Université d'Ottawa
Faculté de Génie

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/GNG1506 Engineering Computation (Fall 2009)

Name (surname, first name):_____

Student Number:_____

Examiners: Professor Ymina Sami
Professor Mohamed Eid
Professor A. H. G. Al-Dhaher

Final Examination, December 23, 2009 Time allowed: 3 hours

Note:

- This is a close-book examination and you are required to abide by the University's regulations regarding the conduct of exams
- Answer ALL questions.
- Use the provided space to answer the following questions. If more space is needed, use the back of the page.
- Read all the questions carefully before you start.

Good luck

Question	Marks gained %	Marks allocated %
1		20
2		20
3		25
4		25
5		10
Total		**100**

Question 1
Part A (15 marks)
Give the output generated by the following program:

```
#include<stdio.h>
#include<math.h>
#define TEST 4
#define WEST 2.30
int main()
{
        Int i=0, j=2, k=10, m;
        Flat a=3.5, b=10.6, x;

/*a) */ m=b;
/*b) */ m=a+b;
/*c) */ m=(int)a+b;
/*d) */ i=i%j;
/*e) */ m=j/k;
/*f) */ x=j/TEST;
/*g) */ i=TEST*WEST;
/*h) */ b/= j;
/*i) */ a=i*k;
/*j) */ x=tan(i-=i);
/*k) */ m=TEST%(int)WEST+1;
/*l) */ x=sqrt(pow(8,j/(TEST-1)));
/*m) */ x=10.123456789;
Return 0;
};
```

```
printf("a) %d\n", m);
printf("b) %d\n", m);
printf("c) %d\n", m);
printf("d) %d\n", i);
printf("e) %d\n", m);
printf("f) %f\n", x);
printf("g) %d\n", i);
printf("h) %f\n", b);
printf("i) %f\n", a);
printf("j) %f\n", x);
printf("k) %d\n", m);
printf("l) %f\n", x);
printf("m) %f\n", x);
```

Answer:

a	b	x	i	j	k	m
3.5	10.6	0.000000	0	2	10	10
90		1.000000	9			14
						13
						0
						1

Question 1
Part B (5 marks)
Consider the following program:

```
#include <stdio.h>
#define price1  1.5
#define price2  2.0
#define price3  3.5
#define price4  5.0

main ()
{
      int age, day;
      double price;
      printf("Please enter the values of age and day\n");
      scanf("%d %d", &age, &day);
      if ((age < 12 ) && (day ==7))
          price = price1;
      else if (age < 12 )
          price = price2;
      else if (day == 7)
          price = price3;
      else
          price = price4;
      printf("please pay $%.2f\n",price);
}
```

What will the output of this program for the following input values:

Input		Output
12	7	3.50
10	2	2.00
2	33	2.00
-1	7	1.50

Question 2
Part A (10 marks)
Assume the following declaration for the array g: /

```
int i, j,  g[3][3] = {{0,0,0},{1,1,1},{2,2,2}};
```

Give the value of sum after each of the following sets of statements are executed. /

I.
```
sum = 0;
for (i = 0; i <= 2; i++)
{
    for (j = 0; j <= 2; j++)
        sum + = g[i][j];
}
```

Answer to Question 2AI

9

II.
```
sum = 1;
for (i = 1; i <= 2; i++)
{
    for (j = 0; j <= 1; j++)
        sum * = g[i][j];
}
```

Answer to Question 2AII

4

III.
```
sum = 0;
for (j = 0; j <= 2; j++)
     sum - = g[2][j];
```

Answer to Question 2AIII

-6

IV.
```
sum = 0;
for ( i = 0; i <= 2; i++)
     sum + = g[i][1];
```

Answer to Question 2AIV

3

Question 2
Part B (10 marks)

Consider a 2-by-5 integer array t and in each section write a series of C statements that:/

I. Inputs the values for the elements of t from the terminal. /

Answer:

```
for (int i=0; i<2; i++)
for (int j=0; j<5; ++)
scanf("%d", t[][]);
```

II. Determines and prints the smallest value in array t. /

Answer:

```
int t[2][5];
int small
if (t[i][j] < small
small = [i][j];
```

```
int i
for i=0; i<=1; ++) {
for int j=0; j<=4; ++) {
```

III. Displays the elements of the first row of t /

Answer:

```
for(int j=0; j<5; j++
printf("%d"\n, t([0][j]
```

IV. Totals the elements of the fourth column of t. /

Answer:

```
for (int i=0; i<2; i++
Sum = +=t[1][3]
```

Question 3

Part A (17 marks)

For the following code, you may assume that a, b, aPtr and bPtr are located at addresses 1000, 2000, 3000 and 4000, respectively, and that `sizeof(int)` yields 4. Give in the boxes shown below, the output generated by each of the following programs.

```
#include <stdio.h>
void fcn(int *, int);

void main()
{
 int a=1, *aPtr=&a,
     b=3, *bPtr=&b;

 printf("%u\n",&bPtr);
 printf("%u\n",&a);

 fcn(aPtr, b);

 printf("%d\n", b);
 printf("%d\n", a);
}

void fcn(int *Ptr, int z)
{
 *Ptr=z * --(*Ptr);
 printf("%d\n", Ptr);
 printf("%d\n", z);
}
```

```
#include <stdio.h>

void fun(int *);
void main()
{
 int a, *aPtr=&a;

 a=100;
 printf("%u\n", aPtr);
 printf("%d\n", a);
 fun(aPtr);
 printf("%d\n", a);
}
void fun(int *y)
{
*y=*y+3;
y=y+3;

printf("%d\n",sizeof(int));
printf("%u\n", y);
}
```

```
#include <stdio.h>
#define M 3
#define N 2
void fcn(int, int[][N], int*);

void main()
{
 int b[M][N]={1}, i, j;

 fcn(b[0][0], b, &b[2][1]);

   for(i=0; i<=M-1; i++)
   {
     for(j=0; j<N; j++)
     printf("%d ", b[i][j]);
   printf("\n");
   }
}

void fcn(int x, int y[][N], int
*z)
{
 int k;

  for(k=4; k>=1; k--)
  *(z-k)=55;

  for(k=0; k<=N-1; k++)
  y[x][k]=x+1;


 y[2][1]=66;
 x=88;
}
```

Answer:

4000 1000 1000 3 3 5	1000 100 4 1003 103	

Question 3
Part B (8 marks)

Give the output generated by the following program:

```
#include <stdio.h>
int Fonction1 (int x, int y);
int main()
{
int i,j;
for (i=0; i<=4;i++)
   {
    for(j=1; j<=4-i;j++)
            printf("*");

    for(j=0; j<=i;j++)
            printf("%d*",Fonction1(i,j));
    printf("\n");
    }
return 0;
}

int Fonction1 (int x, int y)
{
if ((y==0)||(y==x))
   return(1);
else return (Fonction1(x-1,y)+Fonction1(x-1,y-1));
}
```

Answer :

.....

.....

.....

.....

.....

.....

.....

.....

Question 4 (25 marks)
Problem Solving: Computing total payments of loan

Review first two steps of the software report provided (this page and next one), and complete Step 3 and 4 according to the provided instructions.

Step 1 – Statement of the problem.
Design a program that computes the monthly payment and the total payment for a bank loan given the following:

a) Amount of loan
b) Duration of loan in months
c) Interest rate for loan

Your program should read in one loan at a time, perform the required computation, and store the loan attributes (amount of loan, duration, interest, monthly payment and total payment) in an array of structures called `loans`
To simplify we suppose that there is exactly five loans.
The program should continue by writing the five elements of the array in a file called `loans.txt`
Finally, the program should input an interest I, uses the array `loans` to compute and print the sum of the total payments corresponding to the loans having this given interest I.

Step 2 – Collection of Information and Input and Output Description
In the main function, the user is asked to input 5 loans. For each loan the user should give the amount of loan, the duration of loan in months and the interest rate of loan. The main function computes the monthly payment and the total payment for this loan. The computation of monthly payment is done by calling a function called `ComputemonthlyPay()`.The formula for computing the total payment is `TotalPayement=monthly Payment *Duration`
The amount, duration, interest rate, monthly payment, and total payment of a loan are entered as an element of a local variable of the main fuction `loans` which is an array of one dimension, size5, type `type_loan` defined as follows:

```
struct type_loan{
double amount;
int duration;
double interest;
double monthlyPay;
double totalPay;} ;
```

The main function should store the array of structures `loans` in a file called `loans.txt`
Finally, the main function asks the user to input an interest value and calls a function `sumInterest()` that returns the sum of total payments corresponding to the loans having this given interest. The main function finishes by displaying this sum.

The `ComputemonthlyPay()` function has the following prototype:

```
double ComputemonthlyPay(double Amount, double Interest, int
Duration);
```

This function computes monthly payment by applying the following formula:

$$monthlypay = \frac{Interestm \times \exp^{Duration} \times Amount}{\exp^{Duration} - 1.0}$$

Where
`Interstm=Interest/1200.0`
`exp=(1.0+Interestm)`

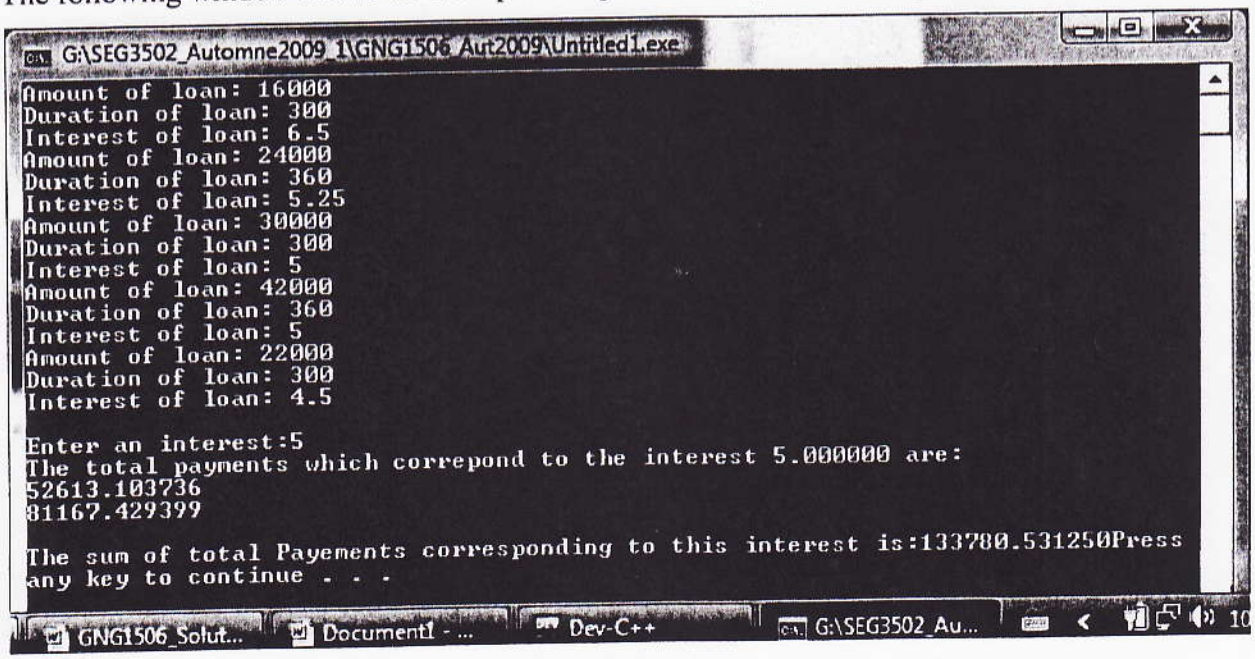
The `ComputemonthlyPay()` function returns the monthly payment computed.

The `sumInterest()` function has the following prototype:

```
double sumInterest(double Interest , type_loan loans[], int
size);
```

This function computes and returns the sum of total payments of the elements of the second parameter `loans` which have as interest the value given as the first parameter. The third parameter `size` is the size of the array `loans[]`.

The following window shows an example of a possible output of the required program:



Step 3:

a) Test Cases (4 marks):

Give two different test cases of the function `ComputemonthlyPay()`

Answer:

Test case1:

Test case2:

Step 3:

b) Algorithm design: (5 Marks)

Design the algorithm, give the pseudocode (C code is not accepted at this stage) for the `ComputemonthlyPay()` function.

Answer:

Step 4 Implementation:

a) (10 marks) We give in the following the C code for the main() function

Note: Portions of various statements in the program have been removed and replaced with a solid line. Restore the 15 missing portions.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#define size 5

struct type_loan{
double amount;
int duration;
double interest;
double monthlyPay;
double totalPay;} ;


double ComputemonthlyPay(double, double, int);
double sumInterest(double , type_loan[], int);

int main() {
type_loan loans[size];
FILE *P_File;
double Interest;
int i=0;
while (_________){
      printf("Amount of loan: ");
      scanf("%lf", _____________);
      printf("Duration of loan: ");
      scanf("%d", ____________________);
      printf("Interest of loan: ");
      scanf("%lf", &loans[i].interest);
      loans[i].monthlyPay=ComputemonthlyPay(_____________
 _____________________________________________);
      loans[i].totalPay=_____________________________;
      ___________________
}
```

Step 4 Implementation:

a) (Continued)

```
P_File = fopen(_____________________________-);
i=0;
while (_____________________){
        fprintf(P_File, "%f\n", loans[i].amount);
        fprintf(P_File, "%d\n", _____________________);
        fprintf(P_File, "%f\n", _____________________);
        fprintf(P_File, "%f\n", _____________________);
        fprintf(P_File, "%f\n", _____________________);
        _________________;
}
fclose(_____________________);

printf("\nEnter an interest:");
scanf("%lf", &Interest);
printf("\nThe sum of total Payments corresponding to this
interest is:%f", sumInterest(_____________________________));
system("PAUSE");
return(0);
}
```

Step 4 Implementation:

b) Write the C code for the sumInterest() function **(6 marks)**

Answer:

Question 5: (10 marks)

(a) Provide Visual Basic for Applications code (that runs in the Microsoft Excel Application) to swap two integers stored in the two cells A7 and B8 of the Worksheets "swap".

Answer:

(b) Assume that an excel sheet stores 100 temperature readings. Provide Visual Basic for Applications code to down-sample the temperature readings to 20 and store the results in sheet 2. That is, for each five samples, copy only one sample into sheet 2 of the excel document.

Answer:

(c) Provide Visual Basic for Applications code that prompts the user for her name, then search for her name in the first 50 cells, of column 1, of an Excel worksheet called "names". The code should print back a message if the name is found (and print the cell where the name is found), or print "not found" otherwise.

Answer:

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(arg1,...) {	function definition
declarations	local variable declarations
statements	
`return` *value*;	
`}`	
`/* */`	comments
`main(int argc, char *argv[])`	main with args
`exit(ary)`	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(pts,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,max,"*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)

Visual Basic Reference:

1. To access a cell in an excel sheet:

WorkSheets(1).Cells(x,y)

2. Message Box:

YourName = Application.InputBox("Enter your Student number")

MsgBox(("Hello World")

3. If/Else statement:

If *logical expression* Then *Instruction bloc*
ElseIf *logical expression* Then *Instruction bloc*
ElseIf …

End If

4. Loops:

For *Count* = *StartValue* To *EndValue* [Step *Increment*] *instruction bloc*
Next [*Count*]

Do While *logical expression*
instruction Bloc
Loop

Do
instruction Bloc
Loop while *logical expression*