Basic Ubuntu Commands and Syntax of C

Course: Introduction to Programming and Data Structures

Laltu Sardar

Institute for Advancing Intelligence (IAI),
TCG Centres for Research and Education in Science and Technology (TCG Crest)

tcg crest
Inventing Harmonious Future

August 13, 2024

Basic Ubuntu Commands

List files/directories and change path

For windows: install mobaxterm or WSL

1. $ `pwd` → Print Working Directory
2. $ `ls` → List : print the list of files and directories in current path
3. $ `ls <targetDirPath>` → List : print the list of files and directories in the targeted directory Path
4. $ `cd` → Change working directory to Home directory.
5. $ `cd` <targetDirPath> → Change working directory to targeted directory
6. $ `cd .` → Change to **Current** directory
7. $ `cd ..` → Change to **Parent** directory

tcg crest
Inventing Harmonious Future

Make/Delete/Copy a file/directory

- $ `cp <srcFilePath> <destFilePath>` → COPY a *File* at srcFilePath to destFilePath
- $ `cp -r <srcDirPath> <destDirPath>` → COPY a directory
- $ `exit`, `^d` → EXIT an ongoing program
- $ `mkdir <directoryName>` → MAKE the directory
- $ `rmdir <directoryName>` → REMOVE the directory
- $ `rm <fileName>` → REMOVE the file fileName
- $ `rmdir <directoryName>` → REMOVE the directory
- $ `rm -r <directoryName>` → REMOVE the directory
- $ `mv <srcFilePath> <destFilePath>` → MOVE the file

tcg crest
Inventing Harmonious Future

Printing Contents of a File

1. $ `cat <fileName>` → whole content
2. $ `head <fileName>` → HEAD of the file
3. $ `man <cmdName>` → show MANUAL of cmdName
4. Press "`q`" to Quit
5. $ `top` → Display ongoing programs
6. $ `kill -9 <programID>` → Kill the program with id programID
7. others– $ `wget, time`,

tcg crest
Inventing Harmonious Future

Basic Syntax of C Program

Why Learn C?

- C is the **foundation of many** other programming languages. If you learn C, you will have a better understanding of how other languages work.
- C is a **powerful language** that can be used to write a wide variety of programs.
- C is **fast and efficient**, which makes it ideal for writing performance-critical applications.
- C is **portable**, which means that your programs can be compiled and run on a variety of different computer platforms.
- C is a good language to learn for beginners.

tcg crest
Inventing Harmonious Future

Disadvantages of C Programming

1. **Steep learning curve**: C can be difficult to learn due to its complex syntax and low-level system access.
2. **Lack of memory management:** C lacks automatic memory management, leading to memory leaks and bugs if not handled properly.
3. **No built-in support for object-oriented programming:** C lacks built-in object-oriented programming support, making it harder to write object-oriented code.
4. **No built-in support for concurrency:** C lacks built-in concurrency support, making multithreaded applications more challenging.
5. **Security vulnerabilities:** C programs are prone to security vulnerabilities like buffer overflows if not written carefully.

tcg crest
Inventing Harmonious Future

Hello World! in C

The following code is a simple "Hello, World!" program in C:

```c
#include <stdio.h>
int main() {
    printf("Hello, World!\n");
    return 0;
}
```

1. This program first includes the `stdio.h` header file, which contains the `printf()` function.
2. The `main()` function is the entry point of the program.
3. The `printf()` function prints the string "Hello, World!" to the console.
4. The `return 0;` statement tells the operating system that the program has terminated successfully.

tcg crest
Inventing Harmonious Future

Output your Name

```
1  //FileName: namePrint.c
2  //Prints given name
3  #include <stdio.h>
4  main()
5  {
6     char name[] = "Your Name"
7     printf("hello, %s\n", name);
8  }
```

1. Compilation: gcc -g -Wall namePrint.c -o namePrint.out
   - gcc → GNU Compiler Collection
   - gcc -g → generates debug info to be used by GDB debugger
   - -Wall → Show all warnings
2. Run: ./namePrint.out
3. ".out" – not mandatory

tcg crest
Inventing Harmonious Future

The compilation process of a C program

The Stages of Compilation

The compilation process of a C program can be divided into four stages:

1. **Preprocessing**: The preprocessor is a program that expands macros and replaces #include directives with the contents of the header files.
2. **Compilation**: The compiler converts the C code into assembly language.
3. **Assembly**: The assembler converts the assembly language into machine code.
4. **Linking**: The linker combines the machine code from multiple object files into an executable file.

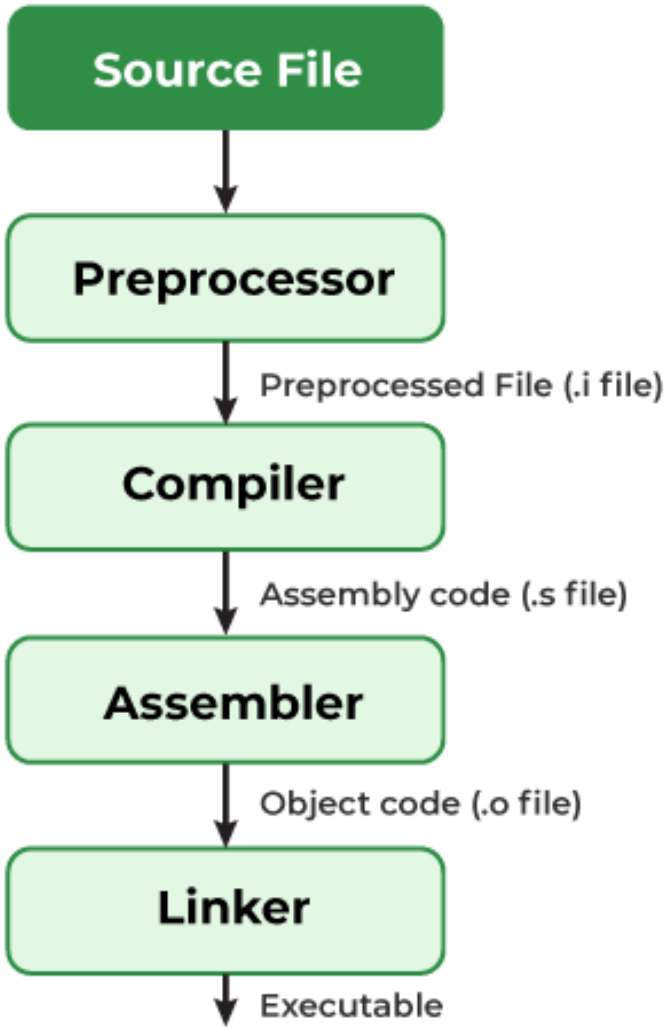
tcg crest
Inventing Harmonious Future

Executing a code

1. Normal Compilation:
   `gcc filename.c -o filename` or simply `gcc filename.c`
2. Run/execute the code:
   `./filename` or `./a.out`
3. Too see all stages:
   `gcc -g -Wall -save-temps filename.c -o filename`

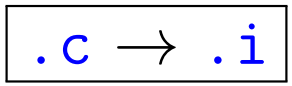
tcg crest
Inventing Harmonious Future

Execution Stages



tcg crest
Inventing Harmonious Future

The Role of the Preprocessor

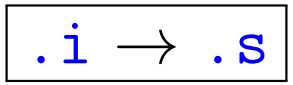


The preprocessor is a program that is run before the compiler. It is responsible for the following tasks:

- Expanding macros
- Replacing #include directives with the contents of the header files
- Performing conditional compilation
- Executing directives that control the compilation process

tcg crest
Inventing Harmonious Future

The Role of the Compiler

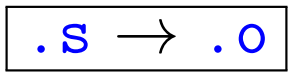


The compiler is a program that converts the C code into assembly language. It is responsible for the following tasks:

- Analyzing the C code for syntax errors
- Generating assembly language code for each statement in the C code
- Performing optimizations to the assembly language code

tcg crest
Inventing Harmonious Future

The Role of the Assembler



The assembler is a program that converts the assembly language into machine code. It is responsible for the following tasks:

- Converting each instruction in the assembly language into machine code
- Generating a symbol table for the machine code

tcg crest
Inventing Harmonious Future

The Role of the Linker

`.o` $\rightarrow$ `.out`

The linker is a program that combines the machine code from multiple object files into an executable file. It is responsible for the following tasks:

- All the linking of function calls with their definitions
- Linking the object files together
- Resolving symbol references between the object files
- Generating a relocation table for the executable file

tcg crest
Inventing Harmonious Future

Errors Detection During Compilation

• Errors in a C program can be detected during any of the four stages of compilation.
• The preprocessor, compiler, assembler, and linker all have their own set of error messages.

Type of error messages

- Syntax errors: Detected by the preprocessor and compiler. Usually caused by incorrect use of the C language syntax.
- Logical errors: Not detected by the preprocessor or compiler. Usually caused by incorrect logic in the C code.
- Runtime errors: Detected when the C program is executed. Usually caused by incorrect input data or memory corruption.

tcg crest
Inventing Harmonious Future

Commenting in C

Introduction

Comments are a way to add human-readable text to a C program. They are not executed by the compiler, but they can be helpful for understanding the code.

Single-Line Comments

- Single-line comments start with the // character and continue to the end of the line.
- For example:
  ```
  // This is a single-line comment
  ```
- Single-line comments are often used to explain the purpose of a line of code or to provide a brief explanation of what is happening.

tcg crest
Inventing Harmonious Future

Comments

Multi-Line Comments

- Multi-line comments start with the /* character and end with the */ character.
- For example:

```
/* This is a multi-line comment
that can span multiple lines */
```

- Multi-line comments are often used to document the algorithm used in a function or to describe the input and output of a program.

tcg crest
Inventing Harmonious Future

Good Practices for Using Comments

Here are some good practices for using comments in C programs:

- Use comments to **explain the purpose** of the code.
- Use comments to **document the algorithm** used in the code.
- **Describe the input and output** of the code.
- Mark out sections of code that are not yet finished.
- Add humor or personality to the code, but only if it is appropriate.
- Avoid using comments to explain what the code is doing. **The code should be self-explanatory.**
- Avoid using comments to replace documentation. Documentation should be written in a separate document.

tcg crest
Inventing Harmonious Future

Tokens in C

Introduction

- A token is the **smallest unit of program** text that the C compiler can understand.
- Tokens are made up of letters, numbers, and special symbols.

Some examples of tokens in C

- **Identifiers**: Names used to refer to variables, functions, and other entities.
- **Keywords**: Words with special meaning to the C compiler.
- **Operators**: Symbols used for operations on data.
- **Punctuators**: Symbols used to separate tokens or indicate program structure.

tcg crest
Inventing Harmonious Future

Identifiers

Identifiers are names used to refer to variables, functions, and other entities in a C program.

Rule of Identifiers

- Identifiers can be made up of letters { `a, b, ···, z, A, B,···, Z, _` }, numbers { `0,1,··· ,9`}. **Max length 31**
- The first character of an identifier must be a letter.
- Examples of valid identifiers in C:
  `my_variable` `function_name` `__constant_name`
- Examples of invalid identifiers in C:
  `123variable` `function-name` `_variable`

Practice

- Variable name should be given in such a way that usage of the variable can be guessed easily from its name.

Keywords

- Keywords are words that have special meaning to the C compiler.
- **Keywords cannot be used as identifiers.**

auto	break	case	char
const	continue	default	do
double	else	enum	extern
float	for	goto	if
int	long	register	return
short	signed	sizeof	static
struct	switch	typedef	union
unsigned	void	volatile	while

Table: Examples of keywords in C:

tcg crest
Inventing Harmonious Future

Operators in C

Operator	Description	Example
+,-	Addition, Subtraction	`a + b, a - b`
*, /	Multiplication, Division	`a * b, a / b`
%	Modulus	`a % b`
++	Increment	`a++, ++a`
--	Decrement	`a--, --a`
==	Equal to	`a == b`
!=	Not equal to	`a != b`
<, >	Less than, Greater than	`a < b, a > b`
<=	Less than or equal to	`a <= b`
>=	Greater than or equal to	`a >= b`
&&	Logical and	`a && b`
||	Logical or	`a || b`
!	Logical not	`!a`

tcg crest
Inventing Harmonious Future

Bitwise Operators in C

Operator	Description	Example	Result
&	Bitwise AND	`a & b`	Bits that are set in both `a` and `b`
|	Bitwise OR	`a | b`	Bits that are set in either `a` or `b` or both
^	Bitwise XOR	`a ^ b`	Bits that are set in one of `a` or `b` but not both
~	Bitwise NOT	`~a`	Inverts all the bits
«	Left shift	`a « n`	Shifts the bits of `a` to the left by `n` positions
»	Right shift	`a » n`	Shifts the bits of `a` to the right by `n` positions

Table: Bitwise Operators in C

tcg crest
Inventing Harmonious Future

Punctuators

are symbols –>used to separate tokens or to indicate the structure.

Punctuator	Use	Example
< >	Header name	`#include <limits.h>`
[ ]	Array delimiter	`char a[7];`
{ }	Initializer list, function body, or compound statement delimiter	`char x[4] = {'H', 'i', '!', '\0'};`
( )	Function parameter list delimiter; also used in expression grouping	`int f(x, y);`
*	Pointer declaration	`int *x;`
,	Argument list separator	`char x[4] = {'H', 'i', '!', '\0'};`
:	Statement label	`labela:  if (x == 0) x += 1;`
=	Declaration initializer	`char x[4] = {"Hi!"};`
;	Statement end	`x += 1;`
...	Variable-length argument list	`int f(int y, ...);`
#	Preprocessor directive	`#include "limits.h"`
' '	Character constant	`char x = 'x';`
" "	String literal or header name	`char x[] = "Hi!";`

Table: Punctuators in C

tcg crest
Inventing Harmonious Future

Miscellinious

Building block of a Programming Language

1. Memory = space for calculations, rough work, etc.
2. Identifier = names given to memory locations for convenience
3. Instructions = each step in the procedure

Variables and Arithmetic Expressions

```c
/* filenme: FahToCel.c
   print Fahrenheit-Celsius table
   for fahr = 0, 20, ..., 300
*/
#include <stdio.h>
main()
{
   int fahr, celsius;      //variable Declaration
   int lower, upper, step;
   lower = 0; /* lower limit of temperature scale */ // variable assignment
   upper = 300; /* upper limit */
   step = 20; /* step size */
   fahr = lower;
   while (fahr <= upper) {     //while loop
      celsius = 5 * (fahr-32) / 9;
      printf("%d\t%d\n", fahr, celsius);
      fahr = fahr + step;
   }
}
```

Description

- Variable declaration, definition, assignment
  E.g. `int a;`, `a = 10;`, `a=b`
- Each variable must have a format specifier in printf

Format Specifiers

- Format specifiers define the type of data to be printed on standard output.
- You need to use format specifiers whether you're printing formatted output with `printf()` or accepting input with `scanf()`.

Some frequently used format specifiers

1. %d – decimal integer; %f – floating point
2. %6d – decimal integer, at least 6 characters wide
3. %6f – floating point, at least 6 characters wide
4. %.2f – floating point, 2 characters after decimal point
5. %6.2f – floating point, at least 6 wide and 2 after decimal point
6. %s – string variable, %c – single character

Symbolic Constants

```
1   #include <stdio.h>
2   #define LOWER 0 /* lower limit of table */
3   #define UPPER 300 /* upper limit */
4   #define STEP 20 /* step size */
5   /* print Fahrenheit-Celsius table */
6   main()
7   {
8     int fahr;
9     for (fahr = LOWER; fahr <= UPPER; fahr = fahr + STEP)
10    printf("%3d %6.1f\n", fahr, (5.0/9.0)*(fahr-32));
11  }
```

```
1   #define name replacement list
```

- symbolic constants are *string of characters*:
- They are not variables
- they do not appear in declarations
- In compiled files, they do not exists
- Conventionally written in upper case only

tcg crest
Inventing Harmonious Future

Input from terminal **during** execution

- printf() : returns total number of Characters Printed, Or negative value if an output error or an encoding error
- scanf() : Reads input of any datatype from (stdin).
  - Stops reading when it encounters **whitespace, newline or EOF**
  - Returns total number of Inputs Scanned successfully, or EOF if input failure occurs before the first receiving argument was assigned.
- gets(): Reads string from standard input.
  - Stops stops reading input when it encounters **newline or EOF**.
  - Returns total number of Inputs Scanned successfully, or EOF if input failure occurs before the first receiving argument was assigned.

Note: gets() does not stop reading input when it encounters whitespace instead it takes whitespace as a string.

```c
// Program to compute average of two float variables
#include<stdio.h>

float average(float a, float b){
    return ((a+b)/2.0);
}

int main(){
    float a, b, avg;

    scanf("%f %f", &a, &b);  // taking input from terminal
    avg = average(a, b);     //Compauting avarage
    printf("%f",avg); //writing on terminal
    return 0;
}
```

```c
// Program to compute average of two float variables
#include<stdio.h>

float average(float a, float b){
    return ((a+b)/2.0);
}

int main(){
    float a, b, avg;

    scanf("%f %f", &a, &b);  // taking input from terminal
    avg = average(a, b);     //Compauting avarage
    printf("%f",avg); //writing on terminal
    return 0;
}
```

- Sometimes input is large–
- Sometime we have many inputs
- embedding data directly into the source code– **a bad idea and Not practical**
- We require to take input data from files.

TOP Secret to be an Expert in programming

Only Secret: Practice!

- Practice code/program writing
- Practice to solve daily eligible problems with coding
- Practice to take new coding challenges

tcg crest
Inventing Harmonious Future

Control Flow

Type of control flow

styles

C provides two styles of flow control:

1. `Branching`: Branching is deciding what actions to take.
   Example: If, if-else, if-else if-else, switch
2. `Looping`: looping is deciding how many times to take a certain action.
   Example: while loop, for loop, Do-while loop, etc.

Loop controller

1. `Break`: jump out of a loop.
2. `continue`: continues with the next iteration

tcg crest
Inventing Harmonious Future

If

```
1     if (condition) {
2       // block of code to be executed
3       //if the condition is true
4     }
```

Example:

```
1     int a = 10;
2     int b = 2;
3     if (a > b) {
4       printf("a is greater than b");
5     }
```

tcg crest
Inventing Harmonious Future

If-Else

```
if (condition) {
  // block of code to be executed
  //if the condition is True
}else{
  // block of code to be executed
  //if the condition is False
}
```

```
int a = 10;
int b = 2;
if (a > b) {
  printf("a is greater than b");
}else{
  printf("a is less than b");
}
```

If-Else in a single line:

```
condition ? expression-true : expression-false
```

```
int a = 10, b = 2;
(a > b)? printf("a is greater than b"): printf("a is less than b");
```

Else-If

```
if (test expression1) {
  // statement(s)
}
else if(test expression2) {
  // statement(s)
}
else if (test expression3) {
  // statement(s)
}
.
.
else {
  // statement(s)
}
```

```
if (marks > 85) {
  printf("First Class with Distinction");
}
else if(marks > 60) {
  printf("First Class");
}
else if (marks>40) {
  print("Passed");
}
else {
  print("Failed");
}
```

Switch: Psudocode

```
switch (expression)
{
  case constant1:
  // statements
  break;

  case constant2:
  // statements
  break;
  .
  .
  .
  default:
  // default statements
}
```

Switch: Example

```
1   char operation;
2   double n1, n2;
3   printf("Enter an operator (+, -, *, /): ");
4   scanf("%c", &operation);
5   printf("Enter two operands: ");
6   scanf("%lf %lf",&n1, &n2);
7
8   switch(operation)
9   {
10    case '+':
11    printf("%.1lf + %.1lf = %.1lf",n1, n2, n1+n2);
12    break;
13
14    case '-':
15    printf("%.1lf - %.1lf = %.1lf",n1, n2, n1-n2);
16    break;
17
18    case '*':
19    printf("%.1lf * %.1lf = %.1lf",n1, n2, n1*n2);
20    break;
21
22    case '/':
23    printf("%.1lf / %.1lf = %.1lf",n1, n2, n1/n2);
24    break;
25
26    // operator doesn't match any case constant +, -, *, /
27    default:
28    printf("Error! operator is not correct");
29  }
```

For and While

```
for ( init; condition; increment ) {
   statement(s);
}
```

```
int i;

/* for loop execution */
for( i = 1; i < 10; i = i + 1 ){
   printf("value of i: %d\n", i);
}
```

```
while(condition) {
   statement(s);
}
```

```
int i = 1;

/* while loop execution */
while( i < 10 ) {
   printf("value of i: %d\n", i);
   i++;
}
```

for and while loop

Some frequently used format specifiers

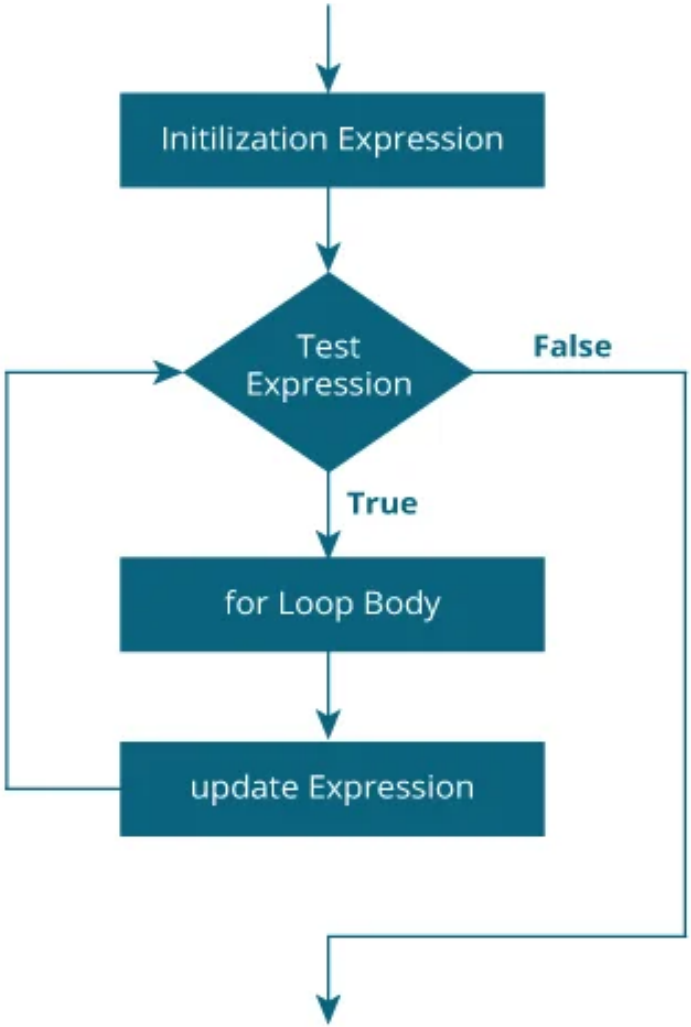
1. The for statement is a loop– a generalization of the while.
2. Three parts— separated by semicolons.
3. The first part– the initialization
4. The second part– Loop controller/ loop terminator
5. The third part– condition re-evaluation

'For' or 'while': which to use?

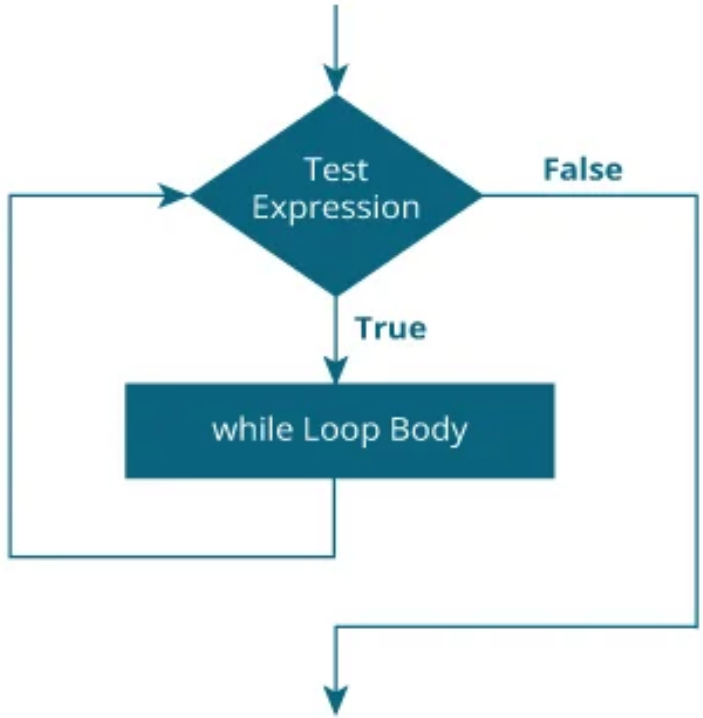
- whatever you want
- 'for' is more compact. It keeps the loop control statements together in one place

tcg crest
Inventing Harmonious Future

Flowchart of for



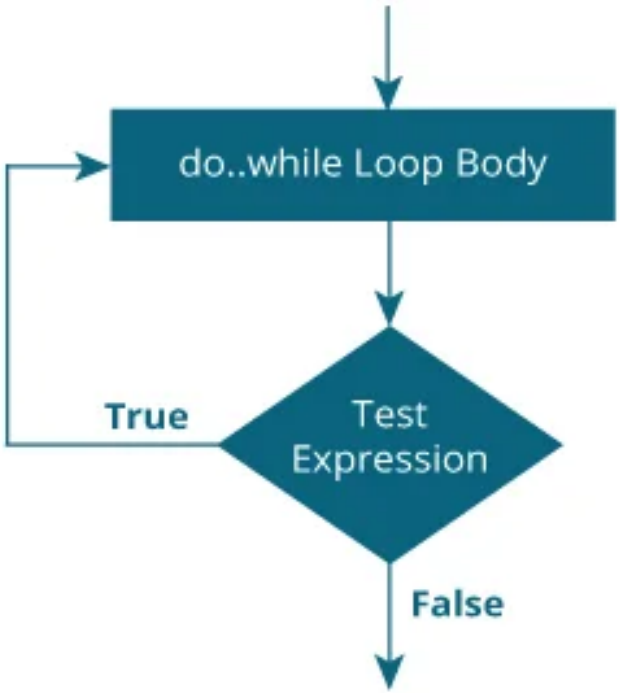
Flowchart of For loop



Flowchart of For while loop

tcg crest
Inventing Harmonious Future

Do-While



Flowchart of Do-while loop

Pseudocode:

```
do {
  // the body of the loop
}
while (testExpression);
```

Example:

```
int i;

/* for loop execution */
i = 1;
do{
  printf("value of i: %d\n", i);
  i = i + 1 ;
} while(i < 10);
```

Question: What to use For or While or Do-While?

tcg crest
Inventing Harmonious Future

Break and Continue

`break` statement **terminates** a loop

```
1     for (int i = 1; i <= 40; i++) {
2       printf("value of i: %d\n", i);
3       if (i == 10) {
4         break;  // terminates the loop
5
6       }
7     }
```

`continue` **skips** a current iteration of a loop.

```
1     for (int i = 1; i <= 10; i++{
2       printf("value of i: %d\n", i);
3       if (i == 3) {
4         continue;
5       }
6     }
```

tcg crest
Inventing Harmonious Future