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 Yamina Sami
Professor Mohamed Eid
Professor A. H. G. Al-Dhaher

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 8
#define WEST 4.6

int main()
{
int i=0, j=4, k=20, m;
float a=3.7, b=10.5, x;


/* a) */  x=10.12;                                  printf("a) %f\n", x);
/* b) */  m=b*2;                                    printf("b) %d\n", m);
/* c) */  x=sqrt(pow(8,j/(TEST-1)));                printf("c) %f\n", x);
/* d) */  m=a+b;                                    printf("d) %d\n", m);
/* e) */  m=(int)a+b;                               printf("e) %d\n", m);
/* f) */  m=TEST%(int)WEST+1;                       printf("f) %d\n", m);
/* g) */  i=i%(j+1);                                printf("g) %d\n", i);
/* h) */  m=j/k;                                    printf("h) %d\n", m);
/* i) */  x=j/(TEST*2);                             printf("i) %f\n", x);
/* j) */  x=tan(i-=i);                              printf("j) %f\n", x);
/* k) */  i=TEST*WEST;                              printf("k) %d\n", i);
/* l) */  b/=j;                                     printf("l) %f\n", b);
/* m) */  a=i*k;                                    printf("m) %f\n", a);
return 0;
};
```

Solution:

```
a) 10.120000
b) 21
c) 1.000000
d) 14
e) 13
f) 1
g) 0
h) 0
i) 0.000000
j) 0.000000
k) 36
l) 2.625000
m) 720.000000
Press any key to continue . . .
```

Question 1 Part B (5 marks)

What does the following code segment display? Use each of these inputs: 345, 82, 6. Then give the output of the code.

```
printf("\n Enter a positive integer   ");
scanf("%d, &num);
        do{
                printf("%d   ", num % 10);
                num /= 10;
        }while (num > 0);
        printf('\n");
```

Solution:

```
Enter a positive integer 345
        5       4       3

Enter a positive integer 82
        2       8

Enter a positive integer 6
        6
```

Question 3 (20 marks, 2 marks each part)
Answer each of the following. Assume that single-precision floating-point numbers are stored in 4 bytes, and that the starting address of the array is at location 1002500 in memory. Each part of the exercise should use the result of previous parts where appropriate.

A) Declare an array of type float called **numbers** with 10 elements and initialize the elements to the values 0.0, 1.1, 2.2, 3.3, ……..9.9. Assume the symbolic constant **SIZE** has been defined as 10.

Solution:

```
float number[SIZE] = {0.0, 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8, 9.9};
```

B) Declare a pointer **nPtr** that points to an object of type float

Solution:

```
float  *nPtr;
```

C) Print the elements of array **numbers** using array subscript notation. Use a **for** structure and assume an integer control i has been declared. Print each number with 1 position of precision to the right of the decimal point.

Solution:

```
for ( i = 0; i < SIZE; i++)
    printf( "% .1f ", numbers[ i ]);
```

D) Give two separate statements that assign the starting address of array **numbers** to the pointer variable **nPtr.**

Solution:

```
nPtr = numbers;

nPtr = &numbers[ 0 ];
```

E) Print the elements of array **numbers** using pointer/offset notation with the pointer **nPtr.**

Solution:

```
for ( i = 0; i < SIZE; i++ )
        printf( "%.1f ", *(nPtr + i))
```

F) Print the elements of array **numbers** using pointer/offset notation with the array name as the pointer.

Solution:

```
for ( i = 0; i < SIZE; i++)
        printf( "%.1f ", *( numbers + i ));
```

g) Print the elements of array **numbers** by subscripting pointer **nPtr.**

Solution:

```
for ( i = 0; i < SIZE; i++)
        printf( "%.1f ", nPtr[i]);
```

h) Refer to element 4 of array **numbers** using array subscript notation, pointer/offset notation with array name as the pointer, pointer subscript notation with **nPtr** and pointer/offset notation with **nPtr.**

Solution:

```
numbers[ 4 ]

*(numbers + 4)

nPtr[ 4 ]

*( nPtr + 4 )
```

I) Assuming that **nPtr** points to the beginning of array **numbers**, what address is referenced by nPtr + 8? What is stored in that location?

Solution:

The address is 1002500 + 8 * 4 = 1002532. The value is 8.8

J) Assume **nPtr** points to **numbers** [5], what address is referenced by nPtr - =4. What is the value stored at that location?

Solution:

The address of numbers[5] is 1002500 + 5 * 4 = 1002520

The address of nPtr - = 4 is 1002520 -4 * 4 = 1002504

The values at that location is 1.1

Question 3

Part A (17 marks)

For the following code, you may assume that `a`, `b`, `aPtr` and `bPtr` are located respectively at addresses 1000, 2000, 3000 and 4000, 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 *);

int main()
{
 int a=2, *aPtr=&a,
     b=4, *bPtr=&b;

 printf("%u\n",&aPtr);
 printf("%u\n",&b);

 fcn(a, bPtr);

 printf("%d\n", b);
 printf("%d\n", a);
 return(0);
}

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

```
#include <stdio.h>

void fun(int *);
int main()
{
 int b, *bPtr=&b;

 b=10;
 printf("%u\n", bPtr);
 printf("%d\n", b);
 fun(bPtr);
 printf("%d\n", b);
 return(0);
}
void fun(int *y)
{
*y=*y-4;
y=y-4;
printf("%d\n",sizeof(int));
printf("%u\n", y);
}
```

```
#include <stdio.h>

#define M 3
#define N 2
void fcn(int, int[][N], int*);

int main()
{
 int b[M][N]={2}, i, j;

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

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

void fcn(int x, int y[][N], int
*z)
{
 int k;
  for(k=4; k>1; k--)
  *(z-k)=25;
  for(k=0; k<N; k++)
  y[x][k]=x+1;
 y[2][1]=30;
 x=40;
}
```

Answer:

3000 **2000** **2000** **2** **10** **2**	**2000** **10** **4** **1884** **6**	**1 1** **25 25** **0 30**

Question 3
Part B (8 marks)
Give the output generated by the following program:

```
#include <stdio.h>
#include <stdlib.h>
int Fonction1 (int x, int y);
int main()
{
int i,j;
for (i=0; i<=3;i++)
   {

    for(j=0; j<3-i;j++)
             printf("*");

    printf("%d*",Fonction1(i,j));

    printf("\n");
    }

system("PAUSE");
return 0;
}

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

Answer :

*****-9***
****-5***
5
9*

Question 4 (25 marks)
Problem Solving: Hardware store inventory

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

Step 1 – Statement of the problem.
Design a program that determines the items of a hardware store to be ordered. We keep an inventory that includes the following information about each item.

a) Identification number of item
b) Name of item
c) Price of item
d) Available quantity of item
e) Stock of security of item

The stock of security of an item is the minimal quantity of this item that must be available to meet the requirements of the customers.
Your program should read in one item at a time from a file called `items.txt`, and store the item attributes (number of item, name, price, available quantity and stock of security) in an array of structures called `items`. To simplify we suppose that there are exactly five items.
The program should use the array `items` to output the items that must be ordered. These are the items whose available quantity falls below their stock of security.
Finally, the program proceeds by giving the number and the price of the most expensive item.
Step 2 – Collection of Information and Input and Output Description
In the main function, the program reads 5 items from the file `items.txt` and stores them in the array `items`. For each item, the main function reads from the file the number, the name, the price, the available quantity, and the stock of security.
The number, the name, the price, the available quantity, and the stock of security are entered as an element of a local variable, `items`, of the main function. `items` is an array of one dimension, size5, type `type_item` defined as follows:

```
typedef struct{
int number;
char name[20];
double price;
double quantity;
double qty_sec;} type_item;
```

The main function continues by calling a function Order() that displays the attributes of the items that are to be ordered.
The main function ends by calling a function MaxPrice() that computes the number of the most expensive item and its corresponding price.

The Order()function has the following prototype:

```
void Order(type_item items[], int size);
```

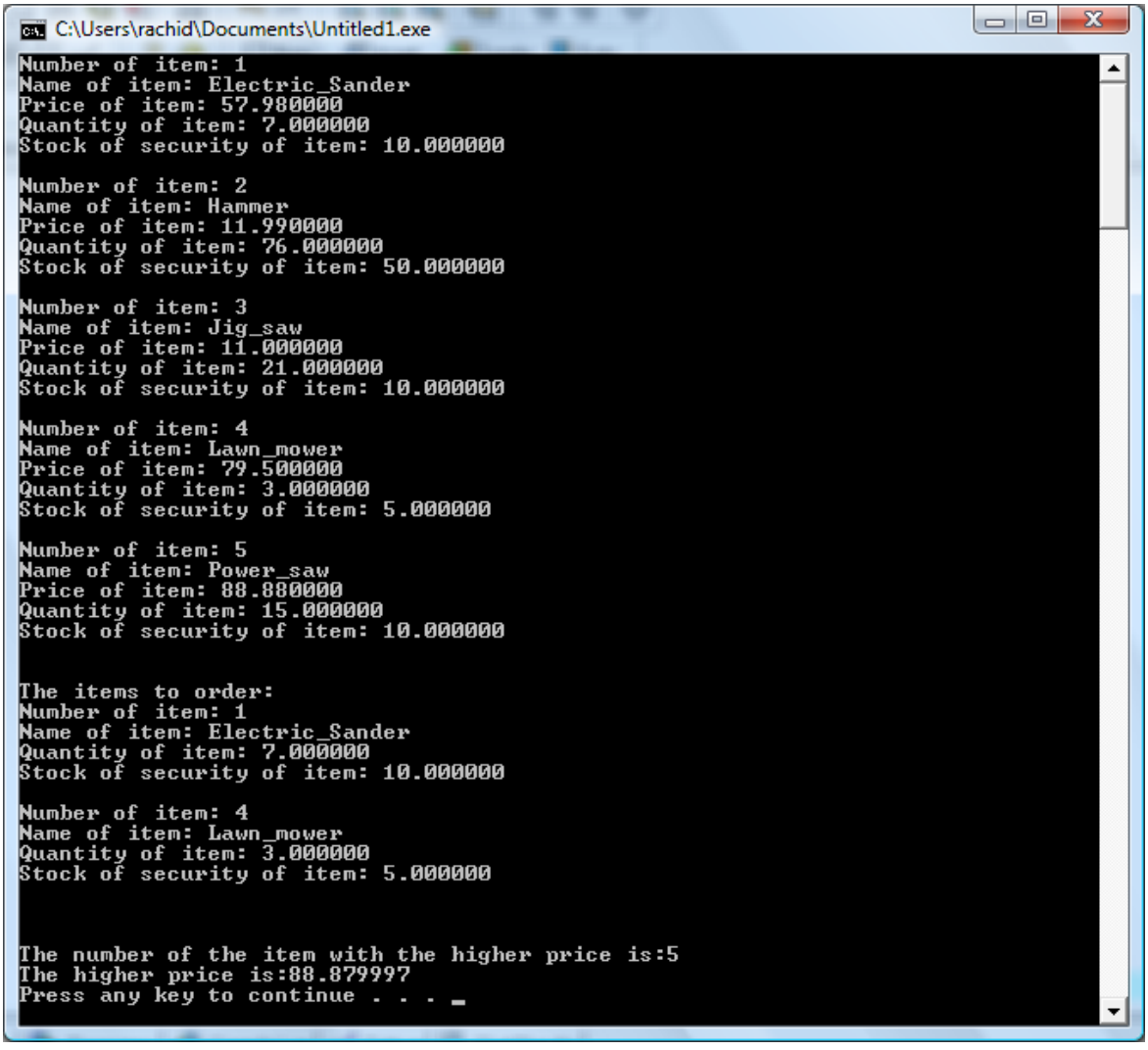
This function displays the attributes of all the items available in quantities less than their stock of security. The first parameter is the array `items[]`. The second parameter is the size of the array `items[]`.

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

```
int MaxPrice(double * price, type_item items[], int size);
```

This function `MaxPrice()` computes and returns the number of the most expensive item of `items[]` (the second parameter). If the maximum price appears more than once in `items[]`, MaxPrice() returns the number of the most expensive item that appears first. The third parameter is the size of the array `items[]`. The first parameter is a pointer to the price of the most expensive item.

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



Step 3:
a) Test Cases (4 marks):
Give two different test cases of the function `MaxPrice()`

Answer:
Test case1:
Items[0].number=1
Items[0].price=43.5
Items[1].number=2
Items[1].price=50
Items[2].number=3
Items[2].price=60
Items[3].number=4
Items[3].price=20
Items[4].number=5
Items[4].price=60

Returned value:3
Maximum price:60

Test case2:
Items[0].number=1
Items[0].price=67
Items[1].number=2
Items[1].price=70
Items[2].number=3
Items[2].price=50
Items[3].number=4
Items[3].price=12
Items[4].number=5
Items[4].price=70

Returned value:2
Maximum price:70

Step 3:
b) Algorithm design: (5 Marks)
Design the algorithm (C code is not accepted at this stage) for the `MaxPrice()` function.
To access the fields of a structure variable use the same syntax as C.

Answer (I am giving the C code since I do not know the syntax of your pseudo code).

```
int MaxPrice(double * price, type_item items[], int size1){
float Max=0;
int numMax;
int i=0;
while (i<size1){
        if (items[i].price>Max){
           Max=items[i].price;
           numMax=items[i].number;
        }
        i++;
}
(*price)=Max;
return numMax;
}
```

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>
#define size 5

typedef struct{
int number;
char name[20];
double price;
double quantity;
double qty_sec;} type_item;


int MaxPrice(double *, type_item[], int);
void Order(type_item[], int);

int main() {
type_item items[size];
double price;

FILE *P_File;
P_File = fopen(_____________________);
int i=0;
while (_____________){
      fscanf(P_File, "%d\n", &items[i].number);
      fscanf(P_File, "%s\n", _____________);
      fscanf(P_File, "%lf\n", _____________);
      fscanf(P_File, "%lf\n", _____________________);
      fscanf(P_File, "%lf\n", ____________________________);
      ____________________;
}
fclose(P_File);
```

Step 4 Implementation:

a) (Continued)

```
i=0;
while (_____________________){
      printf("Number of item: ");
      printf("%d\n", _____________________);
      printf("Name of item: ");
      printf("%s\n", _____________________);
      printf("Price of item: ");
      printf("%f\n", _____________________);
      printf("Quantity of item: ");
      printf("%f\n", _____________________);
      printf("Stock of security of item: ");
      printf("%f\n\n", items[i].qty_sec);
      _____________;
}



printf("\nThe items to order:\n");
Order(_____________________);

printf("\n\nThe number of the item with the higher price is:%d",
MaxPrice(_____________________________));
printf("\nThe higher price is:%f\n", price);

system("PAUSE");
return(0);
}
```

Step 4 Implementation:

a) Answer:

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

typedef struct{
int number;
char name[20];
double price;
double quantity;
double qty_sec;} type_item;


int MaxPrice(double *, type_item[], int);
void Order(type_item[], int);

int main() {
type_item items[size];
double price;

FILE *P_File;
P_File = fopen("items.txt", "r");
int i=0;
while (i<size){
      fscanf(P_File, "%d\n", &items[i].number);
      fscanf(P_File, "%s\n", items[i].name);
      fscanf(P_File, "%lf\n", &items[i].price);
      fscanf(P_File, "%lf\n", &items[i].quantity);
      fscanf(P_File, "%lf\n", &items[i].qty_sec);
      i++;
}
fclose(P_File);
```

Step 4 Implementation:

a) Answer (Continued):

```
i=0;
while (i<size){
      printf("Number of item: ");
      printf("%d\n", items[i].number);
      printf("Name of item: ");
      printf("%s\n", items[i].name);
      printf("Price of item: ");
      printf("%f\n", items[i].price);
      printf("Quantity of item: ");
      printf("%f\n", items[i].quantity);
      printf("Stock of security of item: ");
      printf("%f\n\n", items[i].qty_sec);
      i++;
}



printf("\nThe items to order:\n");
Order(items, size);

printf("\n\nThe number of the item with the higher price is:%d",
MaxPrice(&price, items, size));
printf("\nThe higher price is:%f\n", price);

system("PAUSE");
return(0);
}
```

Step 4 Implementation:

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

Answer

```
void Order(type_item items[], int size1){
int i=0;
while (i<size1){
      if (items[i].quantity < items[i].qty_sec){
            printf("Number of item: ");
            printf("%d\n", items[i].number);
            printf("Name of item: ");
            printf("%s\n", items[i].name);
            printf("Quantity of item: ");
            printf("%f\n", items[i].quantity);
            printf("Stock of security of item: ");
            printf("%f\n\n", items[i].qty_sec);
      }
      i++;
}
}
```

Question 5: (10 marks)

(a) Provide Visual Basic for Applications code (that runs in the Microsoft Excel Application) to find the maximum of three cells (A7, A8, and A9) of sheet 1. The maximum value should be printed to the user using a message box.

Solution

```
Dim max As Integer
max = Sheets(1).Cells(7, 1)
If max < Sheets(1).Cells(8, 1) Then
   max = Sheets(1).Cells(8, 1)
End If
If max < Sheets(1).Cells(9, 1) Then
max = Sheets(1).Cells(9, 1)
End If
   MsgBox ("The maximum value is: " & max)
```

(b) Assume that an excel sheet stores 100 grades in the first column of sheet 1. Provide Visual Basic for Applications code to adjust the grades by adding 10% of each grade and store the adjusted grades in the second column. The code should also compute the adjusted average and display it to the user in a message box.

Solution

```
Dim x As Integer
Dim sum As Double
sum = 0
For x = 1 To 100 Step 1
   Sheets(1).Cells(x, 2) = 1.1 * Sheets(1).Cells(x, 1)
   sum = sum + Sheets(1).Cells(x, 2)
Next
MsgBox ("The adjusted average is: " & sum / 100)
```

(c) Provide Visual Basic for Applications code that prompts the user for two numbers (X and Y), then generate 100 random numbers between X and Y. All the random numbers should be stored in sheet 1 of the excel document.

Solution

```
Dim x As Integer
Dim y As Integer
Dim counter As Integer

x = Application.InputBox("Enter first number: ")
y = Application.InputBox("Enter second number: ")

For counter = 1 To 100 Step 1
   Sheets(1).Cells(counter, 1) = Int(Rnd * y) + x
Next
```

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)