Université d'Ottawa
Faculté de génie

École d'ingénierie et de
technologie de l'information

uOttawa

L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Information
Technology and Engineering

GNG1506/1106 **Midterm Examination Solution** **October 22, 2010**

Mohamad Eid and Fadi Malek

Time allowed: 90 minutes
Closed book examination
Non-programmable calculators are allowed

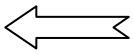
Attempt all questions
Questions carry the weights indicated
The total number of points for the examination is 100

Answer the questions in the spaces provided
Use both sides of these sheets if necessary

Les réponses en français sont acceptées.

Name: _____ **Student Number:** _____

Part 1:	35
Part 2:	35
Part 3:	30
Total:	100



Do not write in this box!

Part 1 - Tracing *(35 points)*

Write down the screen output generated by the following program:

A) (13 points)

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

#define SIZE 5
#define CONST 273.2

int main() {
int a = 2, b, c=5, i;
float y[SIZE]={1.0, -2.0, 5, 6.0};
float x, z;

a -= 1;
printf("a) a = %d\n",a);

b=a+CONST;
printf("b) b = %d\n",b);

y[a]=y[a-1];
printf("c) y[a] = %f\n", y[a]);

i=(a==SIZE);
printf("d) i = %d\n",i);

x = SIZE%(a+1);
printf("e) x = %f\n",x);

z = 2*(9-4)/2;
printf("f) z = %f\n",z);

if (!(y[1] <= 1 && y[0] >= -2.0))
   printf("The condition is valid\n");

return(0);
}
```

Output Screen

a) a = 1
b) b = 274
c) y[a] = 1.00
d) i = 0
e) X = 1.00
f) Z = 5.00

B) (6 points)

```
int x = 1, y = 0;

if(x < 0)
   printf("Choice 1");
else if (x != 0 && y == 0)
     printf("Choice 2");
else
     printf("Choice 3");
```

Output Screen

Choice 2

C) (8 points)

```
  int counter = 1;

  while (counter < = 10)
  {
   if(counter%3 == 0)
    {
     counter ++;
     printf("%d\n", counter);
    }
     counter ++;
  }
```

Output Screen

4
7
10

D) (8 points)

```
     int i, j;

     for(i=0;i<3;i=i+1)
     {
          for(j=0;j<2;j=j+1)
              printf("%d", i+j);

          printf("\n");
     }
```

Output Screen

01
12
23

Part 2 - Debugging *(35 points)*

A) (17 points)

This program is supposed to check whether a number is prime or not. Find five errors (syntax and/or Logical) in the program. Find the program line numbers where these errors occur and explain them.

```
1.      #include <stdio.h>
2.      #include <stdbool.h>
3.
4.      bool isPrime (int num);
5.
6.      int main() {
7.        Int aNum;
8.        bool result;
9.
10.       printf("Enter positive number to check if prime\n");
11.       scanf("%d", aNum);
12.
13.       result = isPrime(aNum);
14.
15.       if(result == true)
16.           printf("The number %d is prime \n", aNum);
17.       else
18.           printf("The number %d is not prime \n", aNum);
19.
20.       return 0;
21.     }
22.
23.     bool isPrime (int N)
24.     {
25.       bool  result = true;
26.       int i;
27.
28.       for (i=2; i<N; i++)
29.          if(i%N == 0)
30.              result = false;
31.       Return (result);
32.     }
```

a) Line number:

_____

_____

b) Line number:

c) Line number:

d) Line number:

e) Line number:

B) (9 points)

Find three errors in this program.

```
1.  int choice = -1, num1, num2, trials = 0;
2.
3.  // give the user a max of five trials
4.  while (choice != -1 && trials < 5)
5.    {
6.         printf("Enter a number: ");
7.         scanf("%d", &num1);
8.         printf("Enter another number: ");
9.         scanf("%d", &num2);
10.        printf("Their sum is %d\n" , (num1+num2));
11.        printf("Enter -1 to repeat, 1 to exit");
12.        scanf("%d", &choice);
13.        trials++;
14.   };
```

a) Line number:

b) Line number:

c) Line number:

C) (9 points)
Find three errors in this program.

```
1.      int a = - 75;
2.
3.      if (a ≠ 0)
4.        {
5.          printf("The number is different from 0 \n");
6.          printf("Please enter another number \n");
7.        }
8.      elseif (a < 0)
9.          printf("The number is negative \n");
10.     else
11.         printf("The number is positive \n");
12.
```

a) Numéro de ligne:/Line number:

b) Numéro de ligne:/Line number:

c) Ligne numéro:/Line number:

Part 3 – Problem Solving: Computing Range of Temperature Readings (30 points)

Review first two steps of the software report provided, and complete the following steps according to the provided instructions.

Step 1 – *Statement of the problem*.

How cold is it outside? The temperature alone is not enough to answer. Other factors including wind speed, relative humidity, and sunshine play important roles in determining coldness outside. In 2001, the National Weather Service (NWS) implemented the new wind-chill temperature to measure the coldness using temperature and wind speed. The formula is given as follows:

$$t_{wc} = 13.12 + 0.6215t_a - 11.37v^{0.16} + 0.3965t_a v^{0.16}$$

where t_a is the outside air temperature measured in degrees Celsius and v is the speed measure in kilometers per hour. The formula **cannot** be used for wind speeds below 5 km/h or temperatures above 10 ^{o}C. Design a software system to compute the wind-chill temperature.

Step 2-a – Collection of Information and *Input and Output Description*

The main function interacts with the user to get the air temperature and wind speed values and validate the inputs (temperature should be less than 10 and the velocity should be larger than 5. The main should prompt the user until a valid value is input. The main function then calls the function `ComputeWindChillTemp()` and returns the wind chill temperature value, and displays the results using the following format (the characters typed in by the user are in bold and italics):

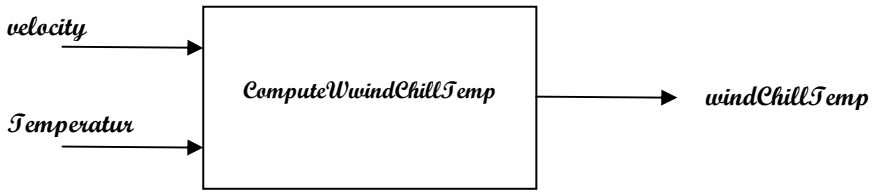
```
Please enter the air temperature: -33
Please enter the wind speed: 2
Wind speed is invalid, please input value larger than 5
Please enter the wind speed: 25
The wind chill temperature is -48.31 for air temperature -33.0
and for wind speed 25.0
```

The `ComputeWindChillTemp()` function receives the outside temperature and the wind speed values and return the corresponding wind chill temperature. The following equation is used:

$$t_{wc} = 13.12 + 0.6215t_a - 11.37v^{0.16} + 0.3965t_a v^{0.16}$$

Note: you might need to use the power function defined in `math.h` library: `pow(x, y)`!

Step 2-b – *Black box diagram (4 points):* Draw the input/output diagram for the function `ComputeWindChillTemp()`:



Step 3-a – *Test Cases (6 points)*: Give three different test cases for the program.

Step 3-b – *Algorithm design*:

Design the algorithm (C code is not accepted at this stage) for the main function.

Answer:

Main subprogram( ) (12 points)

```
Declare Velocity, Temp, and WinsChillTemp as floats

Repeat
        Print "Please enter the air temperature"
        Real value into Temp
        If Temp is larger than 10
                Print "temperature is invalid, please input value less than 10"
while Temp larger than 10

Repeat
        Print "Please enter the air velocity"
        Real value into Velocity
        If Velocity is less than 5
                Print "velcoity is invalid, please input value larger than 5"
while Velocity is less than 5

assign ComputeWindChillTemp (temp, velocity) to WinsChillTemp
Print "The wind chill temperature is ", WinsChillTemp
```

Question 4 (Implementation)

Translate the following pseudocode to C code.

ComputeWindChillTemp (temp, velocity) (8 points)

Declare windChillTemp and var
Assign power(velocity, 0.16) to var
Assign 13.12 + 0.6215*temp – 11.37*var + 0.3965*temp*var to windChillTemp
Return windChillTemp

C Code

```
float ComputeWindChillTemp (float temp, float velocity)
{
      float windChillTemp, var;
      var = power(velocity, 0.16);
      windChillTemp = 13.12 + 0.61215*temp + 0.3965*temp*var;
      return (windChillTemp);
}
```

Extra page (you can detach this page and use it as scratch)

GNG1106 Pseudo-code Reference

Variables: The name of a **variable** in pseudo-code shall respect the same conventions as in C. It is not necessary, nor desirable to declare variables in pseudo-code (this is left as a coding exercise). Example:
variableName
Symbolic Constants: written using a name in UPPERCASE letters. Il is not necessary to define symbolic constants (e.g. PI). But if its definition is necessary (i.e. its value is not evident), use: ***Define MAX as 10***

Instruction blocs is a sequence of pseudo-code instructions with the same indentation. Shall be represented in this reference guide as

Assignment Operation:
Assign *expression* ***to*** *variable_name*
Arithmetic Operators: ***+, -, *, /, mod***
Comparison Operators:
Larger than
Smaller than
Equal to
Not equal to
Larger or equal to
Smaller or equal to
Logical Operators: ***AND, OR, NOT***

Output to the screen:

Print *<list of arguments>*
Example: ***Print "The value of x is ", x***
Argument can be string (in double quotes), special character (TAB, NEWLINE, BELL), a variable or an expression.

Input from the keyboard:

Read value into *variableName*
Read values into *var1, var2, var3, ...*

Decision Structures:

If *logical_expression*

If *logical_expression*

Otherwise

If *logicial_expression*

Otherwise if *logical_expression*

.
.
.

Otherwise if *logical_expression*

Otherwise

Looping Structures

Repeat while *logical expression*

Repeat

While *logical expression*

Defining a subprogram:

name(par1, par2, ..)

*The list par1, par2, ... are the parameters of the subprogram. The subprogram may return a value using "****Return*** *expression".*

Calling a subprogram:

Example: ***Assign name(arg1, arg2,...) to x***
The list arg1, arg2, ... are the arguments of the call to the subprogram.

Array (1-D array):
• A name is used to reference an array, e.g. ***arr.***
• An index is added to the name to reference an element of the array, e.g. ***arr[0], arr[4],***

Matrix (2-D array):
• A name is used to reference an array, e.,g. ***mat.***
• An index is added to the name to reference a row in the matrix, e.g. ***mat[4], mat[0]***
• Two indexes are added to the name to reference a element of the matrix, e.g. ***mat[0][3], mat[4][5]***
.

C Reference Card (ANSI)

Program Structure/Functions

type fnc(*type*$_1$,...)	function declarations
type name	external variable declarations
`main() {`	main routine
declarations	local variable declarations
statements	
`}`	
type fnc(*arg*$_1$,...) {	function definition
declarations	local variable declarations
statements	
`return` *value*`;`	
`}`	
`/* */`	comments
`main(int argc, char *argv[])`	main with args
`exit(`*arg*`)`	terminate execution

C Preprocessor

include library file	`#include <`*filename*`>`
include user file	`#include "`*filename*`"`
replacement text	`#define` *name text*
replacement macro	`#define` *name*(*var*) *text*
Example. `#define max(A,B) ((A)>(B) ? (A) : (B))`	
undefine	`#undef` *name*
quoted string in replace	`#`
concatenate args and rescan	`##`
conditional execution	`#if, #else, #elif, #endif`
is *name* defined, not defined?	`#ifdef, #ifndef`
name defined?	`defined(`*name*`)`
line continuation char	`\`

Data Types/Declarations

character (1 byte)	`char`
integer	`int`
float (single precision)	`float`
float (double precision)	`double`
short (16 bit integer)	`short`
long (32 bit integer)	`long`
positive and negative	`signed`
only positive	`unsigned`
pointer to `int`, `float`,...	`*int, *float,...`
enumeration constant	`enum`
constant (unchanging) value	`const`
declare external variable	`extern`
register variable	`register`
local to source file	`static`
no value	`void`
structure	`struct`
create name by data type	`typedef` *type name*
size of an object (type is `size_t`)	`sizeof` *object*
size of a data type (type is `size_t`)	`sizeof(`*type name*`)`

Initialization

initialize variable	*type name*=*value*
initialize array	*type name*`[]={`*value*$_1$,...`}`
initialize char string	`char` *name*`[]="`*string*`"`

Constants

long (suffix)	`L` or `l`
float (suffix)	`F` or `f`
exponential form	`e`
octal (prefix zero)	`0`
hexadecimal (prefix zero-ex)	`0x` or `0X`
character constant (char, octal, hex)	`'a', '\ooo', '\xhh'`
newline, cr, tab, backspace	`\n, \r, \t, \b`
special characters	`\\, \?, \', \"`
string constant (ends with `'\0'`)	`"abc...de"`

Pointers, Arrays & Structures

declare pointer to *type*	*type* `*`*name*
declare function returning pointer to *type*	*type* `*f()`
declare pointer to function returning *type*	*type* `(*pf)()`
generic pointer type	`void *`
null pointer	`NULL`
object pointed to by *pointer*	`*`*pointer*
address of object *name*	`&`*name*
array	*name*`[`*dim*`]`
multi-dim array	*name*`[`*dim*$_1$`][`*dim*$_2$`]`...

Structures

`struct` *tag* `{`	structure template
declarations	declaration of members
`};`	
create structure	`struct` *tag name*
member of structure from template	*name.member*
member of pointed to structure	*pointer* `->` *member*

Example. `(*p).x` and `p->x` are the same

single value, multiple type structure	`union`
bit field with *b* bits	*member* `:` *b*

Operators (grouped by precedence)

structure member operator	*name.member*
structure pointer	*pointer*`->`*member*
increment, decrement	`++, --`
plus, minus, logical not, bitwise not	`+, -, !, ~`
indirection via pointer, address of object	`*`*pointer*`, &`*name*
cast expression to type	*(type) expr*
size of an object	`sizeof`
multiply, divide, modulus (remainder)	`*, /, %`
add, subtract	`+, -`
left, right shift [bit ops]	`<<, >>`
comparisons	`>, >=, <, <=`
comparisons	`==, !=`
bitwise and	`&`
bitwise exclusive or	`^`
bitwise or (incl)	`|`
logical and	`&&`
logical or	`||`
conditional expression	*expr*$_1$ `?` *expr*$_2$ `:` *expr*$_3$
assignment operators	`+=, -=, *=, ...`
expression evaluation separator	`,`

Unary operators, conditional expression and assignment operators group right to left; all others group left to right.

Standard Utility Functions <stdlib.h>

absolute value of int n	abs(n)
absolute value of long n	labs(n)
quotient and remainder of ints n,d	div(n,d)
returns structure with div_t.quot and div_t.rem	
quotient and remainder of longs n,d	ldiv(n,d)
returns structure with ldiv_t.quot and ldiv_t.rem	
pseudo-random integer [0,RAND_MAX]	rand()
set random seed to n	srand(n)
terminate program execution	exit(status)
pass string s to system for execution	system(s)
Conversions	
convert string s to double	atof(s)
convert string s to integer	atoi(s)
convert string s to long	atol(s)
convert prefix of s to double	strtod(s,endp)
convert prefix of s (base b) to long	strtol(s,endp,b)
same, but unsigned long	strtoul(s,endp,b)
Storage Allocation	
allocate storage	malloc(size), calloc(nobj,size)
change size of object	realloc(ptr,size)
deallocate space	free(ptr)
Array Functions	
search array for key	bsearch(key,array,n,size,cmp())
sort array ascending order	qsort(array,n,size,cmp())

Time and Date Functions <time.h>

processor time used by program	clock()
Example. clock()/CLOCKS_PER_SEC is time in seconds	
current calendar time	time()
$time_2$-$time_1$ in seconds (double)	difftime($time_2$,$time_1$)
arithmetic types representing times	clock_t,time_t
structure type for calendar time comps	tm

tm_sec	seconds after minute
tm_min	minutes after hour
tm_hour	hours since midnight
tm_mday	day of month
tm_mon	months since January
tm_year	years since 1900
tm_wday	days since Sunday
tm_yday	days since January 1
tm_isdst	Daylight Savings Time flag

convert local time to calendar time	mktime(tp)
convert time in tp to string	asctime(tp)
convert calendar time in tp to local time	ctime(tp)
convert calendar time to GMT	gmtime(tp)
convert calendar time to local time	localtime(tp)
format date and time info	strftime(s,smax,"*format*",tp)
tp is a pointer to a structure of type tm	

Mathematical Functions <math.h>

Arguments and returned values are double

trig functions	sin(x), cos(x), tan(x)
inverse trig functions	asin(x), acos(x), atan(x)
arctan(y/x)	atan2(y,x)
hyperbolic trig functions	sinh(x), cosh(x), tanh(x)
exponentials & logs	exp(x), log(x), log10(x)
exponentials & logs (2 power)	ldexp(x,n), frexp(x,*e)
division & remainder	modf(x,*ip), fmod(x,y)
powers	pow(x,y), sqrt(x)
rounding	ceil(x), floor(x), fabs(x)

Flow of Control

statement terminator	;
block delimeters	{ }
exit from switch, while, do, for	break
next iteration of while, do, for	continue
go to	goto *label*
label	*label*:
return value from function	return *expr*

Flow Constructions

if statement	if (*expr*) *statement* else if (*expr*) *statement* else *statement*
while statement	while (*expr*) *statement*
for statement	for ($expr_1$; $expr_2$; $expr_3$) *statement*
do statement	do *statement* while(*expr*);

switch statement	switch (*expr*) {
	case $const_1$: $statement_1$ break;
	case $const_2$: $statement_2$ break;
	default: *statement*
	}

ANSI Standard Libraries

<assert.h>	<ctype.h>	<errno.h>	<float.h>	<limits.h>
<locale.h>	<math.h>	<setjmp.h>	<signal.h>	<stdarg.h>
<stddef.h>	<stdio.h>	<stdlib.h>	<string.h>	<time.h>

Character Class Tests <ctype.h>

alphanumeric?	isalnum(c)
alphabetic?	isalpha(c)
control character?	iscntrl(c)
decimal digit?	isdigit(c)
printing character (not incl space)?	isgraph(c)
lower case letter?	islower(c)
printing character (incl space)?	isprint(c)
printing char except space, letter, digit?	ispunct(c)
space, formfeed, newline, cr, tab, vtab?	isspace(c)
upper case letter?	isupper(c)
hexadecimal digit?	isxdigit(c)
convert to lower case?	tolower(c)
convert to upper case?	toupper(c)

String Operations <string.h>

s,t are strings, cs,ct are constant strings

length of s	strlen(s)
copy ct to s	strcpy(s,ct)
up to n chars	strncpy(s,ct,n)
concatenate ct after s	strcat(s,ct)
up to n chars	strncat(s,ct,n)
compare cs to ct	strcmp(cs,ct)
only first n chars	strncmp(cs,ct,n)
pointer to first c in cs	strchr(cs,c)
pointer to last c in cs	strrchr(cs,c)
copy n chars from ct to s	memcpy(s,ct,n)
copy n chars from ct to s (may overlap)	memmove(s,ct,n)
compare n chars of cs with ct	memcmp(cs,ct,n)
pointer to first c in first n chars of cs	memchr(cs,c,n)
put c into first n chars of cs	memset(s,c,n)