

***** Unit-IV *****

----- Functions and Dynamic Memory Allocation -----

***** :Functions: *****

Function: A function is a self contained block of statements written to execute a specific task.

We define the functions to divide the large size programs into a smaller sub programs(functions). Each sub program can be a function.

Advantages:

By using functions we can improve the application development speed.

By using functions the programmer can identify and rectify(debug) the errors easily.

By using functions we can develop the applications in structured programming.

By using functions we can have the reusability.

We have 2 types of functions used in the C programming.

1. Predefined functions(Library Functions)

2. User defined functions

1. Predefined functions(Library Functions):

Predefined functions are developed by the C compiler developers in C language. These are available in the various header libraries. We can include the particular header library in the program to access the functions available under the header library.

Ex:

1. printf(),scanf(),.... are available in stdio.h

2. strlen(),strcpy(),strcat(),... are available in string.h

3. clrscr() available in conio.h

2. User defined functions:

These functions are developed by the programmer for the specific requirement.

We need to follow the following steps to define and execute the user defined functions.

Step 1: Function prototype declaration

Step 2: Function body definition

Step 3: Function calling

Step 1: Function prototype declaration:

The function prototype declaration contains the information about the function like function name,function parameters with datatypes and function return type

We can declare the function prototype to tell the compiler that we are going to define and use the user defined function

syntax:

```
returntype functionname(parameters with their datatypes);
```

Ex 1:

```
void addition(int a,int b);
```

Ex:2

```
float get_avg(int total);
```

=> In the above prototype functionname is the name of function(Ex: addition,get_avg).

=> We can declare the function prototype at the definition section of the program.

=> We need to follow all the rules for defining the identifiers for defining the function name.

=> The function must be followed by the parenthesis(open and close ()).

=> The function may or may not take any parameters.

=> If the function requires any parameters to pass then we need to pass the parameters with their datatypes.

=> If there is more than one parameter to be passed, then we need separate each parameter with comma(,) operator.

=> The function may not take the parameters if not required. If no parameters are required then we write the empty parenthesis with the function name.

=> If the function requires more than one datatype of parameter then we can pass different datatype's parameters.

=> The function may return or may not return the result to calling function. if it returns the result to calling function then we need to write the return type. when we have return type we mention the datatype of the value to be returned.

=> If the function returns a value to calling function then it returns only one value.

=> we can return the value using return keyword.

=> If the function won't return a value to calling function then we write void. void means nothing.

=> The function prototype declaration must be terminated with semi colon.

Step 2: Function body definition:

After declaring the function prototype we need to define the function body. Function body contains the statements to be executed within the function.

The function is written within the braces({ }) after the function name.

syntax:

```
returntype functionname(parameters with their datatypes)
{
statements
.....
.....
}
```

Ex:1

```
void addition(int a,int b)
{
int c;
c=a+b;
printf("Addition is %d\n",c);
}
```

Ex:2

```
float get_avg(int total)
{
float avg;
avg=total/3.0;
return avg;
}
```

=> When we define the function body we are not required write the semi colon at the end of function header(function signature).

=> The first line of function body is known as function header or function signature.

=> The function name should be same in the function prototype and function header.

=> The function parameters with their datatypes,sequence must be same in function prototype and function header.

=> The number of parameters of function prototype must match with the number of parameters in function header.

=> The return type in the function header must with the return type of function prototype.

=> The function prototype must match with the function header at function body definition.

Step 3: Function calling

Once the function body is defined, we need to execute the function by calling it.

We can call the function in two ways.

1. When the function does not have return type(void).
2. When the function having return type.

1. When the function does not have return type(void):

We can call the function when it does not have return type as follows.

syntax:

functionname(parameters);

Ex:1

```
int a,b;  
a=10;  
b=20;  
addition(a,b);
```

(or)

```
addition(10,20);
```

2. When the function having return type:

We can call the function when it has the return type as follows.

syntax:

```
returntype variable=functionname(parameters);  
int total=180;  
float result;  
result=get_avg(total);
```

(or)

```
result=get_avg(180);
```

=> When we call the function, we must terminate it with semicolon.

=> If the function has parameters then we need to pass that many parameters at the time function calling.

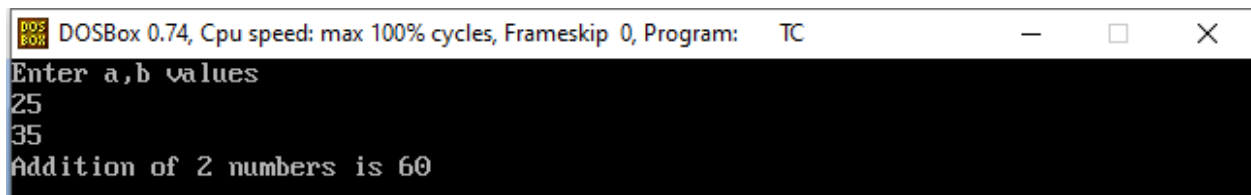
=> We can call the function either from the main function or from any other user defined function.

Write a program to demonstrate the user defined functions for addition of 2 numbers.

fun1.c

```
#include<stdio.h>
void addition(int a,int b);
void addition(int a,int b)
{
int c;
c=a+b;
printf("Addition of 2 numbers is %d\n",c);
}
main()
{
int a,b;
clrscr();
printf("Enter a,b values\n");
scanf("%d%d",&a,&b);
addition(a,b);
return 0;
}
```

Output:



```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter a,b values
25
35
Addition of 2 numbers is 60
```

calling function:

The function which calls the other function is known as calling function.
(in the above example main function is the calling function)

called function:

the function which called by the other function is known as called function.(int the above example addition function is the called function)

Note:

When the calling function encounters the function call statement then immediately the execution control will be transferred to the called function and executes the statements written inside it then immediately the execution control will be transferred back to the next statement of function call statement inside the calling function.

HW1:

Write a program to demonstrate the function for calculating the subtraction

HW2:

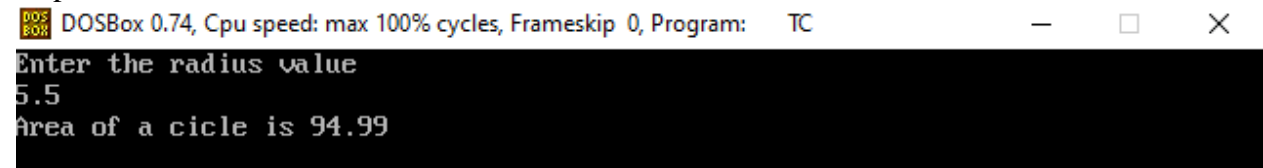
Write a program to demonstrate the function for calculating the multiplication

Write a program to calculate the area of a circle

funarea.c

```
#include<stdio.h>
void circle_area(float radius); /*prototype decleration*/
void circle_area(float radius)
{
    float ca;
    ca=3.14*radius*radius;
    printf("Area of a cicle is %.2f\n",ca);
}
main()
{
    float radius;
    clrscr();
    printf("Enter the radius value\n");
    scanf("%f",&radius);
    circle_area(radius);
    return 0;
}
```

output:



```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter the radius value
5.5
Area of a cicle is 94.99
```

HW1:

Write a program to calculate the area of a triangle using functions.

hint:

```
void triangle_area(float base,float height);
```

HW2:

Write a program to calculate the area of a rectangle using functions.

hint:

```
void rectangle_area(float length,float breadth);
```

Types of functions:

Based on the return types and parameters passing to the function we can classify the functions as follows.

1. function with return type and with parameters
2. function with return type and without (no)parameters
3. function without return type(void) and with parameters
4. function without return type(void) and (no)without parameters

Write a program to calculate the addition of 2 numbers by using functions with all categories

1. int addition(int a,int b);
2. int addition();
3. void addition(int a,int b);
4. void addition();

funadd1.c

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

```
int addition(int a,int b);
```

```
int addition(int a,int b)
```

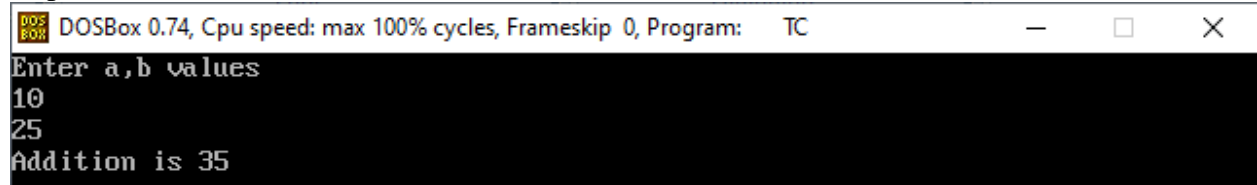
```
{  
int c;  
c=a+b;  
return c;  
}
```

```
main()
```

```
{  
int a,b,c;  
clrscr();  
printf("Enter a,b values\n");  
scanf("%d%d",&a,&b);  
c=addition(a,b);  
printf("Addition is %d\n",c);  
return 0;
```

```
}
```

output:

A screenshot of a DOSBox window titled "DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC". The window has a black background with white text. The text inside the window reads: "Enter a,b values", "10", "25", and "Addition is 35". The window has standard Windows-style title bar controls (minimize, maximize, close) on the right side.

```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter a,b values
10
25
Addition is 35
```

funadd2.c

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

```
int addition();
```

```
int addition()
```

```
{
```

```
int a,b,c;
```

```
printf("Enter a,b values\n");
```

```
scanf("%d%d",&a,&b);
```

```
c=a+b;
```

```
return c;
```

```
}
```

```
main()
```

```
{
```

```
int c;
```

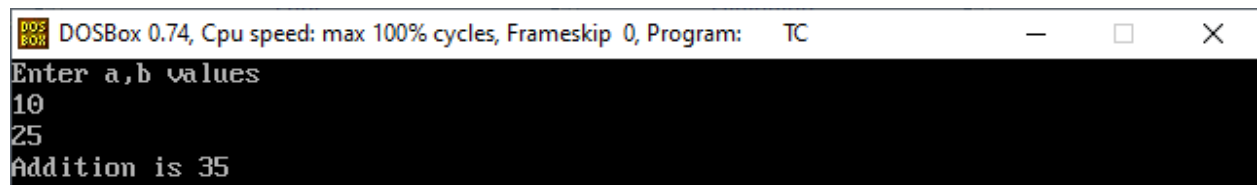
```
c=addition();
```

```
printf("Addition is %d\n",c);
```

```
return 0;
```

```
}
```

output:

A screenshot of a DOSBox window titled "DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC". The window has a black background with white text. The text inside the window reads: "Enter a,b values", "10", "25", and "Addition is 35". The window has standard Windows-style title bar controls (minimize, maximize, close) on the right side.

```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter a,b values
10
25
Addition is 35
```

funadd3.c

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

```
void addition(int a,int b);
```

```
void addition(int a,int b)
```

```
{
```

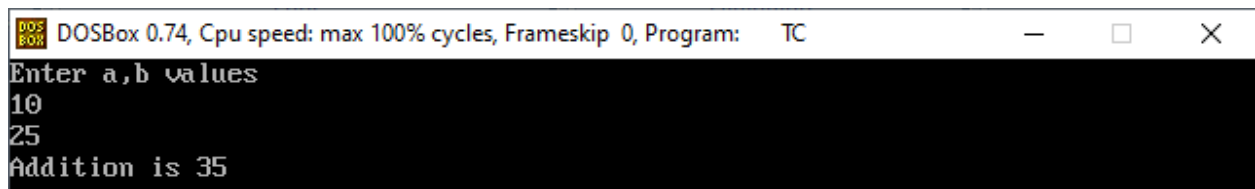
```
int c;
```

```
c=a+b;
```

```
printf("Addition of 2 numbers is %d\n",c);
```

```
}
```

```
main()
{
int a,b;
clrscr();
printf("Enter a,b values\n");
scanf("%d%d",&a,&b);
addition(a,b);
return 0;
}
output:
```



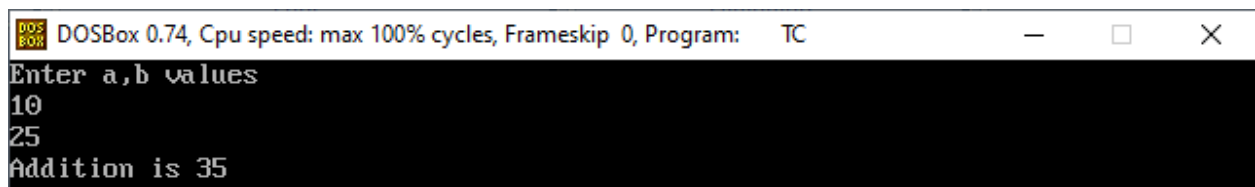
```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter a,b values
10
25
Addition is 35
```

funadd4.c

```
#include<stdio.h>
void addition();

void addition()
{
int a,b,c;
printf("Enter a,b values\n");
scanf("%d%d",&a,&b);
c=a+b;
printf("Addition is %d\n",c);
}
```

```
main()
{
clrscr();
addition();
return 0;
}
output:
```



```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter a,b values
10
25
Addition is 35
```

HW1:

Write a program to calculate the subtraction of 2 numbers using functions with all 4 categories

HW2:

Write a program to calculate the multiplication of 2 numbers using functions with all 4 categories

Write a program to calculate the the total marks, avg and grade of a student by using functions.(take the names of functions as get_total,get_avg and display_grade)

fungrade.c

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

```
int get_total(int sub1,int sub2,int sub3);  
float get_avg(int total);  
void display_grade(float avg);
```

```
int get_total(int sub1,int sub2,int sub3)  
{  
    int total;  
    total=sub1+sub2+sub3;  
    return total;  
}
```

```
float get_avg(int total)  
{  
    float avg;  
    avg=total/3.0;  
    return avg;  
}
```

```
void display_grade(float avg)  
{  
    if(avg>=90)  
    {  
        printf("Grade O\n");  
    }  
    else if(avg>=80 && avg<90)  
    {  
        printf("Grade A+\n");  
    }  
    else if(avg>=70 && avg<80)  
    {  
        printf("Grade A\n");  
    }  
    else if(avg>=60 && avg<70)  
    {  
        printf("Grade B+\n");  
    }  
}
```

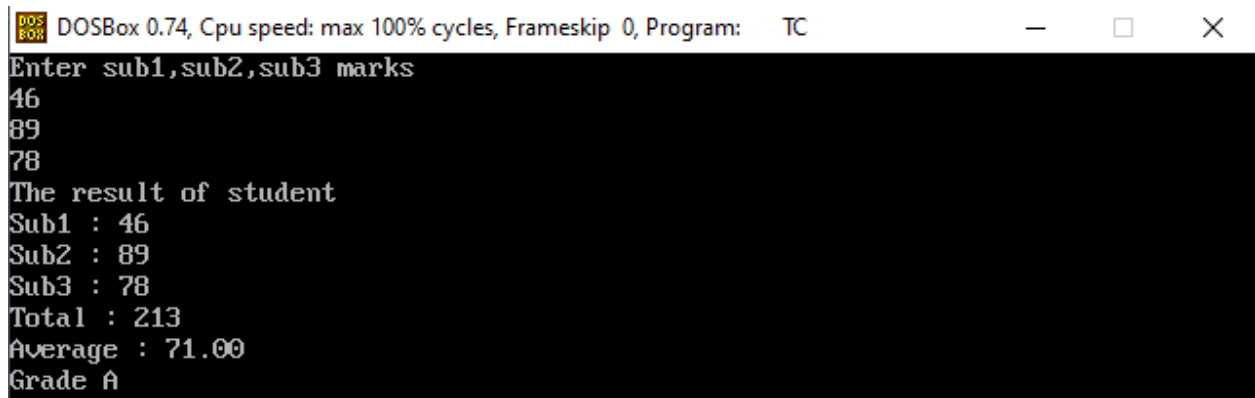
```

}
else if(avg>=50 && avg<60)
{
    printf("Grade B\n");
}
else if(avg>=40 && avg<50)
{
    printf("Grade C\n");
}
else
{
    printf("Failed\n");
}
}

main()
{
    int sub1,sub2,sub3,total;
    float avg;
    clrscr();
    printf("Enter sub1,sub2,sub3 marks\n");
    scanf("%d%d%d",&sub1,&sub2,&sub3);
    total=get_total(sub1,sub2,sub3);
    avg=get_avg(total);
    printf("The result of student\n");
    printf("Sub1 : %d\n",sub1);
    printf("Sub2 : %d\n",sub2);
    printf("Sub3 : %d\n",sub3);
    printf("Total : %d\n",total);
    printf("Average : %.2f\n",avg);
    display_grade(avg);
    return 0;
}

```

output:



```

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter sub1,sub2,sub3 marks
46
89
78
The result of student
Sub1 : 46
Sub2 : 89
Sub3 : 78
Total : 213
Average : 71.00
Grade A

```

Write a program to calculate the area of a circle, triangle and rectangle based on the shape selection using the functions with return type and with parameters

funarea1.c

```
#include<stdio.h>
float circle_area(float radius);
float triangle_area(float b,float h);
float rectangle_area(float l,float b);

float circle_area(float radius)
{
float ac;
ac=3.14*radius*radius;
return ac;
}

float triangle_area(float b,float h)
{
float at;
at=0.5*b*h;
return at;
}

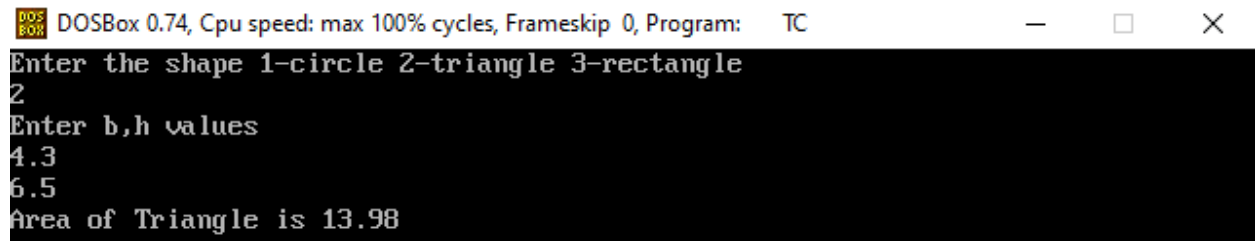
float rectangle_area(float l,float b)
{
float ar;
ar=l*b;
return ar;
}

main()
{
int shape;
float radius,b,h,l,ac,at,ar;
printf("Enter the shape 1-circle 2-triangle 3-rectangle\n");
scanf("%d",&shape);
switch(shape)
{
case 1: printf("Enter radius value\n");
scanf("%f",&radius);
ac=circle_area(radius);
printf("Area of circle is %.2f\n",ac);
break;
case 2: printf("Enter b,h values\n");
scanf("%f%f",&b,&h);
```

```

        at=triangle_area(b,h);
        printf("Area of Triangle is %.2f\n",at);
        break;
    case 3: printf("Enter l,b values\n");
            scanf("%f%f",&l,&b);
            ar=rectangle_area(l,b);
            printf("Area of Rectangle is %.2f\n",ar);
            break;
    default:printf("Choose correct shape and try again !!!\n");
    }
    return 0;
}
output:

```



```

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter the shape 1-circle 2-triangle 3-rectangle
2
Enter b,h values
4.3
6.5
Area of Triangle is 13.98

```

HW1:

Write a program to perform all the arithmetic operations by using functions with return type and with parameters then select the operation to perform using switch statement.(calculator application)

Local Variable:

A variable declared inside to any function or block is known as local variable.

The accessibility or scope of a local variable with in the block or function in which it is declared.

The life time of a local variable is until the execution control remains in the function or block in which the variable is declared.

Global Variable:

The varaibles declared outside to any function or block are known as the Global Variables.

The accessibility or scope of a global variable is with in the functions or blocks of the program.

The life time of global variable is until the execution of the complete program

Formal Parameters(Formal Variables/Formal Arguments):

The variables which are declared in the called function are known as Fromal Parameters.

Actual Parameters(Actual Varaibles/Actual Arguments):

The variables which are declared in the calling function are known as Actual Parameters.

Call-by-value(Pass-by-value) technique:

Write a program to swap 2 numbers using functions with the technique Call-by-value(Pass-by-value)

Swapping is interchange of values

a=10

b=20

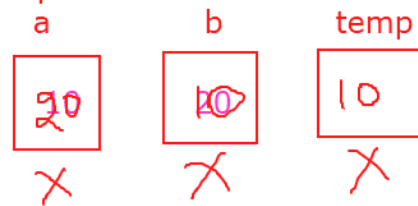
after swapping the values should be

a=20

b=10

```
void swap(int a,int b);  
void swap(10int a,20int b) ←  
{  
  int temp;  
  temp=a;  
  a=b;  
  b=temp;  
}  
  
main()  
{  
  int a,b;  
  a=10;  
  b=20;  
  printf("Before swap a %d b %d\n",a,b);  
  swap(a,b);  
  printf("After swap a %d b %d\n",a,b);  
  return 0;  
}
```

swap function



main function



call1.c

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

```
void swap(int a,int b);
```

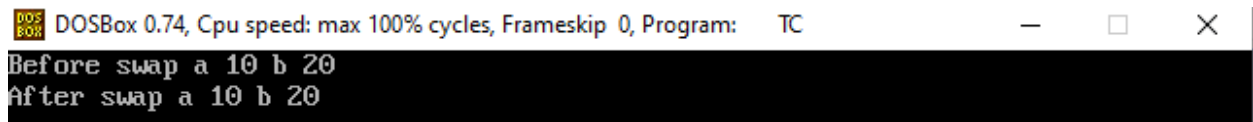
```
void swap(int a,int b)
```

```
{  
  int temp;  
  temp=a;  
  a=b;  
  b=temp;  
}
```

```
main()
```

```
{  
int a,b;  
a=10;  
b=20;  
printf("Before swap a %d b %d\n",a,b);  
swap(a,b);  
printf("After swap a %d b %d\n",a,b);  
return 0;  
}
```

Output:



```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC  
Before swap a 10 b 20  
After swap a 10 b 20
```

Note :

There is draw back in the call-by-value technique that the modification made on the formal parameters does not effect on the actual parameters, hence we cannot get the expected answer in the above program.

to overcome from this we can use the following techniques

1. call-by-reference
2. global variables

Write a program to swap 2 numbers using functions with the technique Call-by-Reference(Pass-by-Reference/Call-by-address/Pass-by-address)

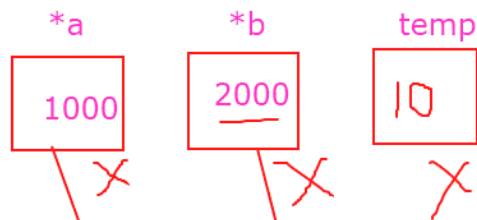
```

#include<stdio.h>
void swap(int *a,int *b);
void swap(int 1000*a,int 2000*b)
{
    int temp;
    temp=*a;
    *a=*b;
    *b=temp;
}

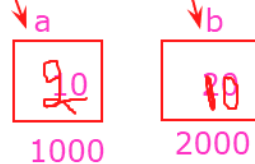
main()
{
    int a,b;
    a=10;
    b=20;
    printf("Before swap a %d b %d\n",a,b);
    swap(&a,&b);
    printf("After swap a %d b %d\n",a,b);
    return 0;
}

```

swap function



main function



call2.c

```

#include<stdio.h>
void swap(int *a,int *b);

void swap(int *a,int *b)
{
    int temp;
    temp=*a;
    *a=*b;
    *b=temp;
}

main()
{
    int a,b;
    a=10;
    b=20;
    printf("Before swap a %d b %d\n",a,b);
    swap(&a,&b);
    printf("After swap a %d b %d\n",a,b);
    return 0;
}

```

Output:

```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Before swap a 10 b 20
After swap a 20 b 10
```

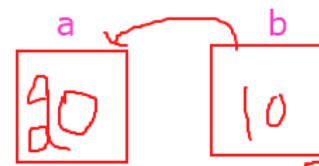
Note :

In the pass-by-reference technique using pointers we can do modification on actual arguments directly with the addresses, hence we get the expected result in the program.

Write a program to swap 2 numbers using functions with Global Variables.

```
#include<stdio.h>
int a,b;
void swap();
void swap()
{
    int temp;
    temp=a;
    a=b;
    b=temp;
}
main()
{
    a=10;
    b=20;
    printf("Before swap a %d b %d\n",a,b);
    swap();
    printf("After swap a %d b %d\n",a,b);
    return 0;
}
```

Global variables



swap function



call3.c

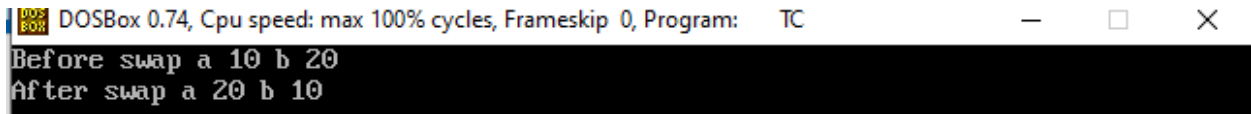
```
#include<stdio.h>
int a,b;
void swap();
void swap()
{
    int temp;
    temp=a;
    a=b;
    b=temp;
}
main()
{
    a=10;
```

```

b=20;
printf("Before swap a %d b %d\n",a,b);
swap();
printf("After swap a %d b %d\n",a,b);
return 0;
}

```

Output:



```

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Before swap a 10 b 20
After swap a 20 b 10

```

Storage Classes:

A storage class specifies the properties of a variable like default value, memory location, scope and life time.

The following are the storage classes exist in C language

1. Automatic (auto)
2. Register (register/reg)
3. Static (static)
4. External (extern)

We can declare the variables using storage class as follows

syntax:

```
storageclass datatype variablename;
```

1. Automatic storage class:

we use the keyword auto for automatic storage class variables.

The variables declared in the program by default known as automatic storage class variables.

To declare automatic storage class variables its not compulsory to use auto keyword.

ex:

```
auto datatype variable;
```

```
auto int x; (or) int
```

Default value : Garbage value/unknown value

Memory location : Main Memory/RAM/Temporary Memory/Volatile Memory

Scope: The scope of automatic storage class variable is accessible within the block or function in which the variable is declared.

Life Time: The lifetime of Automatic storage class variable is until the execution control remains inside the function or block in which the variable is declared.

2. Register (register/reg):

We use the keyword register to declare the register storage class variables.

The memory is allocated in CPU registers for this storage class variables to access them faster.

ex:

```
register datatype variable;
```

```
register int x;
```

Default value : Garbage value/unknown value

Memory location : CPU Registers

Scope: The scope of register storage class variable is accessible within the block or function in which the variable is declared.

Life Time: The lifetime of register storage class variable is until the execution control remains inside the function or block in which the variable is declared.

3. Static Storage class:

we use the keyword static to declare the static storage class variables.

We declare this variable to retain the value of a variable between the various function calls.

ex:

```
static datatype variable;
```

```
static int x;
```

Default value : 0(zero)

Memory location : Main Memory/RAM/Temporary Memory/Volatile Memory

Scope: The scope of static storage class variable is accessible within the block or functions in which they are used.

Life Time: The lifetime of static storage class variable is until the execution control remains inside the function or block in which they are used.

4. External storage class:

we use the keyword extern to declare the external storage class variables. We use this variables to declare in one program and access in other program.

ex:

```
extern datatype variable;
```

```
extern int x;
```

Default value : Garbage value/unknown value

Memory location : Main Memory/RAM/Temporary Memory/Volatile Memory

Scope: The scope of external storage class variable is accessible within the block or functions in which they are used.

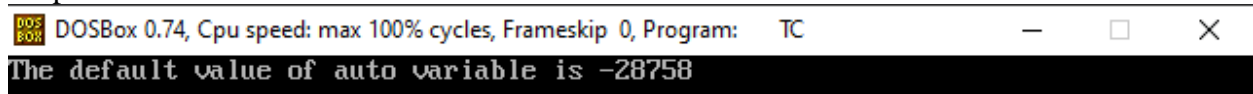
Life Time: The lifetime of static storage class variable is until the execution control remains inside the function or block in which they are used.

Write a program to demonstrate to declare an automatic storage class variable and display its default value.

auto1.c

```
#include<stdio.h>
main()
{
    auto int x;
    clrscr();
    printf("The default value of auto variable is %d\n",x);
    return 0;
}
```

Output:

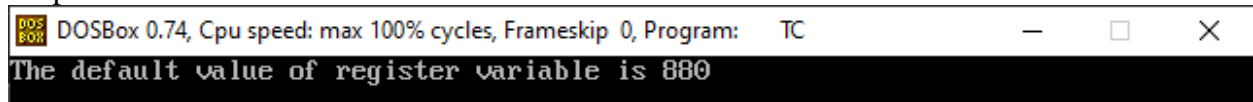


Write a program to demonstrate to declare a register storage class variable and display its default value.

reg1.c

```
#include<stdio.h>
main()
{
    register int x;
    clrscr();
    printf("The default value of register variable is %d\n",x);
    return 0;
}
```

Output:

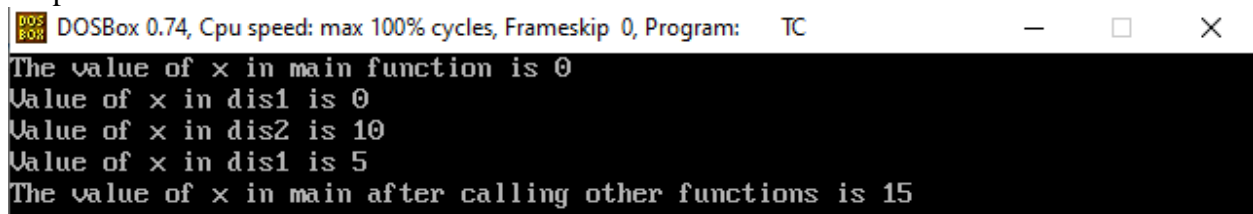
A screenshot of a DOSBox window. The title bar reads "DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC". The window content shows the text "The default value of register variable is 880" on a black background with white text.

Write a program to demonstrate to declare and access the static variable

static1.c

```
#include<stdio.h>
void dis1();
void dis2();
void dis1()
{
    printf("Value of x in dis1 is %d\n",x);
    x=x+10;
}
void dis2()
{
    printf("Value of x in dis2 is %d\n",x);
    x=x-5;
}
main()
{
    static int x;
    printf("The value of x in main function is %d\n",x);
    dis1();
    dis2();
    dis1();
    printf("The value of x in main after calling other functions is %d\n",x);
    return 0;
}
```

Output:

A screenshot of a DOSBox window. The title bar reads "DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC". The window content shows the following output on a black background with white text: "The value of x in main function is 0", "Value of x in dis1 is 0", "Value of x in dis2 is 10", "Value of x in dis1 is 5", and "The value of x in main after calling other functions is 15".

Write a program to demonstrate the declaration and accessing of extern variable

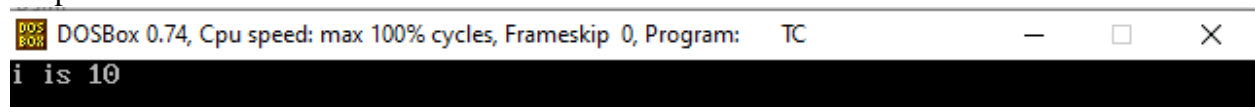
other.c

```
int i;
```

current.c

```
#include<stdio.h>
#include "other.c"
main()
{
    extern int i;
    i=10;
    clrscr();
    printf("i is %d\n",i);
    return 0;
}
```

Output:



Recursive Functions(Recursion):

A function which calls itself is known as a recursive function.
Here, the calling function and called function both are same

Ex:

```
void fun1();
```

```
void fun1()
```

```
{
```

```
.....
```

```
fun1();
```

```
}
```

Write a program to demonstrate the recursive function to display your name for 10 times.

rec1.c

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

```
static int count;
```

```
void display();
```

```
void display()
```

```
{
```

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

```
count++;
```

```
if(count<=10)
```

```
    display();
```

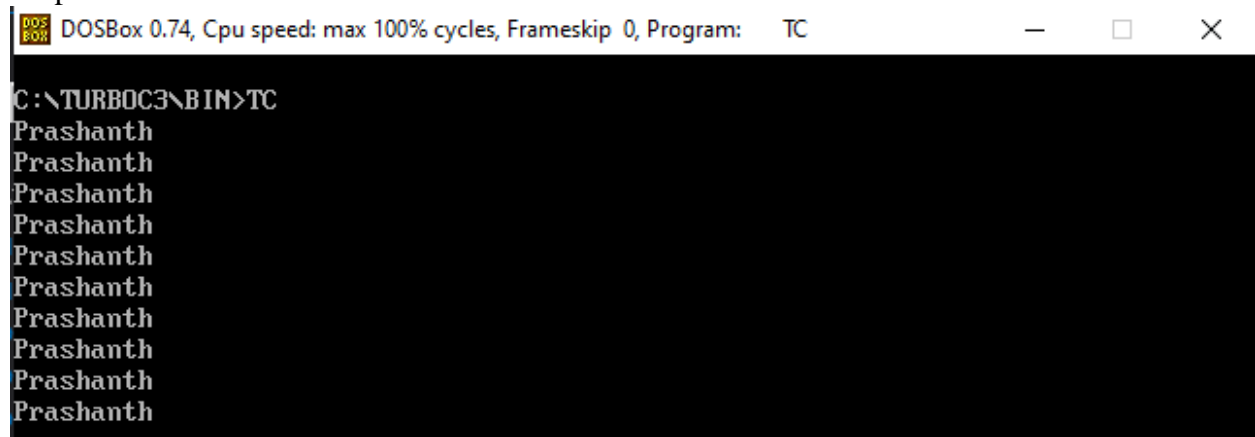
```

}

int main()
{
    display();
    return 0;
}

```

Output:



```

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
C:\TURBOC3\BIN>TC
Prashanth
Prashanth
Prashanth
Prashanth
Prashanth
Prashanth
Prashanth
Prashanth
Prashanth
Prashanth
Prashanth

```

Write a program to calculate the factorial of a given number using recursion

```

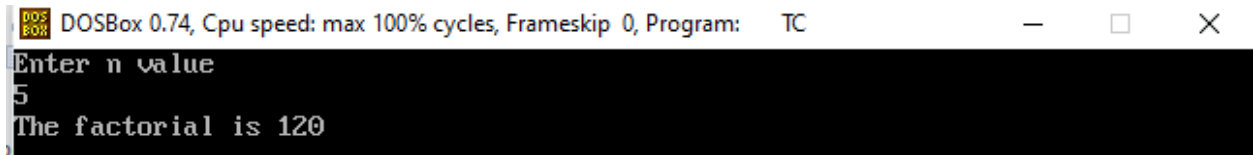
rec2.c
#include<stdio.h>
int fact(int n);

int fact(int n)
{
    if(n==0)
        return 1;
    else
        return n*fact(n-1);
}

int main()
{
    int n,f;
    printf("Enter n value\n");
    scanf("%d",&n);
    f=fact(n);
    printf("The factorial is %d\n",f);
    return 0;
}

```

Output:



```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter n value
5
The factorial is 120
```

Write a program to display the multiplication table of a given number using recursion

rec3.c

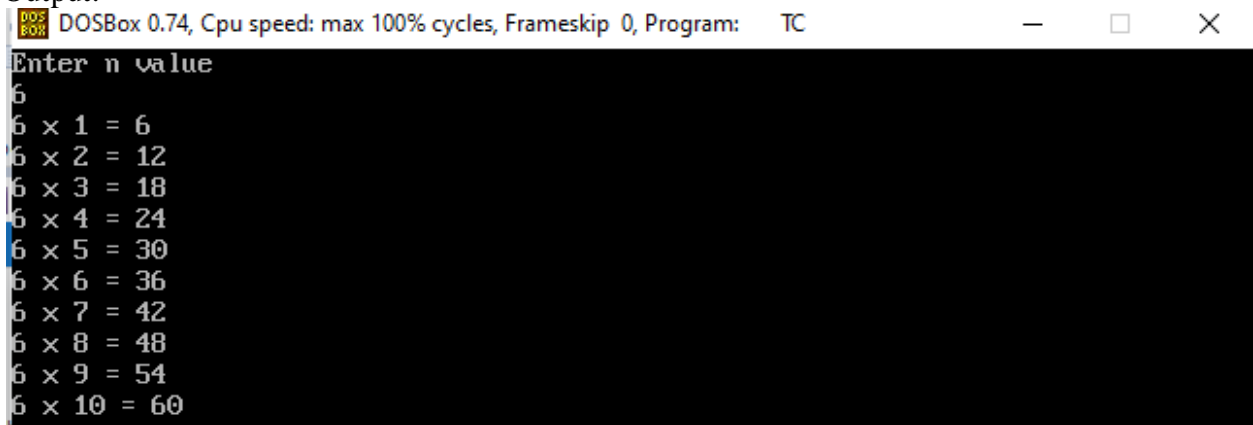
```
#include<stdio.h>

static int count=1;
void display(int n);

void display(int n)
{
printf("%d x %d = %d\n",n,count,count*n);
count++;
if(count<=10)
    display(n);
}

int main()
{
    int n;
    printf("Enter n value\n");
    scanf("%d",&n);
    display(n);
    return 0;
}
```

Output:



```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter n value
6
6 x 1 = 6
6 x 2 = 12
6 x 3 = 18
6 x 4 = 24
6 x 5 = 30
6 x 6 = 36
6 x 7 = 42
6 x 8 = 48
6 x 9 = 54
6 x 10 = 60
```

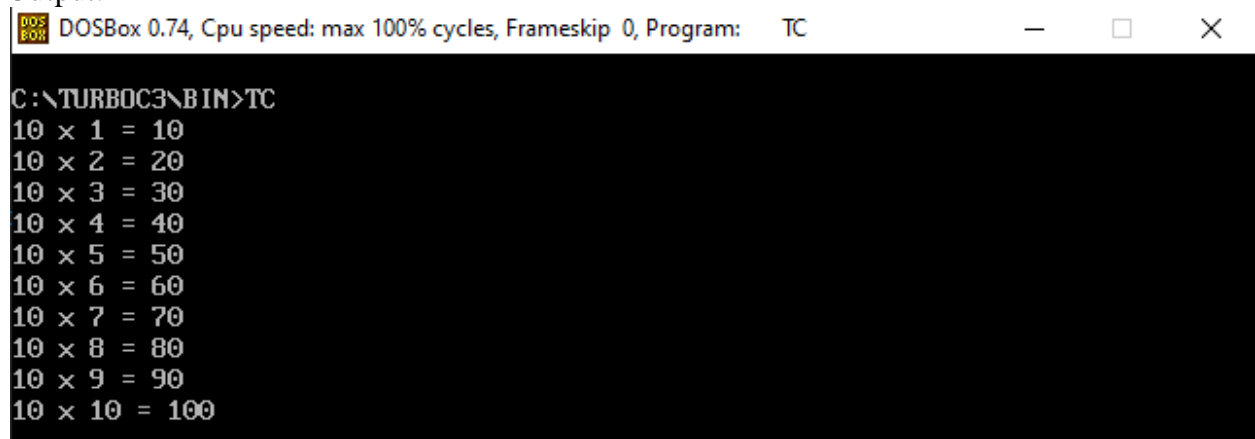
We can also call the main function within main function recursively

Write a program to display the multiplication table of a given number using main function as recursive function

rec4.c

```
#include<stdio.h>
static int count=1;
main()
{
static int n=5;
printf("%d x %d = %d\n",n,count,n*count);
count++;
if(count<=10)
    main();
return 0;
}
```

Output:

A screenshot of a DOSBox window titled "DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC". The window shows a command prompt with the command "C:\TURBOC3\BIN>TC" and the output of the program, which is a multiplication table for the number 10. The output consists of ten lines, each showing a multiplication of 10 by a number from 1 to 10, with the result on the right. The text is white on a black background.

```
C:\TURBOC3\BIN>TC
10 x 1 = 10
10 x 2 = 20
10 x 3 = 30
10 x 4 = 40
10 x 5 = 50
10 x 6 = 60
10 x 7 = 70
10 x 8 = 80
10 x 9 = 90
10 x 10 = 100
```

Dynamic Memory Allocation Functions:

We can create the memory dynamically during run time using dynamic memory allocation functions.

The following are the dynamic memory allocation functions

1. malloc()
2. calloc()
3. realloc()
4. free()

1. malloc():

it allocates the memory to a variable dynamically and returns the address.

syntax:

```
void* malloc(no.of bytes);
```

2. calloc():

it creates the memory for the specified number of elements and returns the base address to the pointer.

syntax:

```
void* calloc(sizeofelement,numberofelements);
```

3. realloc():

it reallocates the memory to an existing array either by increasing the size or by decreasing the size.

```
void* realloc(ptr,sizeofelement,numberofelements);
```

4. free():

It deallocates(deletes) the allocated memory.

(or)

It free the allocated memory

syntax:

```
void free(pointer);
```

Write a program to demonstrate the malloc function to create a memory for a variable.

dm1.c

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

```
main()
```

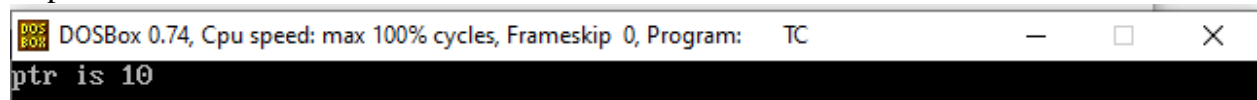
```
{
```

```

int *ptr;
clrscr();
ptr=(int *)malloc(sizeof(int));
*ptr=10;
printf("ptr is %d\n",*ptr);
return 0;
}

```

output:



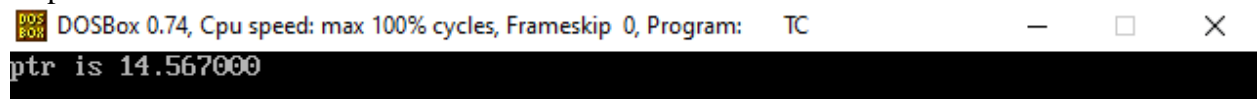
dm2.c

```

#include<stdio.h>
main()
{
float *ptr;
clrscr();
ptr=(float *)malloc(sizeof(float));
*ptr=14.567;
printf("ptr is %f\n",*ptr);
return 0;
}

```

output:



Write a program to demonstrate the calloc function to create an array and display its elements.

dm3.c

```

#include<stdio.h>
main()
{
int *ptr;
int n,i;
clrscr();
printf("Enter how many elements\n");
scanf("%d",&n);
ptr=(int *)calloc(sizeof(int),n);
printf("Enter %d elements\n",n);
for(i=0;i<n;i++)
{
scanf("%d",ptr+i);
}
}

```

```

printf("The addresses and elements are\n");
for(i=0;i<n;i++)
{
printf("%u\t%d\n",ptr+i,*(ptr+i));
}
return 0;
}
output:

```

```

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter how many elements
5
Enter 5 elements
10
20
30
40
50
The addresses and elements are
1910    10
1912    20
1914    30
1916    40
1918    50

```

Write a program to create an array using the malloc function, read the elements and display addresses,elements

dm4.c

```

#include<stdio.h>
main()
{
int *ptr;
int n,i;
clrscr();
printf("Enter how many elements\n");
scanf("%d",&n);
ptr=(int *)malloc(sizeof(int)*n);
printf("Enter %d elements\n",n);
for(i=0;i<n;i++)
{
scanf("%d",ptr+i);
}
printf("The addresses and elements are\n");
for(i=0;i<n;i++)
{
printf("%u\t%d\n",ptr+i,*(ptr+i));
}
}

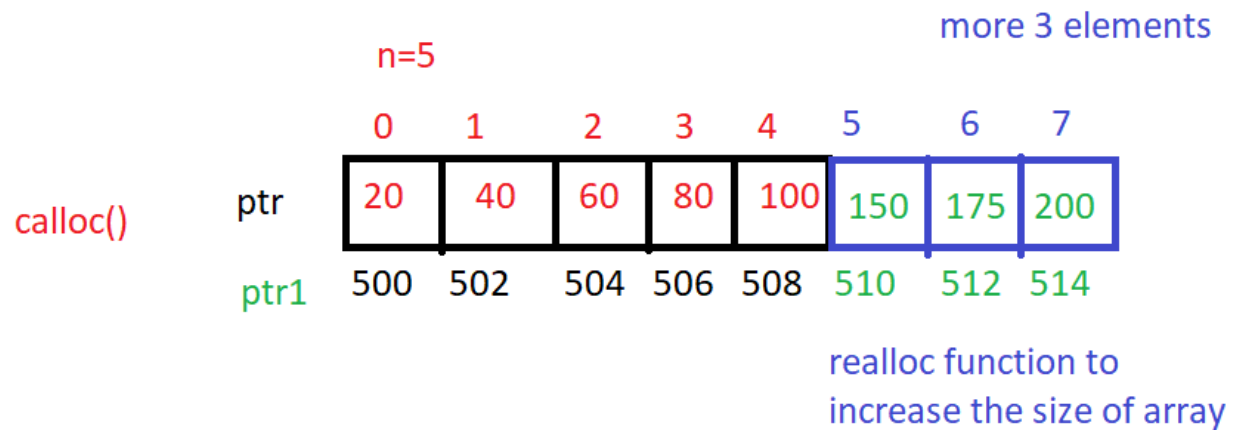
```

```
return 0;
}
```

output:

```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter how many elements
5
Enter 5 elements
10
20
30
40
50
The addresses and elements are
1910 10
1912 20
1914 30
1916 40
1918 50
```

Write a program to create an array to store 5 elements using calloc function, increase the size of array to store some more elements into the same array using realloc function. display addresses and elements.



dm5.c

```
#include<stdio.h>
main()
{
int *ptr,*ptr1;
int n,i,ns;
clrscr();
printf("Enter how many elements\n");
scanf("%d",&n);
ptr=(int *)calloc(sizeof(int),n);
printf("Enter %d elements\n",n);
```

```

for(i=0;i<n;i++)
{
scanf("%d",ptr+i);
}
printf("The addresses and elements are\n");
for(i=0;i<n;i++)
{
printf("%u\t%d\n",ptr+i,*(ptr+i));
}
printf("Enter the new size\n");
scanf("%d",&ns);
ptr1=(int *)realloc(ptr,sizeof(int),ns);
printf("The addresses and elements of array after increase\n");
for(i=0;i<ns;i++)
{
printf("%u \t %d\n",ptr1+i,*(ptr1+i));
}
printf("Enter the new %d elements\n",ns-n);
for(i=n;i<ns;i++)
{
scanf("%d",ptr1+i);
}

printf("The addresses, elements of array after new elements inserted\n");
for(i=0;i<ns;i++)
{
printf("%u\t%d\n",ptr1+i,*(ptr1+i));
}
return 0;
}

```

output:

```

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter how many elements
5
Enter 5 elements
20
40
60
80
100
The addresses and elements are
2092    20
2094    40
2096    60
2098    80
2100   100
Enter the new size

```

```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter the new size
8
The addresses and elements of array after increase
2092    20
2094    40
2096    60
2098    80
2100   100
2102   150
2104   175
2106   200
Enter the new 3 elements
150
175
200
The addresses, elements of array after new elements inserted
2092    20
2094    40
2096    60
2098    80
2100   100
2102   150
2104   175
2106   200
```