**Input & Output Statements in C**

The process of giving something to the computer is known as input. Input is mostly given through keyboard. The process of getting something from the computer is known as output. Output is mostly displayed on monitors.

**Types of I/O Statements**

1. Formatted I/O
   - `scanf()`
   - `printf()`

2. Unformatted I/O
   - `gets()`, `puts()`
   - `getch()`, `getche()`, `putch()`
   - `getchar()`, `putchar()`

The functions used for input and output are stored in the header file `stdio.h`. If programmer use any of the above function it is necessary to include header file.

1. Formatted Input & Output
   (i) `scanf()` function

   The function used to get input from the user during execution of program and stored in a variable of specified form is called `scanf()` function.

   **Syntax:**
   ```c
   scanf("format string", & variable name);
   ```

   **Types:**
   - Single input e.g `scanf("%d", &a);`
   - Multiple input `scanf("%d %c %d", &a, &op, &b);`

   **Example**
   ```c
   #include<stdio.h>
   #include<conio.h>
   void main ()
   {
       int a;
       clrscr();
       printf("Enter integer value: ");
       scanf("%d", &a);
       printf("The value you enter = %d", a);
       getch();
   }
   ```

   **Output**
   Enter integer value: 5
   The value you enter = 5

   (ii) `printf()` Function

   This function is used to display text, constant or value of variable on screen in specified format.

   **Syntax:**
   ```c
   printf("format string", argument list);
   ```
Types:

- `printf("hello world");` // Printf() with no argument list
- `printf("the value of integer=%d",a);` // Printf() with one argument
- `printf("Sum of %d+%d=%d",a,b,a+b);` // Printf() with more than one argument

Example:
```c
#include<stdio.h>
#include<conio.h>
void main ()
{
    printf("HELLO WORLD!");
    getch();
}
```

Output
HELLO WORLD!

2. Unformatted Input & Output

(i) `gets()`
This special function is used to input string values from the user during execution of the program. As the user presses enter key when typing after the string. A null character (\0) is automatically entered at the end of the string.

Syntax:
`gets(variable name);`

(ii) `puts function`
The function used to display the string value on the screen.

Syntax:
`puts(parameter/value/variable);`

Example:
```c
#include<stdio.h>
#include<conio.h>
void main ()
{
    char name[15];
    clrscr();
    printf("enter your name:");
    gets(name);
    printf("your name is:");
    puts(name);
    getch();
}
```

Output:
enter your name: somebody
your name is: somebody

(iii) `getch()`
`getch()` is used to get a character from console but does not echo to the screen. It reads a single character directly from the keyboard, without echoing to the screen.
Syntax
ch = getch();

Program
#include <stdio.h>
void main()
{
    char ch;
    ch = getch();
    printf("Input Char Is :\%c",ch);
}

(iv) getche()
getche() is used to get a character from console, and echoes to the screen. getche reads a single character from the keyboard and echoes it to the current text window, using direct video or BIOS.

Syntax
ch = getche();

Program
#include <stdio.h>
void main()
{
    char ch;
    ch = getche();
    printf("Input Char Is :\%c",ch);
}

(v) putchar()
putchar displays any alphanumeric characters to the standard output device. It displays only one character at a time.

Syntax
putchar(variable_name);

Example
char ch = 'a';
putchar(ch)

Program
#include <stdio.h>
void main()
{
    char ch = 'a';
    putchar(ch);
}

(vi) getchar()
The getchar function is a part of the standard C input/output library. It returns a single character from a standard input device (typically a keyboard). The function does not require any arguments, though a pair of empty parentheses must follow the word getchar.
Syntax
character variable = getchar();
where character variable refers to some previously declared character variable.

(vii) putchar()
The putchar function like getchar is a part of the standard C input/output library. It transmits a single character to the standard output device (the computer screen).

Syntax
putchar(character variable)
where character variable refers to some previously declared character variable.

Program
#include <stdio.h>
int main()
{
    int c;
    printf("Enter a value : ");
    c = getchar();
    printf("\nYou entered: ");
    putchar(c);
    return 0;
}
Enter a value: this is test
You entered: t

Control Structure
A statement that is used to control the flow of execution in a program is called control structure. It combines instruction into logical unit. Logical unit has one entry point and one exit point.

Types of control structures
1. Sequence
2. Selection (branching)
3. Repetition (Iteration)

1. Sequence Statements
Statements are executed in a specified order. No statement is skipped and no statement is executed more than once.

Example
#include<stdio.h>
void main()
{
    int a=5;
    printf("Square of a = %d",a);
}

Output
Square of a = 25
2. Selection Statements
Selection structures allow the computer to make decisions in your programs. It selects a statement or set of statements to execute on the basis of a condition.

Types:
- if-else statement
- switch statement

(i) If Statement
The simplest if structure involves a single executable statement. Execution of the statement occurs only if the condition is true.

Syntax:
if (condition)
    statement;

Flow chart:

```
if (conditional expression)
    statement block executed if condition is true(satisfied).
else
    false branch
```

Example:
```c
#include<stdio.h>
#include<conio.h>
void main ()
{
    int marks;
    clrscr();
    printf("Enter your marks:");
    scanf("%d", &marks);
    if(marks >=50)
        printf("CONGRATULATIONS...!!! you have passed.");
    getch();
}
```

Output
Enter your marks: 67
CONGRATULATIONS...!!! you have passed.

(ii) If-else statement
In if-else statement if the condition is true, then the true statement(s), immediately following the if-statement are executed otherwise the false statement(s) are executed. The use of else basically allows an alternative set of statements to be executed if the condition is false.
Syntax:
if (condition)
{
    Statement(s);
}
else
{
    Statement(s);
}

Flow chart:

Example:
#include<stdio.h>
#include<conio.h>
void main ()
{
    int y;
    clrscr();
    printf("Enter a year:'");
    scanf("%d",&y);
    if (y % 4==0)
        printf("%d is a leap year.'",y);
    else
        printf("%d is not a leap year.'",y);
    getch();
}

Output
Enter a year: 2004
2004 is a leap year.

(iii) if -else if statement (else if ladder)
It can be used to choose one block of statements from many blocks of statements. The condition which is true only its block of statements is executed and remaining are skipped.
**Syntax:**

```c
if (condition)
{
    statement(s);
}
else if (condition)
{
    statement(s);
}
else
{
    statement;
}
```

**Example:**

```c
#include <stdio.h>
#include <conio.h>
void main()
{
    int num;
    clrscr();
    printf("Enter your mark: ");
    scanf("%d",&num);
    printf(" You entered %d", num);
    if(num >= 80)
    {
        printf(" You got A grade");
    }
    else if ( num >=60 && num<80)
    {
        printf(" You got B grade");
    }
    else if ( num >=50&& num<60)
    {
        printf(" You got C grade");
    }
    else if ( num < 50)
    {
        printf(" You Failed in this exam");
    }
    getch();
}
```

**Output**

Enter your mark: 67
You got B grade
(iv) Nested If
In nested if statement if the first if condition is true the control will enter inner if. If this is true the statement will execute otherwise control will come out of the inner if and the else statement will be executed.

Syntax:
if ( condition 1)
{
    if ( condition 2)
        statement 1 ;
    else
        statement 2 ;
}
else
{
    if (condition 3)
        statement 3;
    else
        statement 4;
}

Example:
#include<stdio.h>
#include<conio.h>
void main ()
{
    int a,b,c;
    clrscr();
    printf("Enter three number:");
    scanf("%d %d %d,&a,&b,&c);
    if (a>b)
    {
        if (a>c)
            printf("a is big");
        else
            printf("c is big");
    } else
    {
        if(b>c)
            printf("b is big");
        else
            printf("c is big");
    }
    getch();
}

Output
Enter three number:
4
Switch Statement

- The control statements which allow us to make a decision from the number of choices is called switch (or) Switch-case statement.
- It is a multi way decision statement, it test the given variable (or) expression against a list of case value.

Rules for Switch

- The expression in the switch statement must be an integer or character constant.
- No real numbers are used in an expression.
- The default is optional and can be placed anywhere, but usually placed at end.
- The case keyword must be terminated with colon (:) 
- No two case constant are identical.
- The values of switch expression is compared with case constant in the order specified i.e from top to bottom.
- A switch may occur within another switch, but it is rarely done. Such statements are called as nested switch statements.
- The switch statement is very useful while writing menu driven programs.

Syntax:

```c
switch(expression)
{
    case 1:
        statement;
        break;
    case 2:
        statement;
        break;
    .
    .
    .
    case N:
        statement;
        break;
    default:
        statement;
}
```

Program:

```c
#include<stdio.h>
#include<conio.h>
void main()
{
    int num1,num2,ans1,choice;
    float ans2;
```
clrscr();
printf("\n enter two numbers");
scanf("%d%d",&num1,&num2);
printf("\n1.Addition \n 2.Subtraction \n 3.Multiplication \n 4.Division \n");
printf("\n Enter the choice:");
scanf("%d",&choice);
switch(choice)
{
    case 1:
        ans1=num1+num2;
        printf("Addition =%d",ans1);
        break;
    case 2:
        ans1=num1-num2;
        printf("Subtraction =%d",ans1);
        break;
    case 3:
        ans1=num1*num2;
        printf("Multiplication =%d",ans1);
        break;
    case 4:
        ans2=(float)num1/num2;
        printf("Division =%f",ans2);
        break;
    default:
        printf("wrong choice");
        break;
}
}

getch();

Output
enter two numbers
23
41
1.Addition
2.Subtraction
3.Multiplication
4.Division
Enter the choice 1
64

3. Iteration (or) Looping Statements
Loop is used to repeatedly execute set of statement. Structure that repeats a statement is known as iterative, repetitive or looping construct.
Purpose:
- Execute statements for a specified number of times.
- To use a sequence of values.
Types
1. While loop
2. Do while loop
3. For loop

(i) while loop
It executes one or more statements while the given condition remains true. It is useful when number of iterations is unknown.

Syntax
initialization;
while (condition)
{
    statement;
    increment/decrement;
}

Flow chart

Example
#include<stdio.h>
#include<conio.h>
void main (void)
{
    int n=1;
    clrscr();
    while (n<=5)
    {
        printf("\n%d",n);
        n++;
    }
    getch();
}

Output
1
2
3
4
5
(ii) do while loop
Do while loops are useful where loop is to be executed at least once. In do while loop condition comes after the body of loop. This loop executes one or more statements while the given condition is true.

Syntax
initialization;
do
{
    statement(s);
    increment/decrement;
}while (condition);

Flow chart

Example
#include<stdio.h>
#include<conio.h>
void main (void)
{
    int a=1;
    clrscr();
do
    {
        printf("n %d",a);
        a++;
    }while (a<=5);
    getch();
}

Output
1
2
3
4
5

Difference Between While Loop and Do – While Loop

<table>
<thead>
<tr>
<th>Sl.No.</th>
<th>while loop</th>
<th>do-while loop</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>The while loop tests the condition before each iteration.</td>
<td>The do – while loop tests the condition after the first iteration.</td>
</tr>
</tbody>
</table>
2. If the condition fails initially the loop is Skipped entirely even in the first iteration.  Even if the condition fails initially the loop is executed once.

(iii) **For loop**
For loops are used when the number of iterations is known before entering the loop. It is also known as counter-controlled loop.

**Syntax**
for (initialization; condition; increment/decrement)
{
    statement();
}

**Flow chart**

![Flow chart](image)

**Example**
#include<stdio.h>
#include<conio.h>
void main()
{
    int n;
    for(n=1;n<=5;n++)
    {
        printf("\n %d",n);
    }
    getch();
}

**Output**
1
2
3
4
5
Nested loop
A loop within a loop is called nested loop. In this the outer loop is used for counting rows and the internal loop is used for counting columns. Any loop can be used as inner loop of another loop.

Syntax
for (initialization; condition; increment/decrement)
{
  for(initialization; condition, increment/decrement)
  {
    statements(s);
  }
}

Example
#include<stdio.h>
#include<conio.h>
void main ()
{
  int i,j;
  clrscr();
  for(i=1;i<=3;i++)
  {
    for(j=1;j<=2;j++)
    {
      printf("\n%d\t%d",i,j);
    }
  }
  getch();
}

Output
1  1
1  2
2  1
2  2
3  1
3  2

Continue Statement
• Continue statement transfer the control to the start of block. It is used in the body of loop. The continue statement is used to transfer the control to the beginning of the loop, there by terminating the current iteration of the loop and starting again from the next iteration of the same loop.
• The continue statement can be used within a while or a do – while or a for loop.
• The general form or the syntax of the continue statement is continue;
• The continue statement does not have any expressions or arguments.
• Unlike break, the loop does not terminate when a continue statement is encountered, but it terminates the current iteration of the loop by skipping the remaining part of the loop and resumes the control tot the start of the loop for the next iteration.
Program
#include <stdio.h>
#include<conio.h>
void main()
{
    int i;
    clrscr();
    i = 1;
    for(i=0;i<5;i++)
    {
        if(i==3)
            continue;
        printf("%d",i);
    }
    getch();
}

Output
0
1
2
4

break Statement
1. It is a tool to take out the control. It transfers control to end of block. It is used in body of loop and switch statements. A break statement is used to terminate of to exit a for, switch, while or do – while statements and the execution continues following the break statement.
2. The general form of the break statement is
   break;
3. The break statement does not have any embedded expression or arguments.
4. The break statement is usually used at the end of each case and before the start of the next case statement.
5. The break statement causes the control to transfer out of the entire switch statement.

Program
#include <stdio.h>
#include<conio.h>
void main()
{
    int i;
    clrscr();
    i = 1;
    for(i=0;i<5;i++)
    {
        if(i==3)
            break;
        printf("%d",i);
    }
}
Output
0
1
2

<table>
<thead>
<tr>
<th>Sl.No.</th>
<th><strong>break</strong></th>
<th><strong>continue</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Used to terminate the loops or to exit loop from a switch.</td>
<td>Used to transfer the control to the start of loop.</td>
</tr>
<tr>
<td>2.</td>
<td>The break statement when executed causes immediate termination of loop containing it.</td>
<td>Continue statement when executed causes Immediate termination of the current iteration of the loop.</td>
</tr>
</tbody>
</table>

**goto statement**
It is an unconditional transfer of control. It transfers the control to the specific point. The goto statement is marked by label statement. Label statement can be used anywhere in the function above or below the goto statement. It is written as goto label;

**Syntax**
```
goto label;  
-----------
-----------
label:     goto label;  
-----------
-----------
statement; 
-----------
-----------
```

**Example:**
```
#include <stdio.h>
#include <conio.h>
void main()
{
    int a=3;
    clrscr();
    Start: printf("Welcome\n");
         if(a<5)
         {
             goto Start;
             a++;
         }
         getch();
}
```

**Output**
Welcome
Welcome
Welcome

**Arrays**
- C programming language provides a data structure called **the array**, which can store a fixed-size sequential collection of elements of the same type.
An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables.

A specific element in an array is accessed by an index.

All arrays consist of contiguous memory locations.

The lowest address corresponds to the first element and the highest address to the last element.

Arrays are of two types:
1. One-dimensional arrays
2. Multidimensional arrays

1. One-dimensional arrays

Declaration
data_type array_name[array_size];

For example:
int age[5];
Here, the name of array is age. The size of array is 5, i.e., there are 5 items (elements) of array age. All element in an array are of the same type (int, in this case).

Array elements
Size of array defines the number of elements in an array. Each element of array can be accessed and used by user according to the need of program. For example:
int age[5];

Note that, the first element is numbered 0 and so on.
Here, the size of array age is 5 times the size of int because there are 5 elements.

Initialization of one-dimensional array:
Arrays can be initialized at declaration time in this source code as:
int age[5]={2,4,34,3,4};
It is not necessary to define the size of arrays during initialization.
int age[]={2,4,34,3,4};
In this case, the compiler determines the size of array by calculating the number of elements of an array.

![Initialization of one-dimensional array]

**Accessing array elements**
In C programming, arrays can be accessed and treated like variables in C. For example:

```c
scanf("%d", &age[2]);
/* statement to insert value in the third element of array age[]. */
scanf("%d", &age[i]);
/* Statement to insert value in (i+1)th element of array age[]. */
/* Because, the first element of array is age[0], second is age[1], ith is age[i-1] and (i+1)th is age[i]. */
printf("%d", age[0]);
/* statement to print first element of an array. */
printf("%d", age[i]);
/* statement to print (i+1)th element of an array. */
```

**Program**
/* C program to find the sum marks of n students using arrays */
#include <stdio.h>
void main()
{
    int marks[10], i, n, sum = 0;
    printf("Enter number of students: ");
    scanf("%d", &n);
    for (i = 0; i < n; ++i)
    {
        printf("Enter marks of student%d: ", i + 1);
        scanf("%d", &marks[i]);
        sum += marks[i];
    }
    printf("Sum = %d", sum);
}

**Output**
Enter number of students: 3
Enter marks of student1: 12
Enter marks of student2: 31
Enter marks of student3: 2
sum = 45

**2. Multidimensional Arrays**
C programming language allows programmer to create arrays of arrays known as multidimensional arrays. For example:
float a[2][6];
Here, a is an array of two dimension, which is an example of multidimensional array.
For better understanding of multidimensional arrays, array elements of above example can be thoughted of as below:

<table>
<thead>
<tr>
<th>col 1</th>
<th>col 2</th>
<th>col 3</th>
<th>col 4</th>
<th>col 5</th>
<th>col 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>row 1</td>
<td>a[0][0]</td>
<td>a[0][1]</td>
<td>a[0][2]</td>
<td>a[0][3]</td>
<td>a[0][4]</td>
</tr>
<tr>
<td>row 2</td>
<td>a[1][0]</td>
<td>a[1][1]</td>
<td>a[1][2]</td>
<td>a[1][3]</td>
<td>a[1][4]</td>
</tr>
</tbody>
</table>

**Initialization of Multidimensional Arrays**
In C, multidimensional arrays can be initialized in different number of ways.

```c
int c[2][3]={{1,3,0}, {-1,5,9}};
```

OR

```c
int c[][3]={{1,3,0}, {-1,5,9}};
```

OR

```c
int c[2][3]={1,3,0, -1,5,9};
```

**Initialization Of three-dimensional Array**

double cprogram[3][2][4]=
    {
        {{-0.1, 0.22, 0.3, 4.3}, {2.3, 4.7, -0.9, 2}},
        {{0.9, 3.6, 4.5, 4}, {1.2, 2.4, 0.22, -1}},
        {{8.2, 3.12, 34.2, 0.1}, {2.1, 3.2, 4.3, -2.0}}
    };

Suppose there is a multidimensional array arr[i][j][k][m]. Then this array can hold i*j*k*m numbers of data. Similarly, the array of any dimension can be initialized in C programming.

**Program**

```c
#include<stdio.h>

void main()
{
    int a[5][5],b[5][5],c[5][5];
    int row1,row2,col1,col2,i,j,k;
    printf("Enter the row value of first matrix\n");
    scanf("%d",&row1);
    printf("Enter the column value of first matrix\n");
    scanf("%d",&col1);
    printf("Enter the row value of second matrix\n");
    scanf("%d",&row2);
    printf("Enter the column value of second matrix\n");
    scanf("%d",&col2);
    if((row1==row2)&&(col1==col2))
    {
        printf("Matrix can be added \n");
        printf("Enter the values of first matrix\n");
        for(i=1;i<=row1;i++)
        {
```
for(j=1;j<=col1;j++)
{
    scanf("%d",&a[i][j]);
}

printf("Enter the values of second matrix\n");
for(i=1;i<=row2;i++)
{
    for(j=1;j<=col2;j++)
    {
        scanf("%d",&b[i][j]);
    }
}
for(i=1;i<=row1;i++)
{
    for(j=1;j<=col1;j++)
    {
        c[i][j]=a[i][j]+b[i][j];
    }
}
printf("Sum of the two matrix is\n");
for(i=1;i<=row1;i++)
{
    for(j=1;j<=col1;j++)
    {
        printf("%d\t",c[i][j]);
    }
    printf("\n");
}
else
    printf("Addition cannot be perform");

OUTPUT
Enter the row value of first matrix 2
Enter the column value of first matrix 2
Enter the row value of second matrix 2
Enter the column value of second matrix 2
Matrix can be added
Enter the values of first matrix
2 4
5 6
Enter the values of first matrix
3 1
2 8
Sum of the two matrix
5 5
7 14