define SIZE  3    /* size of the board */

typedef enum {B, W} Square_t;

typedef Square_t Board_t[SIZE][SIZE];
                      /* Representation of the board */

/* global variable  */
Board_t  theGame;
```

   (a) Write a procedure to initialise a game by marking the squares alternately as black and white.

```
int initGame( void )
{
```

   (5 marks)

```
}
```

P.T.O.

(b) Suppose you are given a procedure to print out information about a square as follows.

```
void showSquare( Square_t s )
{
  switch (s) {
  case B:
    printf( "B " );
    break;
  case W:
    printf( "W " );
}
```

Write a procedure that will take the current state of the game, held in the global variable `theGame`, and display the board by printing something like

```
B W B
W B W
B W B
```

```
void displayBoard( void )
{
```

(5 marks)

```
}
```

8. Give the type and value of each of the following C expressions, e.g. the term `1+2` has type `int` and value `3`. Put a cross in the type box and leave the value box blank if the expression is invalid. (On the final page you will find a table that gives the precedence and associativity of the various operators in C.)

(a) `5/2`

type ☐ value ☐ *(1 mark)*

(b) `5%2`

type ☐ value ☐ *(1 mark)*

(c) `5.0/2.0`

type ☐ value ☐ *(1 mark)*

(d) `5%2.0`

type ☐ value ☐ *(1 mark)*

(e) `2*3/4*5`

type ☐ value ☐ *(1 mark)*

(f) `'A'-'C'`

type ☐ value ☐ *(1 mark)*

(g) `'A-C'`

type ☐ value ☐ *(1 mark)*

(h) `"A-C"`

type ☐ value ☐ *(1 mark)*

(i) `*"A-C"`

type ☐ value ☐ *(1 mark)*

(j) `4 <= 3 || 2 >= 1`

type ☐ value ☐ *(1 mark)*

9. (a) Write the output of the following program in the box provided.

```
void prime_factors(int number) {
  int factor;

  printf("%d : ", number);
  for (factor=2; factor*factor <= number; factor++) {
    if (number % factor == 0) {
      printf("%d\n", factor);
      prime_factors(number/factor);
      return;
    }
  }
  printf("%d\n", number);
}

int main() {
  prime_factors(210);
}
```

(8 marks)

(b) Explain briefly why the recursion in the above function is guaranteed to terminate.

(2 marks)

10. Consider a linear search procedure, which searches an integer array for a given key.

```
int LinearSearch(int n, int a[], int searchKey)
/*  Returns 1 if and only if searchKey is contained
 *  in the array, i.e., there exists an index i with 0 <= i < n
 *  such that a[i] == searchKey.
 */
{
   int i;

   for (i = 0;  i < n;  ++i) {
      if (a[i] == searchKey) return 1;
   }
   return 0;
}
```

(a) Suppose `n` = 8, and the array `a` contains the values

```
a[0],a[1],...  = 0, 23, 3, 7, 12, 9, 13, 17
```

Write down the number of comparison tests made by the procedure for the given values of `searchKey`.

`searchKey`	no. of comparisons
3	
9	
10	

(2 marks)

(b) Suppose the values in the array are *sorted* in increasing order. Will the procedure in general be more efficient for such data?

(2 marks)

(c) Name a different search procedure that is more efficient on sorted data than any linear search procedure.

(2 marks)

(d) Explain why the function `LinearSearch()` terminates for any initial values of its parameters.

(4 marks)

Appendix: Precedence and Associativities of some C Operators

Precedence decreases line by line from the top to the bottom of the table. Operators on the same line have the same precedence.

<table>
<tr><th>Operators</th><th>Associativity</th></tr>
<tr><td>! + (unary) - (unary)</td><td>right to left</td></tr>
<tr><td>* / %</td><td>left to right</td></tr>
<tr><td>+ (binary) - (binary)</td><td>left to right</td></tr>
<tr><td>< <= > >=</td><td>left to right</td></tr>
<tr><td>== !=</td><td>left to right</td></tr>
<tr><td>&&</td><td>left to right</td></tr>
<tr><td>||</td><td>left to right</td></tr>
<tr><td>= += -= *= /= %=</td><td>right to left</td></tr>
</table>