First Year First Semester Course
M.Tech. (CS) [Batch 2021-23]

Lectures #03-#04

Introduction to Programming

C: Data Input/Output and Control Statements

Course Instructor:

Dr. Monidipa Das

DST-INSPIRE Faculty

Machine Intelligence Unit (MIU), Centre for Artificial Intelligence and Machine Learning (CAIML)

Indian Statistical Institute (ISI) Kolkata, India

C: Data Input/Output

Data Input/Output

- Data Input/Output (I/O) from the terminal
  - scanf
  - printf
  - getchar
  - putchar

scanf function

- Reads user input from keyboard
- It requires a control string and a list of variables into which the value received from the keyboard will be stored

```
scanf("control string",arg1,arg2, …, argn);
```

```
int a, b;
float c;
scanf("%d%d%f", &a, &b, &c);
```

Variable list (note the '&' before the variable name)

control string

```
scanf ("%d", &var1);
scanf ("%c", &mychar);
scanf ("%f", &weight);
scanf ("%d%f", &a, &b);
```

scanf function

- **Commonly used conversion characters**

Conversion Character	Data Item meaning
%c	Single charater
%d	Decimal integer
%e	Floating point value
%f	Floating point value
%g	Floating point value
%hd	Short integer
%ld	Long integer
%i	Decimal/hexadecimal/octal integer
%o	Octal integer
%s	String
%u	Unsigned decimal integer
%X	Hexadecimal integer

We can also specify the maximum field-width of a data item, by specifying a number indicating the field width before the conversion character.

Example:
scanf ("%3d %5d", &a, &b);

printf function

- Performs output to the standard output device (typically defined to be the screen)

```
printf("control string",arg1,arg2, …, argn);
```

- It requires a control string in which we can specify:
  - The *text to be printed out*
  - *Format specifications* on how to print the values

```
printf("The number is %d\n", num);
```

  - *Escape sequence characters : \n, \t, and \b*
  - The format specification %d causes the value listed after the format string to be embedded in the output as a decimal number in place of %d
  - Output will appear as: The number is 125
  - The conversion characters are the same as in scanf

- The arguments arg1, arg2, … represent the individual output data items.

Formatted Output

float a=3.0, b=7.0;

```
printf("%f %f %f %f",a,b,a+b,sqrt(a+b));
```

3.000000 7.000000 10.000000, 3.162278

```
printf("%4.2f %5.4f\na+b=%3.3f\tSquare Root=%-6.3f",a,b,a+b,sqrt(a+b));
```

3.00 7.0000

a+b=10.000 ⟷ Square Root=3.162

Horizontal tab

For integer, character and string decimal point will not be there. Rest is the same.

Examples

- Examples:

```
printf ("Average of %d and %d is %f\n", a, b, avg);
printf ("Hello \nGood \nMorning \n");
printf("%3d %3d %5d", a, b, a*b+2);
printf("%7.2f %4.1f", x, y);
```

a= 15
b= 12
x= 24.716355
y= 179236.56

What would be the outputs?

```
Average of 12 and 15 is 13.500000
Hello
Good
Morning
 12  15   182  24.72 179236.6
```

Character I/O


```
char ch1;
scanf("%c",&ch1);    /* Reads a character */
printf("%c",ch1);    /* Prints a character */
ch1=getchar();              /* Reads a character */
putchar(ch1);               /* Prints a character */

char name[20];
scanf("%s",name);    /* Reads a string */
printf("%s",name);   /* Prints a string */
gets(name);          /* Reads a string */
puts(name);          /* Prints a string */
```

C: Control Statements

Control Statements

- Control statements help controlling the flow of execution

Statement takes more than one branches based upon a **condition test** comprising of relational and/or logical (or may be arithmetic) operators.

Some set of **statements execute iteratively** until a **condition test** comprising of relational and/or logical (or may be arithmetic) operators are not being satisfied.

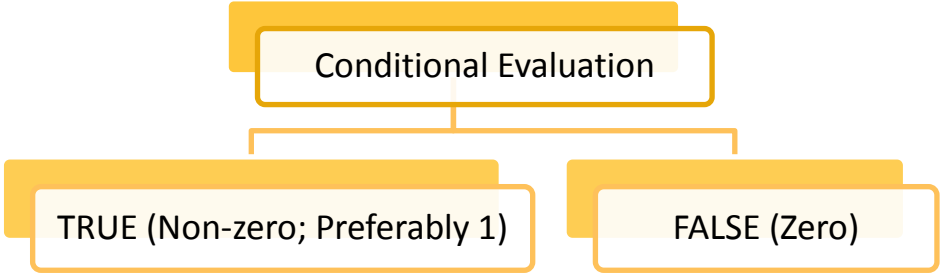
Conditions

- Using relational operators.
  - Four relation operators: <, <=, >, >=
  - Two equality operators: ==, !=

- Using logical operators / connectives.
  - Two logical connectives: &&, ||
  - Unary negation operator: !

Condition Tests (Examples)


```
if(count <= 100) /* Relational */
if((math+phys+chem)/3 >= 60) /* Arithmetic, Relational */
if((sex=='M') && (age>=21)) /* Relational, Logical */
if((marks>=80) && (marks<90) ) /* Relational, Logical */
if((balance>5000) || (no_of_trans>25)) /* Relational, Logical */
if(!(grade=='A')) /* Relational, Logical */
```



Branching in C Programming

- Supports decision-making capability
- Can be achieved by using following statements
  - ✓ **if** statement
  - ✓ **switch** statement
  - ✓ Conditional operator statement
  - ✓ **goto** statement

Simple **if** Statement


```
if (test expression)
{
        Statement-block;
}
Statement-x;
```

```
.....
if(marks<90 && marks>=80)
{
        printf("\nGRADE is A \n");
}
printf(“Condition tested above...\n");
```

if…else and Nesting of **if…else**


```
if (test expression)
{
        True-block statements;
}
else
{
        False-block statements;
}
statement-x;
```

```
if(marks>=40){
        printf("\nStatus: PASS!\n");
}
else{
        printf("\nStatus: FAIL!\n");
}
```

```
if (test condition-1)
{
        if(test condition-2)
        {
                Statement-1;
        }
        else
        {
                Statement-2;
        }
}
else
{
        Statement-3;
}
Statement-x;
```

Nesting of **if…else** Statements


```
#include<stdio.h>
int main(){
        float a, b, c;
        printf("Enter three numbers:\n");
        scanf("%f %f %f",&a, &b, &c);
        printf("\nThe largest number is: ");
        if(a>c){
                if(a>b)
                        printf("%f \n",a);
                else
                        printf("%f \n",b);}
        else{
                if(c>b)
                        printf("%f \n",c);
                else
                        printf("%f \n",b);}

        return 0;}
```

Code for finding the largest among three numbers

else if Ladder


```
if (condition-1)
    Statement-1;
else if (condition-2)
    Statement-2;
else if (condition-3)
    Statement-3;
else if (condition-n)
    Statement-n;
else
    Default-statement;
Statement-x;
```

Branching: **switch** Statement

- ***Multi-way*** decision statement
- **switch** statement tests the value of a given variable (or expression) against a list of **case** values and when a match is found, a block of statements associated with that case is executed

```
switch (expression)
{
        case value-1:
                statement block-1
                break;

        case value-2:
                statement block-2
                break;
        ...
        ...
        default:
                default statement block
                break;

}
statement-x;
```

switch Statement [contd.]


```
switch (getchar())
{
        case 'r':
        case 'R': printf ("RED \n");
                        break;
        case 'g':
        case 'G': printf ("GREEN \n");
                        break;
        case 'b':
        case 'B': printf ("BLUE \n");
                        break;
        default: printf ("Invalid Color \n");
}
```

Ternary conditional operator (?:)

- Takes three arguments (condition, value if true, value if false).
- Returns the evaluated value accordingly.

conditional expression ? expr1: expr2;

- Example:

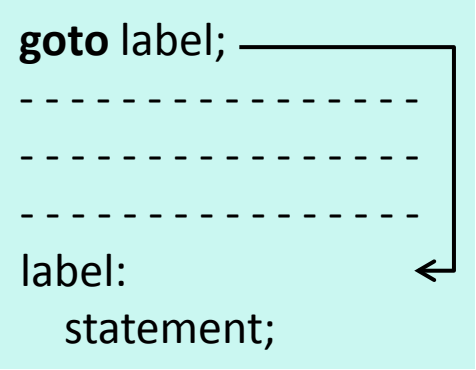
```
bonus = (basicPay<18000) ? basicPay*0.30 : basicPay*0.05;
```

Returns a value

age >= 60 ? printf(“Senior Citizen\n”) : printf(“General Quota\n”);

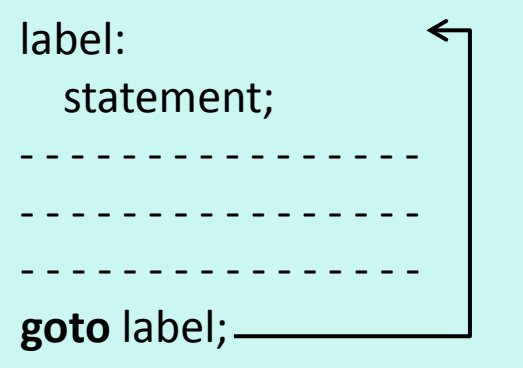
Branching: **goto** Statement

- ***Branch unconditionally*** from one point to another in the program
- **goto** requires a label . The label can be anywhere in the program either before or after `goto label;` statement



Forward jump

Some statements will be skipped



Backward jump

A loop will be formed and some statements will be executed repeatedly

goto Statement [contd.]

- We should try avoiding ***goto*** statement as far as possible
- **goto** example:

```
int main(){
        float x, y;
        read:
                printf("Enter a number:\n");
                scanf("%f",&x);
                if(x<0)
                        goto read;
                y=sqrt(x);
                printf("\nSquare root of %f is: %f\n\n", x, y);
                goto read;
        return 0;
}
```

Control Statements

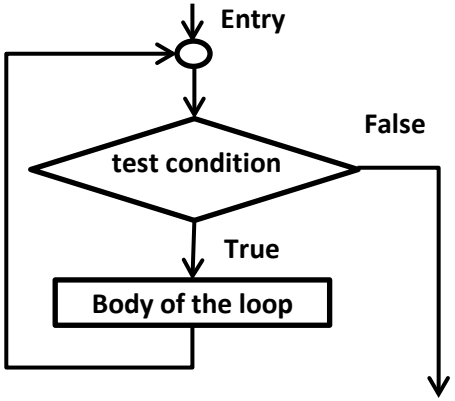
- Control statements help controlling the flow of execution

Statement takes more than one branches based upon a **condition test** comprising of relational and/or logical (or may be arithmetic) operators.

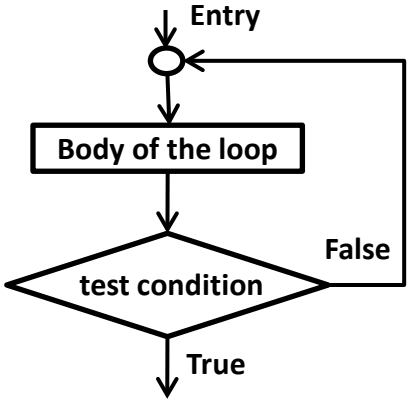
Some set of **statements execute iteratively** until a **condition test** comprising of relational and/or logical (or may be arithmetic) operators are not being satisfied.

Looping

- In looping, a sequence of statements are executed until some conditions for termination of the loop are satisfied.



Entry-controlled loop



Exit-controlled loop

- C supports three constructs for looping
  - The **while** statement
  - The **do while** statement
  - The **for** statement

Looping [contd.]

- Counter-controlled loops
  - We ***know in advance*** exactly how many times the loop will be executed
  - Also called ***definite repetition loops***
  - The control variable is known as ***counter***
  - The number of times we want to execute the loop may be a constant or a variable

- Sentinel-controlled loops
  - A special value, called ***sentinel value*** is used to change the loop control expression from true or false
  - Number of repetitions is ***not known before*** the loop begins executing
  - Also called ***indefinite repetition loops***

Looping [contd.]

- A looping process, in general would include the following **four steps**
  - Setting and initialization of a condition variable
  - Execution of the statements in the loop
  - ***Test for a specified value*** of the condition variable for execution of the loop:
    - Either specified number of iteration or a particular condition satisfaction
  - Incrementing or updating the condition variable

while Statement

- Basic format:

```
while (test condition)
{
        Statement_to_repeat;
}
```

```
int digit = 0;
while (digit <= 9)
        printf ("%d \n", digit++);
```

```
int weight=85;
while ( weight > 80 )
{
        printf(“Go for exercise. \n");
        printf("Enter your weight: ");
        scanf("%d", &weight);
}
```

```
while (test condition)
{
        Statement-1;
        Statement-2;
        ...
        ...
        Statement-n;
}
```

while Statement [contd.]


```
#include<stdio.h>
int main()
{        int count,n;
         float x,y;
         printf("Enter the values of x and n (the value of n should
         be non-negative):\n");
         scanf("%f %d",&x,&n);
         y=1.0;
         count=1;   /* Initialization */
         while(count<=n){   /* Testing */
                  y=y*x;
                  count++; /* Incrementing */
         }
         printf("\nx=%f; n=%d; \nx to the power n=%f\n",x,n,y);
         return 0;
}
```

```
Enter the values of x and n (the value of n should be non-negative):
3.5 3

x=3.500000; n=3;
x to the power n=42.875000
```

Output

A C program for computing $y = x^n$ where n is non-negative integer

while Statement [contd.]

Example of Sentinel-controlled while loop:

```
char ch=‘ ‘;
while (ch !=‘Y’)
      ch= getchar();
```

- **‘Y’: Sentinel value**
- **ch: Sentinel variable**

do while Statement

- In some cases, it must be necessary to execute the body of the loop at least once;
- This can be handled by **do while** statement
- Basic Form:

```
do
{
        Statement_to_repeat;
} while (test condition);
```

```
do
{
        Statement-1;
        Statement-2;
        ...
        ...
        Statement-n;
} while (test condition);
```

do while Statement [contd.]


```c
#include<stdio.h>
/*This code computes the square of a given number.*/
int main()
{
        float num;
        char ch;
        do{
            printf("\n\nEnter a number: ");
            scanf("%f",&num);
            printf("\nSqure of %6.2f is: %6.2f",num,(num*num));
            printf("\n\nDo you want to continue? (Y/N): ");
            fflush(stdin);
            scanf("%c",&ch);
        }while(ch=='Y'||ch=='y');

        printf("\n\nThank you.....Terminating.....\n");
        return 0;
}
```

```
Enter a number: 56.2
Squre of  56.20 is: 3158.44
Do you want to continue? (Y/N): Y

Enter a number: 12
Squre of  12.00 is: 144.00
Do you want to continue? (Y/N): y

Enter a number: 123.59
Squre of 123.59 is: 15274.49
Do you want to continue? (Y/N): N

Thank you.....Terminating.....
```

Output

for Statement

- More concise loop control structure
- Basic Form:

```
for (initial; condition; iteration)
        statement_to_repeat;
```

```
for (initial; condition; iteration)
{
        Statement-1;
        ……
        Statement-n;
}
```

All are expressions:

```
initial→ expr1
condition→ expr2
iteration→ expr3
```

How does it work?

–"expr1" is used to *initialize* some variable (called *index*) that controls the looping action.
–"expr2" represents a *condition* that must be true for the loop to continue.
–"expr3" is used to *change* the value of the *index* initially assigned by "expr1".

for Statement [contd.]


```
#include <stdio.h>
int main()
{        int n, fact, count;
         printf("Enter a number: ");
         scanf("%d",&n);
         fact=1;
         for(count=1;count<=n;count++){
                  fact=fact*count;
         }
         printf("\nFactorial of %d is: %d",n,fact);
         return 0;
}
```

Output:

```
Enter a number: 6

Factorial of 6 is: 720
```

for loop with comma operator


```c
#include <stdio.h>
int main()
{    int n, fact, count;
     printf("Enter a number: ");
     scanf("%d",&n);
     fact=1;
     for(count=1;count<=n;count++){
          fact=fact*count;
     }
     printf("\n%d! = %d",n,fact);
     return 0;
}
```

for loop

```c
#include <stdio.h>
int main()
{
     int n, fact, count;
     printf("Enter a number: ");
     scanf("%d",&n);
     for(fact=1,count=1;count<=n;count++){
          fact=fact*count;
     }
     printf("\n%d! = %d",n,fact);
     return 0;
}
```

for loop with comma operator

The comma operator: We can give several statements separated by commas in place of "expression1", "expression2", and "expression3".

Advanced expression in *for* structure

- **Arithmetic expressions**
  - Initialization, loop-continuation, and increment can contain arithmetic expressions.
  - e.g. Let **x = 4** and **y = 8;** then,

```
for(j = x; j <= 5 * x * y; j += y / x )
```

is equivalent to

```
for ( j = 4; j <= 160; j += 2 )
```

- "Increment" may be negative (decrement)

- If loop continuation condition initially false
  - Body of ***for*** structure not performed
  - Control proceeds with statement after ***for*** structure

Specifying “Infinite Loop”


```
count=1;
while(1) {
        printf("Count=%d\n",count);
        count++;
}
```

```
count=1;
do {
        printf("Count=%d\n",count);
        count++;
} while(1);
```

```
count=1;
for(;;) {
        printf("Count=%d\n",count);
        count++;
}
```

```
for(count=1;;count++) {
        printf("Count=%d\n",count);
}
```

break Statement

- Break out of the loop { }
- can use with
  - **while**
  - **do while**
  - **for**
  - **switch**
- Causes immediate exit from a while, for, do/while or switch structure
- Program execution continues with the first statement after the structure
- Common uses of the break statement
  - ***Escape early from a loop***
  - ***Skip the remainder of a switch structure***

Break from “Infinite Loop”


```
count=1;
while(1) {
        printf("Count=%d\n",count);
        count++;
        if(count>100)
                break;
}
```

```
count=1;
do {
        printf("Count=%d\n",count);
        count++;
        if(count>100)
                break;
} while(1);
```

```
count=1;
for(;;) {
        printf("Count=%d\n",count);
        count++;
        if(count>100)
                break;
}
```

```
for(count=1;;count++) {
        printf("Count=%d\n",count);
        if(count>100)
                break;
}
```

continue Statement

- Skips the remaining statements in the body of a **while**, **for** or **do/while** structure
  - Proceeds with the next iteration of the loop
- **while** and **do/while**
  - Loop-continuation test is evaluated immediately after the continue statement is executed
- **for** structure
  - Increment expression is executed, then the loop-continuation test is evaluated.
  - *expression3* is evaluated, then *expression2* is evaluated.

An Example with **break** and **continue**


```
#include<stdio.h>
// a program to calculate factorial of n
int main()
{       int fact = 1, i = 1, n;
        printf("Enter the value of n: ");
        scanf("%d",&n);
        while (1) {
                fact = fact * i; i++ ;
                if(i<=n)
                    continue; /* Not done yet! Go to next iteration*/
                break;
        }
        printf("\n%d!=%d\n",n,fact);
        return 0;
}
```

Nesting of **for** Loops

- One **for** statement within another **for** statement is allowed in C
- The nesting may continue up to any desired level. However, some standards have certain restriction in this regard
- You should use proper indentation to improve the readability

```
----------
----------
for(i=1;i<10;++i)
{
        ----------
        ----------
        for(j=i;j>=1;j--)
        {
                ----------
                ----------
        }
        ----------
        ----------
}
----------
----------
```

Inner Loop

Outer Loop

More examples of loop [contd.]


```
#include <stdio.h>
//This program checks whether a natural number is prime or not
int main()
{         int n, i=2;
          printf("Enter a natural number: ");
          scanf ("%d", &n);
          while (i < n) {
                    if (n % i == 0) {
                              printf ("%d is NOT PRIME!\n", n);
                              break;
                    }
                    i++;
          }
          if(i>=n)
                    printf ("%d is PRIME!\n", n);
          return 0;
}
```

More examples of loop


```
#include <stdio.h>
/*This program finds the sum of the digits of a number.
We assume the number to be a natural number*/
int main()
{
        int n, sum=0;
        printf("Enter a number: ");
        scanf ("%d", &n);
        while (n != 0) {
                sum = sum + (n % 10);
                n = n / 10;
        }
        printf ("The sum of the digits of the number is %d \n", sum);
        return 0;
}
```

More examples of loop [contd.]


```c
#include <stdio.h>
//This program prints a triangular pattern of numbers
int main()
{	int i,j,row;
	printf("Enter the number of rows: ");
	scanf("%d",&row);

	for(i=1;i<=row;i++){
		for(j=1;j<=i;j++){
			printf("%d ",i);
		}
		printf("\n");
	}
	return 0;
}
```

Output:

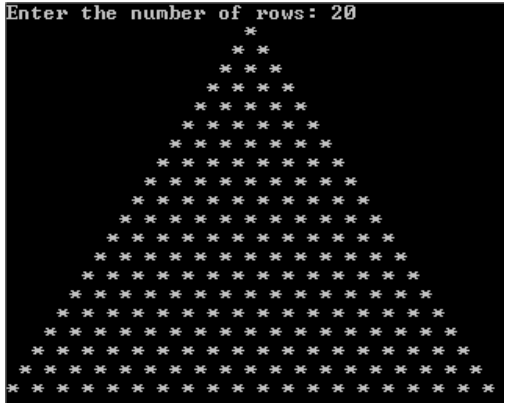
```
Enter the number of rows: 8
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
6 6 6 6 6 6
7 7 7 7 7 7 7
8 8 8 8 8 8 8 8
```

More examples of loop [contd.]


```c
#include <stdio.h>
//This program prints a triangular pattern of *
int main()
{        int i,j,row;
         printf("Enter the number of rows: ");
         scanf("%d",&row);

         for(i=1;i<=row;i++){
                  for(j=1;j<=row-i;j++){
                           printf(" ");}
                  for(j=1;j<=i;j++){
                           printf("* ");}
                  printf("\n");
         }
         return 0;
}
```

Output:



Questions?