Université d'Ottawa
Faculté de genie

École d'ingénierie et de
technologies de l'information

uOttawa

L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Information
Technology and Engineering

GNG1106 **Midterm Examination** **25 February 2008**

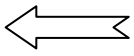
Time allowed: 80 minutes
Closed book examination
Non-programmable calculators are allowed

Attempt all questions
Questions carry the weights indicated
The total number of points for the examination is 70

Answer the questions in the spaces provided

Surname:	**SOLUTION**	**Student number**	
First name:			

Part 1:	
Part 2:	
Part 3:	
Part 4:	
Total:	



Do not write in this box!

Part 1 – Short answer questions *(16 marks)*

4 marks each

Please provide short answers to each question.

(a) Explain the difference between a variable and a symbolic constant from the perspective of program compilation and execution, program memory and working memory.

Answer:

A variable is a location in memory that has an address and which contains a value. An address in working memory gets allocated when the execution of program working memory reaches the variable declaration statement. A value gets stored in this location when variable assignment is done during program execution. A symbolic constant name is textually substituted before compilation (during pre-processing) by the symbolic constant quantity everywhere it appears in the code. This quantity only appears in program memory, because it is part of the compiled program.

(b) Consider the following code fragment:

```
int a = 3, c;
float y = 2.5;
…
c = a + y;
printf("The value of c is %d.", c);
```

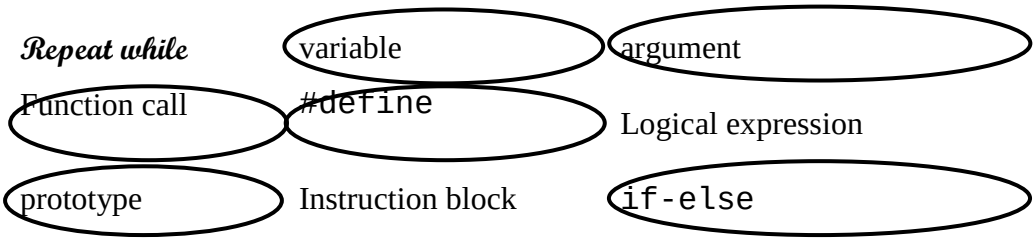
Explain precisely how the arithmetic expression is evaluated by the C compiler and provide the screen output.

Answer:

The integer variable `a` is implicitly promoted to type float, because `y` is of type float. The float summation of `a` and `y` is performed, yielding `5.50000`, and the result is assigned to the integer variable `c,` yielding `5`. A loss of information occurs.

The screen output is… `The value of c is 5.`

(c) Circle any of the following terms that do NOT relate directly to the looping structure:



Consider the following function. In the space below, write down its header.

```
int larComFac(int top, int bottom)
{
   int ctr=top;
   /* Loop until the LCF is found. */
   while( ((top%ctr != 0) || (bottom%ctr != 0)) && ctr > 1)
      ctr--;
   return ctr;
}
```

Answer:

```
int larComFac(int top, int bottom)
```

Part 2 – Code evaluation & logic *(18 marks)*

9 marks each

(a) Consider the code fragment shown below. Assume that the values 4 7 2 were typed in response to the screen prompt, followed by the enter key. What is the code output? That is, what will be printed on the screen?

```
int x, y, z;
...
printf("Enter three integers:");
scanf("%d%d%d", &x, &y, &z);
printf("You entered the values %d, %d and %d.\n", x, y, z);
if (x>y)
   if (y<z)
     printf("%d > %d and %d < %d\n", x, y, y, z);
else
   printf("We find %d < %d.\n", x, y);
```

Answer:

```
Enter three integers: 4 7 2
You entered the values 4, 7 and 2.
```

(b) A C compiler random number generator, using a seed value of unity, produces the following sequence of pseudorandom numbers: 36, 634, 56, 76, 23, 987, 235, 3, 65, 6, 23454, 34… What is the screen output of the C program shown below?

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
   int i, j, a, b;
   float x, y;

   for (i=0; i <= 3; i++)
   {
      x = (float)rand();
      printf("  %f", x);
   }
   printf("\n");
   srand(1);
   for (j=0; j <= 2; j++)
   {
      y = (float)rand();
      printf("  %f", y);
   }
   printf("\n");
   srand(1);
   a=rand();
   b=rand();
   printf("The next two random numbers are %d and %d.\n", a,b);
   system("pause");
}
```

Answer:

```
  36.000000  634.000000  56.000000  76.000000
  36.000000  634.000000  56.000000
The next two random numbers are 36 and 634.
```

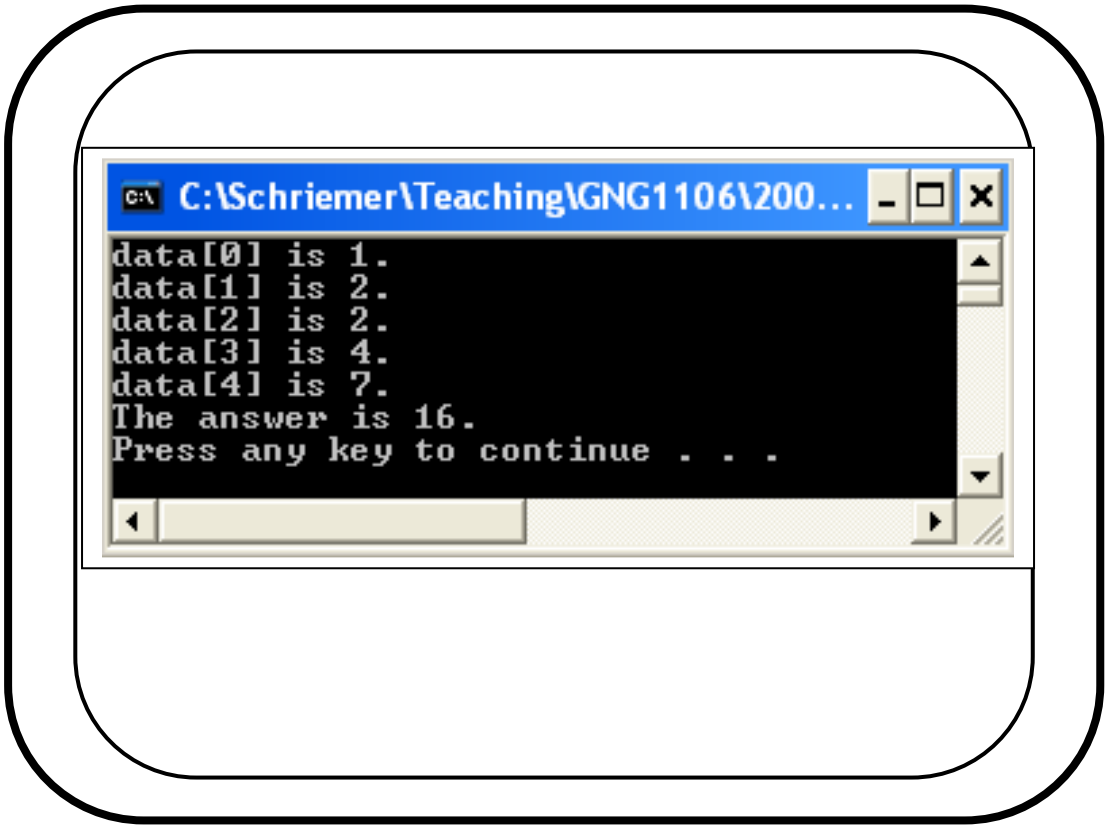
Part 3 – The programming model *(16 marks)*

For the program on the following page, show how the contents of the working memory are changed by the execution of the program. Values are to be recorded within the boxes (which represent assigned memory. These boxes are labeled according to the variable name with which they are associated. Addresses of the variables are given within brackets, where possibly needed. Record successive assignments to a variable as follows:

Variable name	~~2~~, ~~6~~, ~~4~~, 10

That is, new data values "overwrite" old values. If "assigned memory" boxes are missing, add them as necessary, labeling each with its variable name.

Finally, on the terminal screen below, show the output of the program, i.e., what is printed on the screen.



Program Memory

```
#include <stdio.h>
#define POINTS 5

int correct(int [], int);

void main()
{

int temp[POINTS], answer, ctr;

temp[0] = 1;
temp[1] = 3;
for(ctr=2; ctr <= POINTS-1; ctr++){
   temp[ctr] = temp[ctr-1]+ temp[ctr-2];}
answer = correct(temp, POINTS);
printf("The answer is %d.\n", answer);
system("pause");
}


int correct(int data[],int max)
{
   int i, sum;
   sum = 0;
   for(i=0; i<=max-1; i++)
      {
      data[i]=data[i]-i;
      printf("data[%d] is %d.\n",i,data[i]);
      sum = sum + data[i];
      }
return sum;
}
```

Working Memory

temp

(10226)	~~1~~, 1
(10230)	~~3~~, 2
(10234)	~~4~~, 2
(10238)	~~7~~, 4
(10242)	~~11~~, 7

ctr	~~2~~, ~~3~~, ~~4~~, 5
answer	16

data	10226
max	5
i	~~0~~, ~~1~~, ~~2~~, ~~3~~, ~~4~~, 5
sum	~~0~~, ~~1~~, ~~3~~, ~~5~~, ~~9~~, 16

CPU

Part 4 – Problem Solving *(20 marks)*

You are to complete a portion of a software program that can find the real roots of polynomials up to degree 3:

$$f(x) = a_0x^3 + a_1x^2 + a_2x + a_3.$$

The user shall provide the coefficients of the polynomial and the start/end of the search interval.
The program shall find the real roots in the interval and prints them on the screen. Pseudocode for main and the two relevant subprograms is given below. You are to provide ONLY the C code for the main program, NOT the subprograms (these are for reference only). You code should include all necessary statements, including function prototypes and comments.

```
main
    Read values into a0, a1, a2, a3
    Read values into begin, end
    Read value into subinter
    Assign (end-begin)/subinter to n
    Assign 0 to ctr
    Repeat while ctr is not equal to n
        Assign start+(ctr*subinter) to ak
        Assign start+(ctr+1)*subinter to bk
        if bk is larger than end
            Assign end to bk
        find_root(ak, bk, a0, a1, a2, a3)
        Increment ctr
    find_root(end, end, a0, a1, a2, a3)

find_root(left, right, a0, a1, a2, a3)
    Assign poly(left, a0, a1, a2, a3) to func_left
    Assign poly(right, a0, a1, a2, a3) to func_right
    If func_left X func_right is less than 0.0001 and larger than -0.0001
        if func_left is smaller than 0.0001 and larger than -0.0001
            Print "A root exists at ", func_left
    Otherwise
        if func_left X func_right is smaller than 0
            Print "A root exists at", (right+left)/2

poly(x, a0, a1, a2, a3)
    Assign a0*x*x*x to fx
    Assign fx+(a1*x*x) to fx
    Assign fx+(a2*x)+a3 to fx
    Return fx
```

Answer:

```
/************************************************************
File: poly3root.c
Coder: Henry Schriemer, 123456
Date: 25 February 2008
This program determines the real roots of polynomials up to
third order, given the input coefficients and scan interval.
************************************************************/

#include <stdio.h>
#include <math.h>

/*Function prototypes*/

void find_root(float, float, int, int, int, int);
float poly(float, int, int, int, int);

void main()
{
 /*Variable declaration*/
 int a0, a1, a2, a3, n, ctr;
 float begin, end, subinter, ak, bk;
 /*Parameter entry by user.*/
 printf("Enter the four polynomial coefficients:");
 scanf("%d%d%d%d", &a0, &a1, &a2, &a3);
 printf("Enter the scan interval begin and end points:");
 scanf("%f%f", &begin, &end);
 printf("What subinterval size would you like to use?");
 scanf("%f", &subinter);

 /*Number of subintervals to search for roots.*/
 n=ceil((end-begin)/subinter);

 /*Check for root in each subinterval, sequentially*/
 ctr=0;
 while(ctr!=n)
   {
   ak=begin+(ctr*subinter);
   bk=begin+(ctr+1)*subinter;
   if(bk>end){
     bk=end;}
   /*Function call to check for root in particular subinterval*/
   find_root(ak, bk, a0, a1, a2, a3);
   ctr++;
   }
 /*Check to see if the end of the scan interval is a root.*/
 find_root(end, end, a0, a1, a2, a3);
 system("pause");
}
```

Extra page…