MA U11601 Quiz 04 26/11/21 ANSWERS

Answer any 3 questions. Submit them through Blackboard as pdfs, either handwritten and scanned, or typeset. They should be submitted before midnight on Monday, 29 November. If more than three answers are submitted, only the first three will be marked. All questions carry 20 marks.

Show all work. That is, where an answer requires some calculation, show the calculation.

Plagiarism. If copying is detected, all those involved will lose credit, irrespective of who copied from whom.

Question 1. By analysing the operator precedence, explain

```
% cat xxx.c
#include <stdio.h>
int main()
{  char x[] = "hello.................",
   y[] = "goodbye";
   printf("%s\n", x);
   char *a = x, *b=y; // a++ increments the address in a by 1
   while (  *a++ = *b++ );
   printf("%s\n", x);
}
% gcc xxx.c
% a.out
hello.................
goodbye
%
```

Answer. It is the while-loop which matters, of course. Referring to the precedence rules, we can add some parentheses, noting that postfix ++ has higher precedence than prefix *:

```
  while ( *(a++) = *(b++) )
  = has lowest rank
  while ( (*(a++)) = (*(b++)) )

  The {\em value} of b++ is its value before incrementing,
  and likewise for a, so the assignment is the same as
     *a = *b
  except that at the next iteration, the addresses in a and b
  will have been incremented.

  The while loop repeatedly copies from *b to *a.
  The {\em value} of the assignment is the value assigned,
  so when b reaches the null character in "goodbye",
  the loop ends. (Note that x is longer than y, so there
  is no array overflow).
```

Question 2. Write a *recursive* routine `printx(int n)`, assuming $n > 0$, prints the (face-value) hex encoding of n, one hex digit at a time. (Newlines should be *omitted*).

Hint: `char hex[]="0123456789abcdef";`

Answer.

```
#include <stdio.h>

void printx( int n )
{ char hex[] = "0123456789abcdef";
  if ( n>0 )
  { printx( n/16 );
    printf("%c", hex[ n%16 ]);
  }
}            // only printx() is required

int main ( int argc, char * argv[] )
{ int n = atoi ( argv[1] );
  printf("n %d hex ",n);
  printx(n);
  printf("\n");
}
```

Question 3. (i) Simulate `xxx(30)`, where `xxx()` is the following non-recursive function. Remember the 'staggered' layout to show the order in which variables are changed (Section 12.10). (ii) What does `xxx(n)` return, for general $n \geq 1$? (iii) Give an invariant for the while-loop.

```
int xxx(int n)
{
  int x,y,z,m;
  m = 0; x = 1; y = 2;
  while ( y <= n )
  { z = 5*y-6*x;
    x = y; y = z;
    ++m;
  }
  return m;
}
```

Answer.

```
     n    x    y    z    m   y<=n
    30
          1
               2
                    -
                         0
```

```
                             yes
                    4
          2
               4
                         1
                             yes
                    8
          4
               8
                         2
                             yes
                   16
          8
              16
                         3
                             yes
                   32
         16
              32
                         4
                             no
returns 4
```

(ii) In general, returns $\lfloor \log_2(n) \rfloor$. (iii) A reasonable loop invariant: $x = 2^m$ at the m-th iteration and $y = 2^{m+1}$.

Question 4. Disambiguate the following by fully parenthesising, and evaluate. For example,

```
1 - 2*(3.0+4/5) - 6  =
(1 - (2*(3.0+(4/5)))) - 6 =
(1 - 2*(3.0+0))-6 = (1-6.0) -6 = -11.0
------------------------------------------

(i)  1 - 2/3 + 4.0                 (ii) 1 - 2/3.0 + 4
(iii) 3*4/2                        (iv)  3/2*4
(v) 1 - ( 2 - 3 - 4.0 )
```

Answer.

```
(i) (1 - (2/3)) + 4.0 = 5.000000
(ii) (1-(2/3.0))+4 = 1 - .6666667 + 4 = 4.333333
(iii) (3*4)/2 = 6
(iv) (3/2)*4 = 1 * 4 = 4
(v) 1 - ( 2-3-4.0) = 1 - ((2-3)-4.0) = 1 - (-1 -4.0) = 6.000000
```

Question 5. (i) Carefully simulate `xxx(1215,2)`, showing the stack frames associated with each call to `xxx()`, and using indentation to identify them clearly (Section 18):

```
int xxx ( int n, int p )
{
  int q;
  if ( n%p == 0 )
    return p;
  else
  { q = xxx (n,p+1);
    return q;
  }
}
```

(ii) What does `xxx(n,2)` return, given general $n \geq 2$?

Answer.

```
   n p  q   n p  q   n p  q   n p  q
                                        xxx(875,2)

 875 2 --
                                          xxx(875,3)

          875 3 --
                                            xxx(875,4)

                   875 4 --
                                              xxx(875,5)

                            875 5 --

                                              return 5
                                            xxx(875,4) resume
                   875 4  5                 return 5
                                          xxx(875,3) resume
          875 3  5                        return 5
                                        xxx(875,2) resume
 875 2 5                                return 5
```

In general, it returns the smallest prime divisor of n.

Question 6. Identify five mistakes in the program below. Some mistakes may be errors which block compilation, others may raise warnings, and others may be compiled but not work properly.

```
#include <stdio.h>
void xx(int m[4])
{
  int s =0;
  for (i=0; i<4; ++i); // i undeclared
                       // semicolon kills for loop
  { s+=m[i]};          // displaced semicolon
  return s;            // void: cannot return anything
}
main()
```

```
{
  char y[16] = "hello";
  for(i=0; i<16; ++i)
    printf(" %d", y[i]);
  printf("%d %d\n",strlen(y), xx(y));
                      // missing string.h
                      // xx(y) void, meaningless
}
```