Aaron Byrne

C15709609@mydit.ie

Semester One

Assessments  
2 labs worth 14%   
1 Continuous assignments worth 11%  
25% in total

Programming

Michael Collins

Contents

[Week One 1](#_Toc444682840)

[Types of Data 1](#_Toc444682841)

[Variable Name rules 1](#_Toc444682842)

[Comments 1](#_Toc444682843)

[Delimiters 1](#_Toc444682844)

[Week Two 2](#_Toc444682845)

[Arithmetic Operators 2](#_Toc444682846)

[Increment and Decrement 2](#_Toc444682847)

[Operator Priorities 3](#_Toc444682848)

[Type Conversion and Casting 3](#_Toc444682849)

[Week Three 4](#_Toc444682850)

[Single character input 4](#_Toc444682851)

[Pitfalls 4](#_Toc444682852)

[Week Four Control Statements 5](#_Toc444682853)

[if statement 5](#_Toc444682854)

[Operator Meaning 5](#_Toc444682855)

[If/ Else statement 6](#_Toc444682856)

[Logical operations 6](#_Toc444682857)

[Switch Statement 8](#_Toc444682858)

[Week Six Data Structures 9](#_Toc444682859)

[Array 9](#_Toc444682860)

[Symbolic name 10](#_Toc444682861)

[Week 8 Arrays continued 11](#_Toc444682862)

[Two Dimensional Arrays 11](#_Toc444682863)

[Week 9 Pointers 13](#_Toc444682864)

[The indirection operator (\*) 15](#_Toc444682865)

[Week 10 Pointers and Arrays 16](#_Toc444682866)

[Pointer and Arrays 16](#_Toc444682867)

[Week 11 Dynamic Memory Allocation(DMA) 18](#_Toc444682868)

[Malloc 18](#_Toc444682869)

[Calloc 19](#_Toc444682870)

[Realloc 20](#_Toc444682871)

[Week 12 Review Week 22](#_Toc444682872)

[Week One Functions 24](#_Toc444682873)

[Passing parameters 26](#_Toc444682874)

[Return types 27](#_Toc444682875)

[Passing an argument by Value. 28](#_Toc444682876)

[Passing an argument by reference 28](#_Toc444682877)

[Week 3 Passing Arrays to a Function 30](#_Toc444682878)

[/\*passing a 1-d array to a function\*/ 30](#_Toc444682879)

[/\*Passing a 2-d array \*/ 31](#_Toc444682880)

[4 Storage classes 32](#_Toc444682881)

[Auto 32](#_Toc444682882)

[Static 32](#_Toc444682883)

[Extern 33](#_Toc444682884)

[Register 33](#_Toc444682885)

[Mathematical Functions 34](#_Toc444682886)

[Strings 35](#_Toc444682887)

[String literals 35](#_Toc444682888)

[Long character strings 35](#_Toc444682889)

[Strings and Arrays 35](#_Toc444682890)

[Printf and Strings 35](#_Toc444682891)

[Scanf and Strings 35](#_Toc444682892)

[Puts() Function 35](#_Toc444682893)

[Gets() Function 35](#_Toc444682894)

[Initialise a String 36](#_Toc444682895)

[Program using pointer notation and strings 36](#_Toc444682896)

[String Functions 37](#_Toc444682897)

[Strlen() 37](#_Toc444682898)

[Strcpy 37](#_Toc444682899)

[Strcat() 38](#_Toc444682900)

[Strcmp() 38](#_Toc444682901)

[Strncmp 38](#_Toc444682902)

[Past Exam Question. 38](#_Toc444682903)

[2012 Part B. Question 4. (out of 24 marks) 38](#_Toc444682904)

[Exam Change 39](#_Toc444682905)

Semester One

# Week One

*#include <stdio.h>*

*main ()*

*{ Variable Name*

*int v1;  
 float v2; Variables  
 char v3;*

*v1=10;  
 v2=-6.23; Delimiters  
 v3= ‘a’;*

*printf(“v1 contains %d”,v1);  
printf(“v2 contains %f”,v2);  
printf(“v3 contains %c”,v3);*

*}//end main*

## Types of Data

* Integer
  + Whole numbers
  + Range -64,000 🡪 +64,000
  + 4 bits of memory
  + Short int(short)
    - Smaller range
    - 2 bits if memory
  + Long int(long)
    - Larger range
    - 8 bits of memory
  + Unsigned int
    - Moves range into positive
* Floating Point number(float)
  + Decimal point
  + 8 bits of memory
  + Double
    - Double the size of floats
* Character(char)
  + A single character in **single** quotes (‘a’)
* String Double quotes, words
* Constant variables- variables that you can’t change.  
  Must be hard coded

## Variable Name rules

* Must Only contain a number, alphanumeric character or underscore
* Must start with a letter or underscore
* Must not use a command line word
* No limit to variable name length, but best practice is to keep it short.

## Comments

* /\* beginning of a multiline comment  
  the end of a multiline comment\*/
* // single line comment
* Always comment where it is harder to read the code
* Comment where if, else, switch and main ends, helps confusion
* Should always comment at the start of every piece of code;  
  /\* Second week lab.(2.3)  
  The volume of a circle  
  Author; Aaron Byrne C15709609  
  Date; 29/09/2015

\*/

## Delimiters

|  |  |
| --- | --- |
| Name | Delimiter |
| Short or Int | %d |
| Long | %ld |
| Float | %f |
| Double | %lf |
| Char | %c |

# Week Two

## Arithmetic Operators

* int v1, v2, v3, v4 = 0 Only v4 is equal to 0
* int v1= v2= v3= v4 = 0 All variables are equal to 0

|  |  |
| --- | --- |
| Name | Operator |
| Addition | + |
| Subtraction | - |
| Multiplication | \* |
| Division | / |
| Modulus | % |

* Modulus, gives the remainder
  + v1 = 10  
    v2 = v1 % 2  
    Answer is v2=0
  + v1 = 10  
    v2 = v1 % 3  
    Answer is v2 = 1 (1 is the remainder when 3 is dived in 10)

## Increment and Decrement

* Num1 = Num1 + 1;
  + Is the same as Num1++; (no spaces)
* Num1 = Num1 - 1;
  + Is the same as Num1--; (no spaces)
* Num1 = Num1 + 2;
  + Is the same as Num1+=2; (no spaces)
* Num1 = Num1 - 2;
  + Is the same as Num1-=2; (no spaces) Pre and Post increment and decrement

#include <stdio.h>

main ()

{

int v1, v2, v3, v4;

var1 = var2 =1;  
var3 = var1++; Post  
var4 = ++var2; Pre

printf (“Var1 is %d, var2 is %d\n”, var1, var2);

printf(“var3 is %d, var4 is %d\n, var3, var4);

}//end main

Screen Outputs. Var1=2, Var2=2, Var3=1 and Var4=2

* Post
  + Done after the assignment
* Pre
  + Done before the assignment.

## Operator Priorities

|  |  |
| --- | --- |
| Operator | Priority |
| Brackets | Highest |
| Unary Minus (4- -2) | High |
| %, \*, / | Lower |
| +, - | Lowest |

## Type Conversion and Casting

# Week Three

## Single character input

/\*Program to input a single character\*/

*#include <stdio.h>*

*main ()*

*{*

*char ch;*

*printf (“Enter any character\n”) //It recognises any key as character i.e. Enter, spacebar and tab*

*scanf (“%c”, &ch); // scanf (“%1s”, &ch); // 1-character string. Won’t allow enter or tab or spacebar*

*Printf (“\n You entered %c”, ch); //delimiter stays the same*

*}//end main*

* getchar ()
* putchar ()

/\*Program to input a single character using getchar () and putchar (), much more efficient for single character. Not used for int or float\*/

*#include <stdio.h>*

*main ()*

*{*

*char ch;*

*printf (“Enter any character\n”) //Enter the character*

*ch=getchar (); //like scanf ();*

*printf (“\n You entered”); //display the character, space is for the character input*

*putchar(ch);*

*}//end main*

## Pitfalls

Ampersand

* int num1, num2;  
  scanf(%d%d,&num1,num2); //Ampersand
* int num;  
  scanf(“Enter a number\n %d”, &num); // scanf is reading from input. Only a delimiter(s) can be inside scanf

# Week Four Control Statements

if statement  
if (condition)  
{  
stament1;  
  
  
statment5;  
}

*Int balance = 0;*

*Printf (“Enter a balance\n”);*

*Scanf (“%d, balance);*

*If(balance<1)*

*{*

*Print(“insufficient funds”);  
}//end if*

### Operator Meaning

*//if statements without brackets not recommended.*

*If(balance<1)*

*Print (“no money”);*

*Print(“hello world”);  
//it will print all if statement is true if its false it will only skip the first statement.*

== Equivalent to  
 != not equivalent to   
 < Less Than  
 > Greater Than  
 <= less than or equal  
 >= greater than or equal to

*#include<stdio.h>*

*Main ()*

*{*

*Float bank\_balance;*

*bank\_balance=0*

*Printf (“enter your bank balance \n”);*

*Scanf (%f, &bank\_balance*

*If (bank\_balance <0);*

*{*

*printf (“\n Account in the red”);*

*}//end if*

*If(bank\_balance>=0)*

*{*

*Printf (“\n Account in the black”);*

*}//end if*

*}//end main*

## If/ Else statement

*Int a;*

*Int b;*

*a=b=0;*

*if (a==0 && b==0)*

*{*

*If(b==0)*

*{*

*Printf (“a and b are both zero”)*

*{//end if*

*}//end if*

If (condition)

{

Statements

}//end if

Else

{

Statements  
}//end else

*int num1=0;  
int num2=0;  
printf(“Enter your two numbers\n”);  
scanf(%d %d”,&num1,&num2);*

*if(num1>0)*

*{*

*If(num2>5)  
{  
 printf(“\n you entered a number greater than 5”);  
}//end if*

*Else  
{  
 printf(“\n you entered a number between 0-5”);*

*}//end else  
}//end 1st if*

*#include<stdio.h>*

*Main ()*

*{*

*Float bank\_balance;*

*bank\_balance=0*

*Printf (“enter your bank balance \n”);*

*Scanf (%f, &bank\_balance*

*If (bank\_balance <0);*

*{*

*printf (“\n Account in the red”);*

*}//end if*

*else*

*{*

*Printf(“\n Account in the black”);*

*}//end else*

*}//end main*

## Logical operations

|  |  |
| --- | --- |
| Operator | Meaning |
| && | Logical and |
| || | OR |
| ! | Not |

*Int a;*

*Int b;*

*a=b=0;*

*if (a==0 && b==0)*

*{*

*printf (“Both a and b are zero”);*

*}//end if*

*Int a;*

*Int b;*

*a=0*

*b=1;*

*if (a==0 || b==0)*

*if(Condition)*

***{***

*Else if (condition)*

***{***

*Else*

***{***

***}****//end else* ***}****//end else if*

***}****//end if*

*{*

*printf (“an or b is zero”);*

*}//end if*

*Int a;*

*Int b;*

*a=0*

*b=1;*

*if (! (a==0))// if (a! =o)*

*{*

*printf (“a is not zero”);*

*}//end if*

## Switch Statement

Switch ()

/\*Switch example program\*/

#include<stdio.h>

main ()

{

char symbol;

printf (“enter a mathematical operator\n”);

scanf (“%1s”, &symbol);

switch(symbol)

{

Case’+’:

{

Printf (“you’ve entered a plus);

Break;  
 }

Case’-’:

{

Printf (“you’ve entered a minus);

Break;

}

Case’\*’:

{

Printf (“you’ve entered a multiply);

Break;

}

Case’/’:

{

Printf (“you’ve entered a divide);

Break;

}

Default:

{

Printf (“\n Entered an invalid symbol”);

Break;

}//end default

}//end switch

}//end main

# Week Six Data Structures

## Array

To create an Array: data\_type array\_name[size\_of\_array]; Also known as Number of elements in an array

E.g. int ages [100];

Int number [5]; five blocks of memory.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| [0] | [1] | [2] | [3] | [4] |

Arrays can only carry same data type.

Hard code; numbers [0] =2 //puts 2 into element 0 in the array

/\*program the uses and array

\*/

#include <stdio.h>

main ()

{

int number [5];

int I;

printf (“enter 5 numbers”); //asking user for data for the array

for (i = 0; i <5, numbers[i]);

{

scanf (“%d”, &numbers[i]);

}//end for

for (i = 0; i <5, i++);

{

Printf (“\n Element %d is %d \n”, I, numbers[i]);

}//end for

}//end main

In a situation where you try write outside the size of an array is called an “array out of bounds exception”

/\* program that uses and array and find the highest and lowest value\*/

#include <stdio.h>

main ()

{

int numbers [5]

int I;

int highest, lowest

printf (“enter 5 numbers”);

//enter into array

For (i=0; I,5; i++)

{

Scanf (“%d”, &numbers[i]);

}

//assume the highest and lowest value is in the [o] of the array

Highest=numbers [0];

Lowest=number [0];

//find the highest/lowest

For (i=0; i<5; i++)

{

If(highest> numbers[i])

{

Highest= numbers[i];

}//end if

If(lowest> numbers[i]);

{

Lowest=numbers[i];

}//end if

}//end for

Printf (“\n Highest value is %d”, highest);

Printf (“\n Lowest value is %d”, lowest);

}//end main

## Symbolic name

#define SIZE 10 //size all in upper case to differentiate between a variable and a symbolic name, no semicolon.

/\* program that uses and array and find the highest and lowest value\*/

#include <stdio.h>

**#define SIZE 10**

main ()

{

int numbers[**SIZE**];

int I;

int highest, lowest;

printf (“enter **%d numbers\n”, SIZE);**

//enter into array

For (i=0; i<**SIZE**; i++)

{

Scanf (“%d”, &numbers[i]);

}

//assume the highest and lowest value is in the [o] of the array

Highest=numbers [0];

Lowest=number [0];

//find the highest/lowest

For (i=0; i < **SIZE**; i++)

{

If(highest> numbers[i])

{

Highest= numbers[i];

}//end if

If(lowest> numbers[i]);

{

Lowest=numbers[i];

}//end if

}//end for

Printf(“\n Highest value is %d”, highest);

Printf (“\n Lowest value is %d”, lowest);

}//end main

# Week 8 Arrays continued

Don’t hard code in arrays!

Initialising the array. int my\_array [5] = {10, 11,12,13,14};

5 bits of memory in each section of the array. Continuous memory.

**If** int my\_array [5] = {10,11,12,13}; in the array it will be 10, 11, 12, 13, 14, 0

**If** int my\_array [5] = {**,**11,12,13}; in the array it will be 0, 11, 12, 13, 14, 0

**If** in my\_array [50] = {0}; all elements of the array will be zero.

## Two Dimensional Arrays

Int matrix [3][2] ; MATRIX

|  |  |
| --- | --- |
| Row 0 Colum 0 (1) | Row 0 Colum 1 |
| Row 1 Colum 0 | Row 1 Colum 1 |
| Row 2 Colum 0 | Row 2 Colum 1 (6) |

Matrix [0][0] =1;

Matrix [2][1] =6;

***/\* Program using a 2-D array***

***\*/***

***#include <stdio.h>***

***#define ROW 3***

***#define COL 2***

***Main ()***

***{***

***Int matrix[ROW][COL];***

***Int I, j;***

***//enter data into the array***

***For (i=0; i<ROW; i++) //I represent row and j represents the column***

***{***

***For (j=0; j<COL; j++)***

***{***

***Scanf (“%d”, &matrix[i][j]);***

***}//end inner for***

***}//end for***

***//display the data in the array***

***For (i=0; i<ROW; i++) //I represent row and j represents the column***

***{***

***For (j=0; j<COL; j++)***

***{***

***printf (“%d”, matrix[i][j]);***

***}//end inner for***

***}//end for***

***}//end main***

***Initialising 2-D Array***

***int my\_array [4][3] = {1,2,3,4,5,6,7,8,9,10,11,12};***

***Best way to initialise the array;***

***int my\_array[4][3] = {1,2,3,  
4,5,6,  
7,8,9,  
10,11,12  
};***

***int my\_array[4][3] = { ,2,3,  
4, ,,  
7,***

***}; Don’t do this***

***int my\_array [4][3] = {0};***

***For (i=0; i<ROW; i++) //fills array with 1’s***

***{***

***For (j=0; j<COL; j++)***

***{***

***My\_array[i][j] =1;***

***}***

***}***

**Multidimensional Arrays**

Int box [2][2][3]; **NOT ON EXAM**

# Week 9 Pointers

|  |
| --- |
| RAM |
| Operating System |
|  |
| Numbers 4 bits in size |
|  |
|  |
|  |
|  |

Memory map.

Memory address F123

When the os runs a program.exe file.

Memory address is always in hexadecimal.

F123 is the start of the memory address.

|  |  |  |
| --- | --- | --- |
| File table  Program name | variables that are in the program | Memory address |
| My\_prog.exe | Numbers | F123 |

***/\* program to display the address of variables \*/***

***#include<stdio.h>***

***main ()***

***{***

***int var1;  
 int var2;***

***var1=1;  
 var2=’A’;***

***printf(“var1 has a value of %d and is stored at %p\n”, var1, &var1);  
 printf(“var2 has a value of %c and is stored at %p\n, var2, &var2);***

|  |
| --- |
| RAM |
| Operating System |
|  |
| var1 (1) |
|  |
|  |
| var2 (A) |
|  |

***}//end main***

%p is used to display the memory address

F123

F54C

Pointer variables

A variable stores the address of some other variable in the same program.

What it looks like; data\_type \*pointer\_variable\_name; e.g. int \*ptr;

Data type is telling the OS that the pointer is going to store a int variable.

***/\*Program that uses pointer variables  
 \*/***

***#include <stdio.h>***

***main ()***

***{***

***int var1;  
 int var2;  
 int \*ptr1;  
 int \*ptr2;***

***var1=1;  
 var2=’A’;  
 ptr1= &var1;  
 ptr2= &var2;  
  
 printf(“ptr1 contains %p\n, ptr1);  
 printf(“ptr2 contains %p”, ptr2);***

***}//end main***

|  |
| --- |
| *RAM* |
| *OS* |
|  |
| */////// 1 /////////* |
|  |
| */////// A /////////* |
|  |
| */////// F123 /////////* |
|  |
| */////// F24F /////////* |
|  |

**F123  
 var1**

**F24F  
 var2**

**F359**

**F461  
 ptr1  
  
  
  
 ptr2**

## The indirection operator (\*)

The indirection operator is used to access the contents at a variable whose address is stored in a pointer.

***/\* program to demo the indirection operator***

***\*/  
  
main()***

***{***

***int var;  
int \*ptr;***

***var1=1;  
ptr=&var  
  
printf(“ptr contains %p\n”, ptr); //prints ptr contains F123  
printf(“\*ptr contains %d\n”, \*ptr); // prints \*ptr contains 1***

***}//end main***

# Week 10 Pointers and Arrays

//creating the variables

|  |
| --- |
| *RAM* |
| *OS* |
| *var* |
| *F123 /////// 5 /////////* |
|  |
| */////// /////////* |
| *ptr* |
| *F456 /////// F123 /////////* |
|  |
| */////// /////////* |
|  |

***int var;  
int \*ptr;//not the indirection operator***

//anything after here with a \* is and an indirection operator

***var1=1;  
ptr=&var***

***var=5***;//puts 5 into var while overwriting 1.

***\*ptr=5****;//does the same as above, put 5 into where the address of ptr is pointing to.*

***int a, b, c;  
int \*ptr1;  
int \*ptr2; //these are variable not pointers***

***a=10;  
b=5;  
ptr1=&a;//puts address of a into ptr1. Pointer 1 is pointing to a  
ptr2=&b;//puts address of b into ptr2.***

***c=\*ptr1/\*ptr2;*** // go to the address of a&b and use what is in it. Is equal 10/5.

//will not compile because of /\*. Fix it by putting a space or brackets.

## Pointer and Arrays

In C, the name of an array is equivalent to the address of the 1st element.

Array will be assigned a contiguous block of memory. Never be broken into smaller pieces.

|  |
| --- |
| *RAM* |
| *OS* |
| *my\_array* |
| *F123 /////// /////////* |
| *F127 \\\\\\\ \\\\\\\\* |
| *F131 /////// /////////* |
| *F135 \\\\\\\ \\\\\\\\* |
| *F139 /////// /////////* |
|  |
|  |
|  |

e.g. ***in my\_array [5];*** //uses 20 bytes of memory

***&my\_array[0] ≈ my\_array***

***printf (“%p”, &my\_array [0]);***

***printf (“%p”, my\_array);*** //both printf statement are the same.

***&my\_array [0] ≈ my\_array***

***&my\_array [1 ≈ my\_array+1***//both are the same. Plus, one is saying move one element to the next element.

\*my\_array=1;   
 my\_array[0]=1;//same thing assigns 1 to the first element in the array.

**\*\*\*\*array\_name[i] *≈* \*(array\_name +i) \*\*\*\*\*\***

**\*\*\*\*\*Subscript notation *≈* pointer notation\*\*\*\*\*\***

|  |  |
| --- | --- |
| **Subscript notation** | **pointer notation** |
| /\* accessing an array using subscript notation  \*/  #include<stdio.h>  #define SIZE 5  main ()  {  Int my\_array[SIZE};  Int I;  //enter data into array  for (i=o; i<SIZE; i++)  {  scanf (“%d”, &my\_array[i]);  }  for (i=o; i<SIZE; i++)  {  printf (“%d\n”, my\_array[i]);  }  }//end main | /\* accessing an array using pointer notation  \*/  #include<stdio.h>  #define SIZE 5  main ()  {  Int my\_array[SIZE};  Int I;  //enter data into array  for (i=o; i< SIZE; i++)  {  scanf (“%d”, &\*(my\_array +i));  }  for (i=o; i<SIZE; i++)  {  printf (“%d\n”, \*(my\_array + i));  } //i moves to the next element in the array  }//end main |

Don’t forget the brackets in scanf ***(“%d”, &\*(my\_array +i));***

Without the brackets your program won’t run. ***\*my\_array+1;***

# Week 11 Dynamic Memory Allocation(DMA)

Allocate memory at run time.  
Has to be a contiguous block of memory.

## Malloc

ptr=malloc(size)  
pointer to the start of the memory block.  
size of the allocated memory blocks in bytes.

e.g. int \*ptr;

ptr=malloc(40);

|  |
| --- |
| *RAM* |
| *OS* |
| *no\_els* |
| *F123 /////// 5 /////////* |
| *no bytes* |
| *F36B \\\\\\\ 20 \\\\\\\\* |
| *i* |
| *F39A /////// /////////* |
| *ptr* |
| *F44D \\\\\\\ F565 \\\\\\\\* |
|  |
|  |
| *F565 20 bytes of memory ||||||||||||||||||||||||||||||||* |
| *||||||||||||||||||||||||||||||||* |
| *||||||||||||||||||||||||||||||||* |
| *||||||||||||||||||||||||||||||||* |
| *||||||||||||||||||||||||||||||||*  *||||||||||||||||||||||||||||||||*  *||||||||||||||||||||||||||||||||*  *||||||||||||||||||||||||||||||||* |
|  |
|  |
|  |

***/\* Program to demonstrate dynamic memory allocation***

***using malloc(). \*/***

***#include <stdio.h>***

***#include <stdlib.h>***

***main()***

***{***

***int \*ptr;***

***int no\_els, no\_bytes, i ; //no\_els = number of elements***

***printf( "Enter the number of elements " ) ;***

***scanf( "%d", &no\_els ) ;***

***/\* Calculate the number of bytes required by the array. \*/***

***no\_bytes = no\_els \* sizeof(int) ; //sizeof is a function in stdlib.h***

***/\* Allocate the memory. \*/***

***ptr = (int \*) malloc( no\_bytes ) ;***

***//check if successful***

***if ( ptr == NULL )***

***{***

***printf( "cannot allocate memory" ) ;***

***}***

***else***

***{***

***/\* Enter a value for each element of the array. \*/***

***for ( i=0; i<no\_els; i++ )***

***{***

***printf( "Enter element %d ",i ) ;***

***scanf( "%d", & \*(ptr+i )) ;***

***}***

***/\* Display the element values just entered. \*/***

***for ( i=0; i<no\_els; i++ )***

***printf( "Element %d is %d\n", i, \*(ptr +i) ) ;***

***free( ptr ) ; /\* Free the allocated memory. \*/***

***} //end else***

***} //end main***

## Calloc

***/\* Program to demonstrate dynamic memory allocation***

***using calloc(). \*/***

***#include <stdio.h>***

***#include <stdlib.h>***

***main()***

***{***

***int \*ptr;***

***int no\_els, i ; //no\_els = number of elements***

***printf( "Enter the number of elements " ) ;***

***scanf( "%d", &no\_els ) ;***

***/\* Allocate the memory. \*/***

***ptr = (int \*) calloc( no\_els, sizeof(int) ) ;***

***//check if successful***

***if ( ptr == NULL )***

***{***

***printf( "cannot allocate memory" ) ;***

***}***

***else***

***{***

***/\* Enter a value for each element of the array. \*/***

***for ( i=0; i<no\_els; i++ )***

***{***

***printf( "Enter element %d ",i ) ;***

***scanf( "%d", & \*(ptr+i )) ;***

***}***

***/\* Display the element values just entered. \*/***

***for ( i=0; i<no\_els; i++ )***

***printf( "Element %d is %d\n", i, \*(ptr +i) ) ;***

***free( ptr ) ; /\* Free the allocated memory. \*/***

***} //end else***

***} //end main***

## Realloc

***/\* Program to demonstrate dynamic memory allocation***

***Using calloc() and reallocating the memory somewhere else***

***Ptr= realloc(ptr, newsize); newsize = total size of new allocated memory***

***\*/***

***#include <stdio.h>***

***#include <stdlib.h>***

***main()***

***{***

***int \*ptr;***

***int no\_els, i ; //no\_els = number of elements***

***int no\_bytes;***

***int \*temp\_ptr; // so realloc doesn’t overwrite ptr completely***

***printf( "Enter the number of elements " ) ;***

***scanf( "%d", &no\_els ) ;***

***/\* Allocate the memory. \*/***

***ptr = (int \*) calloc( no\_els, sizeof(int) ) ;***

***//check if successful***

***if ( ptr == NULL )***

***{***

***printf( "cannot allocate memory" ) ;***

***}***

***else***

***{***

***printf( "Enter element %d ",i ) ;***

***/\* Enter a value for each element of the array. \*/***

***for ( i=0; i<no\_els; i++ )***

***{***

***scanf( "%d", & \*(ptr+i )) ;***

***}***

***/\* Display the element values just entered. \*/***

***for ( i=0; i<no\_els; i++ )***

***{***

***printf( "Element %d is %d\n", i, \*(ptr +i) ) ;***

***}***

**//changing the size of the memory block**

**printf(“Enter the new no of numbers \n”);**

**scanf(“%d”, &no\_els);**

**//Calculate the size of the new memory block**

**No\_bytes= no\_els \* sizeof(int);**

**//Allocate the new memory or change the size of memory**

**temp\_ptr = ptr; //temp\_ptr used to copy contents of ptr so realloc won’t get rid of the data in ptr**

**ptr=realloc(ptr, no\_bytes);**

***if ( ptr == NULL )***

***{***

***printf( "cannot allocate memory" ) ;***

***}***

***else***

**{**

***/\* Enter a value for each element of the array. \*/***

***for ( i=0; i<no\_els; i++ )***

***{***

***scanf( "%d", & \*(ptr+i )) ;***

***}***

***for ( i=0; i<no\_els; i++ )***

***{***

***printf( "Element %d is %d\n", i, \*(ptr +i) ) ;***

***}***

***} //end else***

**free(ptr);**

***} //end else***

***} //end main***

# Week 12 Review Week

\*DMA\*

Question 3[a] (2013 Exam Paper)

Write a program that dynamically allocates memory for ten floating point numbers. Using this memory enter the ten numbers and display these on separate lines.

***main()***

***{***

***float \*ptr;  
 int i;***

***//allocate memory***

***ptr=(float\*)calloc(10, sizeof(float));***

***//check if successful***

***If(ptr==NULL)***

***{***

***printf(“Failed to allocate memory”);***

***}***

***else***

***{***

***For(i=0; i<10; i++)***

***{***

***Scanf(“%f”, &\*(ptr+i));***

***}***

***for(i=0;i<10;i++)  
 {***

***printf(“%f\n”, \*(ptr+i));***

***}***

***free(ptr);***

***}//end else***

***}//end main***

Question 3 (b)

***//Change the size of the allocated memory***

***Ptr=realloc(ptr,(10\*2)\* sizeof(float));***

***if ( ptr == NULL )***

***{***

***printf( "cannot allocate memory" ) ;***

***}***

***else***

***{***

***/\* Enter a value for each element of the array. \*/***

***for ( i=0; i<no\_els; i++ )***

***{***

***scanf( "%d", & \*(ptr+i )) ;***

***}***

***for ( i=0; i<no\_els; i++ )***

***{***

***printf( "Element %d is %d\n", i, \*(ptr +i) ) ;***

***}***

***} //end else***

***free(ptr);***

***} //end else***

***} //end main***

Semester Two

# Week One Functions

Functions are efficient to re-use code.

Examples of functions. Built in C functions

* Main().
* Printf()
* Scanf()
* If()
* While
* Do-while
* For

Functions can’t be one of the built in functions. General rule of thumb give function relatable to what it does.

*/\* Program that uses a function\*/*

*#include <stdio.h>*

*//Declare function prototype*

*void stars(void); // void = no data being passed to the function stars.*

*// void = return type*

*main()*

*{*

*printf(“before function call”);*

*//call function stars()*

*stars();*

*printf(“\n After function call”);*

*}//end main*

*/\* implement function stars()*

*void stars()*

*{*

*int i; //local to stars function.*

*for (i=o; i<5; i ++)*

*{*

*printf(“\*”);*

*} //end for()*

*}end stars()*

*/\* program that uses two functions*

*\*/*

*#include <stdio.h>*

*//prototypes*

*void stars(void);  
void numbers(void);*

*main  
{  
 printf(“inside main() \n”);*

*//call function stars  
 stars();  
 printf(“\n back inside main()”);  
}//end main*

*/\* implement function stars*

*\*/*

*void stars()  
{  
 int i;  
 for(i=o; i<5; i++)  
 {*

*printf(“\*”);*

*}//end for*

*/\*call functions numbers(). Can call functions within functions. Do not nest too deep. Do not exceed three\*/*

*numbers();*

*}//end stars()*

*/\* implement numbers()*

*\*/*

*void numbers()*

*{*

*int i=0;*

*while(i<11)  
 {  
 printf(“%d \n”, i);*

*i++;*

*}//end while()*

*}//end numbers*

## Passing parameters

*/\* program that uses two functions and* ***passes parameters***

*\*/*

*#include <stdio.h>*

*//prototypes*

*void stars(****int, char****); //compilers needs to know what the data type is not the name  
void numbers(int[]); //passing down arrays.*

*main  
{*

***int input=0;***

***char mychar;*** *printf(“inside main() \n”);*

***printf(“Enter a number \n”);***

***scanf(“%d”, &input);#***

***printf(“Enter a Character \n”);***

***scanf(“%d”, &mychar);***

*//call function stars  
 stars(****input, mychar****);  
 printf(“\n back inside main()”);  
}//end main*

*/\* implement function stars*

*\*/*

*void stars(****int num, char mychar****) //int num and char my char are parameter variables  
{  
 int i;  
 for(i=o; i<****num****; i++)  
 {*

*printf(“%c”, mychar);*

*}//end for*

*/\*call functions numbers(). Can call functions within functions. Do not nest too deep. Do not exceed three\*/*

*numbers();*

*}//end stars()*

*/\* implement numbers()*

*\*/*

*void numbers()*

*{*

*int i=0;*

*while(i<11)  
 {  
 printf(“%d \n”, i);*

*i++;*

*}//end while()*

*}//end numbers*

Most common mistake using parameter. The order you specify in the prototypes must be the same order in the code, it won’t compile. You can have multiple parameters. You can pass down arrays.

## Return types

//prototype

float average(int,int,int)

function will return a float.

**/\* Program to demonstrate the use of the return statement. \*/**

#include <stdio.h>

//prototype

int highest(int,int);//return type

main()

{

int high=0;

int num1, num2;

printf("Enter two different numbers\n");

scanf("%d", &num1);

scanf("%d", &num2);

//call function highest()

high=highest(num1, num2);

printf("\n the highest of %d and %d is %d", num1, num2,high);

}//end main

//implement highest

int highest(int val1, int val2)

{

if(val1>val2)

{

return(val1); // will not execute code after you use a return.

}

else

{

return (val2);

}

}//end function

## Passing an argument by Value.

/\* First way is to Pass by value only passes a copy. Passes a copy to a function and The copy only changes\*/

#include<stdio.h>

//prototype

void fxn(int);

main()

{

int main\_arg=1;

printf("main\_arg is %d", main\_arg);

//call function fxn() pass a copy of the parameter

fxn(main\_arg);

printf("\nmain arg is %d",main\_arg);

}//end main

//implement function fxn()

void fxn(int fxn\_arg)

{

printf("\nfxn\_arg is %d", fxn\_arg);

fxn\_arg++;

}//end function fxn()

## Passing an argument by reference

/\* passing an argument by reference is when you pass the address of the parameter to the function\*/

#include<stdio.h>

//prototype

void fxn(int \*);

main()

{

int main\_arg=1;

printf("main\_arg is %d", main\_arg);

//call function fxn() pass a copy of the parameter

fxn( &main\_arg);

printf("\nmain arg is %d",main\_arg);

}//end main

//implement function fxn()

void fxn(int \*fxn\_arg)

{

printf("\nfxn\_arg is %d", \*(fxn\_arg) );

(\*fxn\_arg)++;

}//end function fxn()

/\* swapping the contenets of two variables using pass by reference.\*/

#include<stdio.h>

//prototype

Void swap(float \*, float \*);

main()

{

float num1, num2;  
 printf(“Enter two numbers\n”);

scanf(“%f”, &num1);  
 scanf(“%f”, &num2);

//swap the contents of the variables

Swap(&num1, &num2);

Printf”\n The variables swapped are %.1f and .1f%\n”, num1, num2);

}//end main

/\* implement swap()

void swap(float \*ptr1, float\*ptr2)

{ //temp variable used for swapping  
 float tmp;

tmp = \*ptr1;  
 \*ptr1 = \*ptr2;  
 \*ptr2 = tmp;  
}//end swap()

# Week 3 Passing Arrays to a Function

## /\*passing a 1-d array to a function\*/

#include<stdio.h>

//prototype

Int sum\_array(int[], int);//second int means the number of elements are int.

// you can swap int[] with int\*, it will work.

main()

{

Int values[5]={1,2,3,4,5};  
 int sum=0;

//get the sum of the array

Sum = sum\_array(values,5); //because it is an array its always pass by reference.

printf(“\n the sum of the array is %d”,sum);

}//end main

/\* implement sum\_array

int sum\_array(int temp\_array[], int size) // temp\_array = values, size = 5.

// if we change int[] to \*int we must change temp\_array to int\*temp\_array

{

int i; total;

for(i=0; i<size; i++)

{

total=total+temp\_array[i];// as above you will need to change temp\_array[] to  
\*(temp\_array+i)

}

return(total);

}//end sum

## /\*Passing a 2-d array \*/

#include <stdio.h>

//prototype

Int sum\_array(int [][2], int); //when you pass a multidimensional array you have to specify the number of columns. The only one you can leave blank is the no of rows.  
 has been a question on past paper.

main  
{

int values[3][2]={6,7,

4,9,

3,1

};

int sum;

//calling the function.

sum=sum\_array(values,3);

printf(“\n The sum of the array is %d”, sum);

}//end main()

//implement sum\_array()

int sum\_array(int matrix[][2], int no\_of\_rows)

{

int total=0, i,j;

for(i=0;i<no\_of\_rows; i++)

{

for(j=0; j<2; j++)

{

total=total+matrix[i][j];

}

}

return(total);

}//end sum\_array

# 4 Storage classes

## Auto

* All variables by default are auto variables
* They are also known as “local” variables
* An auto variables scope or visibility is only within the function they are declared
* Memory for an auto function variable is freed automatically when the function ends in which they are declared
* Example;
  + void fxn()  
    {  
    auto int num=0; // auto is not needed. They are default treated as auto.  
    auto float my\_array[10];  
    }

## Static

* static variables are similar to auto variables in the they are “local” i.e their scope is within the function they are declared
* Key difference is that static variables are **NOT** freed when the function ends. They retain their data.
* Static variables are created and allocated memory in RAM once.
* Example
  + /\* Program that uses a static variable\*/  
    #include <stdio.h>  
    //prototype  
    void fxn(void)  
    main()  
    {  
    int I;  
    //call fxn() five times  
    for(i=0;i<5;i++)  
    {  
    fxn();  
    }//end for  
    }//end main  
    }

//implement fxn()  
void fxn()  
{  
int auto\_var =0;  
static int static\_var=0; // ignored after its created on 2 loop on.  
auto\_var++;  
static\_var++;  
printf(“\n auto\_var is %d static\_var is %d”, auto\_var, static\_var);  
}//end fxn()

At the end of the program the print out will be auto\_var is 1 static\_var is 5.

## Extern

* Extern variables are ones that are typically declared in other programs but can be accessed by another program.
* Sometimes referred to as “global” variables.
* Example; const float pi=3.14;// constant are ok if they are used for the likes of things that are constant. But Extern isn’t something that should be used in Real Life.
* Example code; will not compile
  + /\* program that uses an Extern variable\*/  
    #include <stdio.h>  
    #include “my\_file.h” /\* header file which is a c program. It is called .h instead of .c there is no main in the header file.\*/  
    //prototype  
    void fxn(void);  
    main()  
    {  
    printf(“\n inside main()”);  
    //call fxn()  
    fxn();  
    }  
    /\*implement fxn() \*/  
    void fxn()  
    {  
    extern int value; /\* looks in the header file for the variable. Doesn’t create a new variable. It is already created as a global variable in the header file. Is not required but it makes it easier for everyone to read.\*/   
    value ++;  
    printf(“\n Number is %d”, value);  
    }//end fxn()

## Register

* Register variable is one whose memory is allocated inside the CPU and not RAM.
* If memory cannot be allocated in the CPU, the memory will be allocated by default in RAM.
* Register int i; // runs faster because it is in the CPU and not RAM
* Example; register int i;  
   for(i=0;i<100;i++)  
   {  
   }

## Mathematical Functions

#include <math.h>

Examples of functions;

Cos(x);  
Tan(x);  
Sin(x);

The x must be radians and the return type must be a double

Sqrt(x) [\\ square](file:///\\square) root of a number, x is a double and return type is a double.

Pow(x,y)\\ return type is a double. X is base and y is the power

Rand(); random number, return type interger.

Srand()n;\\ seed value, starting value. Best for randomisation.

# Strings

## String literals

## Long character strings

## Strings and Arrays

In C, a string is an array of characters the number of elements in the array is equal to the number of character in the string PLUS a null character.

Char greeting[6]={‘H’’E’’L’’L’’O’’\0’}; //need to include the null character.  
also   
char greeting[]=”Hello”;//automatically includes the null character.

## Printf and Strings

Printf (“%s”, greeting) prints hello and does not print the null character.

Printf(“%20s”, greeting) to right justify the print statement

Printf(“%-20s”, greeting) to left justify the string.

Printf(“%.3s”, greeting) prints the first three characters, right justified, in the array.

Printf(“%-.3s”, greeting) prints the first three characters, left justified, in the array.

## Scanf and Strings

Char name[21];

Printf(“Enter your first name\n”);

Scanf(“%s”, name);

Scanf(“%20s”, name); // only allows the user to enter in a string of 20 characters.

## Puts() Function

Put string = puts

Used to display the contents of a string (ie char array) to standard output.

Puts also goes to a new line automatically after displaying the string.

Doesn’t allow for manipulation of strings like printf

Puts(name);

Puts(“my\_string”); //hard coding

Puts(char array);

## Gets() Function

Get string =gets

Used to enter a string BUT it allows white space characters.

Wont allow you to enter more than is in the array.

Gets(name);

/\* Accessing individual elements of a character array(string)

\*/

#include <stdio.h>  
main()

{

Char greeting[6] = “Hello”;

Int I;

For (i=0;i<6I++)

{

Printf(“%c”, greeting[i]); //change to \*(greeting+i) for pointer notation.

}//end for

}//end main

## Initialise a String

1. Char greeting[6]={‘H’’E’’L’’L’’O’’\0’}; //need to include the null character.
2. char greeting[]=”Hello”;//automatically includes the null character
3. Assigning a string to a pointer.  
   char \*p;  
   p=”Hello”;

Char \*p=”hello”; //is the same.

## Program using pointer notation and strings

/\*Accessing a string that is assigned to a pointer. \*/

#include <stdio.h>

main()

{

char \*p = ”Hello”;

//Display the string

while(\*p != ’\0’) // to display each character in the string.

{

printf(“%c”, \*p);

p++;

}//end while

}//end main()

Char \*p =”some text”;  
char my\_string[10]=”more text”;

p = my\_string;// variable pointer. You can do this

my\_string=p;// array name or constant pointer. Cannot do this.

## String Functions

# include <string.h>

## Strlen()

Finds the string length.

Returns the number of characters excluding the null character.

char name1[]=”Sharon”;

char name[15]=”Mark”;

char \*name3 =”Patrick”;

int len;

len =strlen(name1);

printf(“%d %d %d %d”, strlen(name1),strlen(name2),strlen(name3), strlen(“Rob”), len);

\*NB the string should be correctly null terminated

Output: 6, 4,7,3,6.

## Strcpy

Copying a string.

Str(string1,string2); destination comes first, source comes second.

This copies string2 into string1.

Compiler wont check the next two it is up to the programmer.

* The source must be NULL terminated.
* You must ensure that the destination string is large enough to it the source string.

char name1[]=”Sharon”;

char name[15]=”Mark”;

strcpy(name2,name1);

puts(name2);

what is in the destination gets overwritten.

## Strcat()

Concatenating a string.

Strcat(string1,string2);

This concatenates (joins) string2 to the end of string1.

Destination string come first and source come second.

Str1[15]=”first & “;

Str2[]=”second”;

Strcat(str1,str2);

Copies what is in str2[] and puts it in str1.

## Strcmp()

String comparison.

Strcmp(string1,string2);

Strcmp returns an integer 0 if the strings are identical.

Both strings need to be NULL Terminated.

Case Sensitive.

Char str1[20];

Char str2[20];

Printf(“Enter two strings\n”);

Scanf(“%s”, str1); gets(str1);

Scanf(“%s”, str2); gets(str2);

If(strcmp(str1,str2)==0)

{

Printf(“Both strings are identical”);

}

Else

{

Printf(“The strings are not identical”);

}

## Strncmp

Strncmp(str1,str2,n);

N is the number of elements you want the program to compare.

## Converting numeric strings to numbers.

#include<stdlib.h>

char str[]=’123’;

int int\_num

long long\_num; it has double memory and range 28

double double\_num; it is double the memory.

Int\_num=atoi(str);// convert into int

Long\_num=atol(str);

Double\_num=atof(str);

## Array of strings

/\* an array of strings can only be done using pointers.

Each character in the string is contiguous but the strings could be anywhere in memory.

The array just points to the address of the first character of the strings.\*/

#include <stdio.h>

main()

{

Char \*months[12] ={“Jan”, “”Feb”, “Mar”, “Apr”,

“May”, “Jun”, “Jul”, “Aug”,

“Sep”, “Oct”, “Nov”, “Dec”

};

Int i;

Printf(”The months are \n”);

for(i-1;i<12;i++)

{

printf(“%s\n”, months[i[);

}\\end for

}\\end main

## Structures

Defining a structure or using a structure template

Struct student\_rec // struct is a keyword. student\_rec is a structure tag.  
{  
 char first\_name[11];  
 char last\_name[21];  
 int studentID;  
 int results[5];

//everything inside is call a structure member.  
}; //semicolon is very important.

Struct student\_rec student1, student2;

/\*program that uses a structure\*/

#inlcude <stdio.h>

//symbolic names to go here.

//structure template it is not a variable.

Struct student\_rec   
{  
 char first\_name[11];  
 char last\_name[21];  
 int studentID;  
 int results[5];

};

//any prototypes go here

main()

{

Struct student\_rec student1, student2;

int i;

//enter details

Printf(“Enter firstname\n”);

Scanf(%10s”, student.first\_name”);

Printf(“Enter Last name\n”);

Scanf(%20s”, student.last\_name”);

Printf(“Enter ID\n”);

Scanf(“%d”, &student1.studentID);

Printf(“Enter results\n”);

For(i=0;i<5;i++)

{

Scanf(%d”, student.results[i]”);

}//end for

//display students details

Student2=student1;//copies everything into student 2.

}//end main

Strycpy(student2.first\_name,”John”);//initialising strings.

Struct student\_rec

{

Char firstname[11];  
 char surname[21];  
 int studentID;  
 int results[5];

}

main()

{

Struct student\_rec student = { “John”,  
 “Smith”,  
 1234,  
 {60,89,48,77,56)  
 };

}//end main

/\* program showing how to initialise a structure \*/

#include<stdio.h>  
#define SIZE 5  
//structure template  
//function prototypes  
main()  
{  
 struct //no tag?  
 {  
 Char firstname[11];  
 char surname[21];  
 int studentID;  
 int results[5];

}student1,student2; //these two are variables

## Pointer and structures

Struct student\_rec

{

Char firstname[11];  
 char surname[21];  
 int studentID;  
 int results[5];

}

main()

{

Struct student\_rec student;

//pointers to structures

Struct student\_rec \*ptr;

//make ptr point to student

ptr=&student;

(\*ptr).studentID=1234; // ptr->studentID=1234;

Strcpy(ptr->firstname “John”); or strcpy(student.firstname, “John”);

}//end main

## Passing a structure to a function

#include <stdio.h>

//template

Struct student\_rec  
{  
 char firstname[11];  
 char surname[21];  
 int studentId;  
 int results[5]  
};  
//prototype  
void get(struct student\_rec \*);  
void display(struct student\_rec);  
main()  
{  
 struct student\_rec student;  
 struct student\_rec \*ptr;  
 ptr =&student;  
//call the get()/display();  
get(ptr);  
display(student);  
}

//get() using pass by reference  
void get (Struct student\_rec \*p)  
{  
int I;  
printf(“Enter details\n”);  
scanf(“%10s”, p->firstname);  
scanf(“%20s”,p->surname);  
scanf(“%d”, &p->studentID);  
for(i=0;i<5;i++)  
{

Scanf(“%d”, &->results[i]);

}//end for  
}//end get

//display pass by value. Only passes a copy.  
void display(struct student\_rec stu)  
{  
int I;  
printf(“Display details\n”);  
printf(“%s”, stu.firstname);  
printf”%s”, stu.surname);  
printf(“%d”, &p->studentID);  
for(i=0;i<5;i++)  
{

printf(“%d”, &->results[i]);

}//end for  
}//end display()

## Nested structures

Struct date  
{  
int day;  
int month;  
int year;  
}

Struct person  
{  
char firstname[11];  
char surname[21];  
struct date DOB;  
};

Main()  
{

Struct details person;  
struct person \*ptr;  
ptr=&student;

Printf(“Enter first name”\n);  
scanf(“%10s” student.firstname);

Printf(“Enter surname\n”);  
scanf(“20s”, student.surname);

Printf(“Enter date of birth”);  
scanf(“%d”, &student.DOB.day);  
scanf(“%d”, &student.DOB.month);  
scanf(“%d”, &student.DOB.year);

//reading into the nested structure using pointers

Printf(“Enter date of birth”);  
scanf(“%d”, &ptr->DOB.day);  
scanf(“%d”, &ptr->DOB.month);  
scanf(“%d”, &ptr->DOB.year);

}

## Structure header file.

#include<stdio.h>  
#include>structures.h>

main()  
{  
struct student, student1, student2;  
}

## Arrays of Structures

Struct student students[50];  
students[0].studentID=1234;

## Structures comparison

Struct shape

{

Int a;  
float b;  
char c;  
};  
  
struct shape s1,s2;  
  
if( (s1.a==s2.a) && (s1.b==s2.b) && (s1.c==s2.c)  
{  
printf(“Identical”);  
}

## typedef statement.

all lower case. Type defintiton.

This allows you to create a synonym for a data type.

typedef int\* INT\_PTR;

INT\_PTR p1; //p1 will be created as a int pointer variable.

## Pre-Processor

Source code -> pre-processor->pre-processor compiled code->fully compiled code

^^^^^^^^^^^

Pre-processor focuses on the # in the code.

#include <stdio.h> //tells the pre-processor to go to the default folder to find this header file  
#inlcude “my\_file.h” // search for this header file in the current folder where the .c file is first. Then it looks at the default folder.

#include “C:\my\_progs\my\_file.h” //don’t specify a path unless it is a design decision.

#define SIZE 5 //symbolic name. SIZE is a macro.  
#define PI 3.14  
#define END\_SENTENCE ‘.’  
#define NEWLINE ‘\n’  
#define DIGITS “1234”

#undef SIZE // undefines size.

Macro parameters

#include <stdio.h>  
#define SQUARE(N) (N)\*(N)

main()

{

int n=2;  
float f =5.5;  
int result =0;  
  
result=SQAURE(2);  
printf(“2 squared is %d”, result);  
}

# Past Exam Question.

## 2014 string question

Some strings in English are spelled the same way backwards.

e.g. kayak, radar, never odd or even

enter a string.  
check if the string is spelled the same way backwards.

Char my\_string[21];  
printf(“Enter a string”);  
scanf(“%20s”, my\_string); //don’t need & with strings

Char copy\_string[21];  
int i;  
//get string length.

for(i=0; j=20; i<21; j>-1; i++; j--)  
{  
 copy\_string[j] = my\_string[i];  
}

if( strcmp(my\_string, copy\_string) ==0);  
{  
 printf(“Same spelt backwards”);  
}  
else  
{  
 printf(“Not identical”);  
}

## 2012 Part B. Question 4. (out of 24 marks)

Part A, 8 marks explain the four data types.  
Part B, 16 marks.

/\* Calculate the area of a circle and square

\*/

#include<stdio.h>

//Prototypes

Float square\_area(int);  
float circle\_are(int):

main()

{

int length =0;  
 int radius=0;  
 float c\_area=0;  
 float s\_area=0;

printf(“please enter the square length\n”);  
 scanf(“%d”,&length);

printf(“\n Enter the circle radius \n”;  
 scanf(“%d”&radius);

//call functions

S\_area=square\_area(length);

c\_area=circle\_area(radius);

printf(“\n The Square area is %.1f”,s\_area);  
 printf(“\n The Circle area is %.1f”, c\_area);

}//end main

//implement the square\_area()

float square\_area(int sq\_length)

{

float area =0;  
area=sq\_length \*sq\_length;  
return(area);

}//end square\_area()

//implement circle\_area()

float circle(int cir\_radius)

{

float area =0;  
area=3.14 \*(cir\_radius\*cir\_radius);

}//end circle\_area()

# Exam Change

New format. Section A- 20 short questions compulsory (20\*2marks) =40%  
 Section B- Answer 2 out of 3 long questions (2\*30marks) =60%