Difference Between Algorithm and Flowchart

geeksforgeeks.org/difference-between-algorithm-and-flowchart

May 28, 2019

Algorithm:

The word Algorithm means “a process or set of rules to be followed in calculations or other problem-solving operations”. Therefore Algorithm refers to a set of rules/instructions that step-by-step define how a work is to be executed upon in order to get the expected results. Let’s take a look at an example for a better understanding. As a programmer, we are all aware of the Linear Search program. (Linear Search)

Algorithm of linear search :

```
1. Start from the leftmost element of arr[] and
one by one compare x with each element of arr[].
2. If x matches with an element, return the index.
3. If x doesn’t match with any of elements, return -1.
```

Flowchart: A flowchart is a graphical representation of an algorithm. Programmers often use it as a program-planning tool to solve a problem. It makes use of symbols which are connected among them to indicate the flow of information and processing. The process of drawing a flowchart for an algorithm is known as “flowcharting”. Example: Draw a flowchart to input two numbers from the user and display the largest of two numbers Let’s see the difference between algorithm and flow chart:-

S.NO	Algorithm	Flowchart
1.	Algorithm is step by step procedure to solve the problem.	Flowchart is a diagram created by different shapes to show the flow of data.
2.	Algorithm is complex to understand.	Flowchart is easy to understand.
3.	In algorithm plain text are used.	In flowchart, symbols/shapes are used.
4.	Algorithm is easy to debug.	Flowchart it is hard to debug.
5.	Algorithm is difficult to construct.	Flowchart is simple to construct.
6.	Algorithm does not follow any rules.	Flowchart follows rules to be constructed.
7.	Algorithm is the pseudo code for the program.	Flowchart is just graphical representation of that logic.

Arithmetic Operators in C

tutorialspoint.com/cprogramming/c_arithmetic_operators.htm

Operator	Description	Example
+	Adds two operands.	A + B = 30
−	Subtracts second operand from the first.	A − B = -10
*	Multiplies both operands.	A * B = 200
/	Divides numerator by de-numerator.	B / A = 2
%	Modulus Operator and remainder of after an integer division.	B % A = 0
++	Increment operator increases the integer value by one.	A++ = 11
--	Decrement operator decreases the integer value by one.	A-- = 9

Example

Try the following example to understand all the arithmetic operators available in C –

```
#include <stdio.h>

main() {

   int a = 21;
   int b = 10;
   int c ;

   c = a + b;
   printf("Line 1 - Value of c is %d\n", c );

   c = a - b;
   printf("Line 2 - Value of c is %d\n", c );

   c = a * b;
   printf("Line 3 - Value of c is %d\n", c );

   c = a / b;
   printf("Line 4 - Value of c is %d\n", c );

   c = a % b;
   printf("Line 5 - Value of c is %d\n", c );

   c = a++;
   printf("Line 6 - Value of c is %d\n", c );

   c = a--;
   printf("Line 7 - Value of c is %d\n", c );
}
```

When you compile and execute the above program, it produces the following result –

```
Line 1 - Value of c is 31
Line 2 - Value of c is 11
Line 3 - Value of c is 210
Line 4 - Value of c is 2
Line 5 - Value of c is 1
Line 6 - Value of c is 21
Line 7 - Value of c is 22
```

Compound Statements

 www2.dmst.aueb.gr/dds/res/cstyle/comp.htm

A compound statement is a list of statements enclosed by braces. There are many common ways of formatting the braces. Please be consistent with our local standard. When editing someone else's code, *always* use the style used in that code.

```
control {
                statement;
                statement;
}
```

The style above is called ``K&R style'', and is preferred if you haven't already got a favorite. With K&R style, the `else` part of an *if-else* statement and the `while` part of a *do-while* statement should appear on the same line as the close brace. With most other styles, the braces are always alone on a line.

When a block of code has several labels (unless there are a lot of them), the labels are placed on separate lines. The fall-through feature of the C *switch* statement, (that is, when there is no `break` between a code segment and the next `case` statement) must be commented for future maintenance. A lint-style comment/directive is best.

```
switch (expr) {
case ABC:
case DEF:
        statement;
        break;
case UVW:
        statement;
        /*FALLTHROUGH*/
case XYZ:
        statement;
        break;
}
```

Here, the last `break` is unnecessary, but is required because it prevents a fall-through error if another `case` is added later after the last one. The `default` case, if used, should be last and does not require a `break` if it is last.

Whenever an `if-else` statement has a compound statement for either the `if` or `else` section, the statements of both the `if` and `else` sections should both be enclosed in braces (called *fully bracketed syntax*).

```
if (expr) {
        statement;
} else {
        statement;
        statement;
}
```

Braces are also essential in *if-if-else* sequences with no second *else* such as the following, which will be parsed incorrectly if the brace after `(ex1)` and its mate are omitted:

```
if (ex1) {
        if (ex2) {
                funca();
        }
} else {
        funcb();
}
```

An *if-else* with *else if* should be written with the *else* conditions left-justified.

```
if (STREQ (reply, "yes")) {
        statements for yes
        ...
} else if (STREQ (reply, "no")) {
        ...
} else if (STREQ (reply, "maybe")) {
        ...
} else {
        statements for default
        ...
}
```

The format then looks like a generalized *switch* statement and the tabbing reflects the switch between exactly one of several alternatives rather than a nesting of statements. `Do-while` loops should always have braces around the body.

Forever loops should be coded using the `for` (;;) construct, and not the `while` (1) construct. Do not use braces for single statement blocks.

```
for (;;)
        function();
```

Sometimes an `if` causes an unconditional control transfer via `break`, `continue`, `goto`, or `return`. The `else` should be implicit and the code should not be indented.

```
if (level > limit)
        return (OVERFLOW)
normal();
return (level);
```

The ``flattened'' indentation tells the reader that the boolean test is invariant over the rest of the enclosing block.

What is C Programming Language? Basics, Introduction, History

guru99.com/c-programming-language.html

June 18, 2022

C is a general-purpose programming language that is extremely popular, simple, and flexible to use. It is a structured programming language that is machine-independent and extensively used to write various applications, Operating Systems like Windows, and many other complex programs like Oracle database, Git, Python interpreter, and more.

It is said that 'C' is a god's programming language. One can say, C is a base for the programming. If you know 'C,' you can easily grasp the knowledge of the other programming languages that uses the concept of 'C'

It is essential to have a background in computer memory mechanisms because it is an important aspect when dealing with the C programming language.

C Basic Commands

Following are the basic commands in C programming language:

C Basic commands	Explanation
#include <stdio.h>	This command includes standard input output header file(stdio.h) from the C library before compiling a C program
int main()	It is the main function from where C program execution begins.
{	Indicates the beginning of the main function.
/*_some_comments_*/	Whatever written inside this command "/* */" inside a C program, it will not be considered for compilation and execution.
printf("Hello_World! ");	This command prints the output on the screen.
getch();	This command is used for any character input from keyboard.
return 0;	This command is used to terminate a C program (main function) and it returns 0.
}	It is used to indicate the end of the main function.

Constants vs Variables in C language

geeksforgeeks.org/constants-vs-variables-in-c-language

June 27, 2019

Constant

As the name suggests the name constants is given to such variables or values in C programming language which cannot be modified once they are defined. They are fixed values in a program. There can be any types of constants like integer, float, octal, hexadecimal, character constants etc. Every constant has some range. The integers that are too big to fit into an int will be taken as a long. Now there are various ranges that differ from unsigned to signed bits. Under the signed bit, the range of an int varies from -128 to +127 and under the unsigned bit, int varies from 0 to 255.

Variable

A **variable** in simple terms is a storage place which has some memory allocated to it. Basically, a variable used to store some form of data. Different types of variables require different amounts of memory and have some specific set of operations which can be applied to them.

Variable Declaration:
A typical variable declaration is of the form:

```
type variable_name;

or for multiple variables:
type variable1_name, variable2_name, variable3_name;
```

A variable name can consist of alphabets (both upper and lower case), numbers and the underscore '_' character. However, the name must not start with a number.

Difference between variable and constant

Constants	Variable
A value that can not be altered throughout the program	A storage location paired with an associated symbolic name which has a value
It is similar to a variable but it cannot be modified by the program once defined	A storage area holds data
Can not be changed	Can be changed according to the need of the programmer
Value is fixed	Value is varying

Data Types in C

geeksforgeeks.org/data-types-in-c

June 30, 2015

Each variable in C has an associated data type. Each data type requires different amounts of memory and has some specific operations which can be performed over it. Let us briefly describe them one by one:
Following are the examples of some very common data types used in C:

- **char:** The most basic data type in C. It stores a single character and requires a single byte of memory in almost all compilers.
- **int:** As the name suggests, an int variable is used to store an integer.
- **float:** It is used to store decimal numbers (numbers with floating point value) with single precision.
- **double:** It is used to store decimal numbers (numbers with floating point value) with double precision.

Different data types also have different ranges upto which they can store numbers. These ranges may vary from compiler to compiler. Below is list of ranges along with the memory requirement and format specifiers on 32 bit gcc compiler.

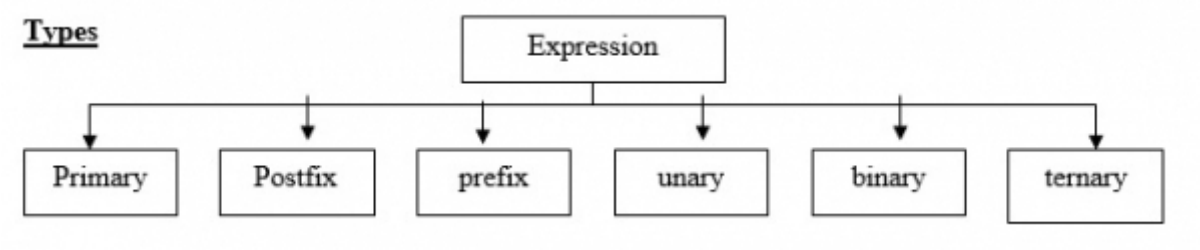
Data Type	Memory (bytes)	Range	Format Specifier
short int	2	-32,768 to 32,767	%hd
unsigned short int	2	0 to 65,535	%hu
unsigned int	4	0 to 4,294,967,295	%u
int	4	-2,147,483,648 to 2,147,483,647	%d
long int	4	-2,147,483,648 to 2,147,483,647	%ld
unsigned long int	4	0 to 4,294,967,295	%lu
long long int	8	-(2^63) to (2^63)-1	%lld
unsigned long long int	8	0 to 18,446,744,073,709,551,615	%llu
signed char	1	-128 to 127	%c
unsigned char	1	0 to 255	%c
float	4		%f
double	8		%lf
long double	16		%Lf

Explain different types of expressions in C program

An expression is a combination of operators and operands which reduces to a single value. An operation is performed on a data item which is called an operand. An operator indicates an operation to be performed on data.

For example, z = 3+2*1

z = 5



- **Primary expressions** – It is an operand which can be a name, a constant or any parenthesized expression. Example – c = a+ (5*b);
- **Postfix expressions** – In a postfix expression, the operator will be after the operand. Example – ab+
- **Prefix expressions** – n a prefix expression, the operator is before the operand. Example – +ab
- **Unary expression** – It contains one operator and one operand. Example – a++, --b
- **Binary expression** – t contains two operands and one operator. Example – a+b, c-d
- **Ternary expression** – It contains three operands and one operator. For Example, Exp1? Exp2 – Exp3. If Exp1 is true, Exp2 is executed. Otherwise, Exp3 is executed.

Example

Given below is the C program explaining the different types of expressions in C language –

Live Demo

```
#include<stdio.h>
int main(){
   int a,b,c,d,z;
   int p,q,r,s,t,u,v;
   printf("enter the values of a,b,c,d:\n");
   scanf("%d%d%d%d",&a,&b,&c,&d);
   r=a++;
   s=--b;
   t=a+b;
   u=c-d;
   v=a+(5*b);
   z = (5>3) ? 1:0;
   printf("unaryexpression=%d\nunary expression=%d\n Binary
   expression=%d\nBinary expression=%d\nPrimary expression=%d\nTernary
expression=%d\n",r,s,t,u,v,z);
}
```

Output

You will see the following output –

```
enter the values of a,b,c,d:
2 3 4 6
unary expression=2
unary expression=2
Binary expression=5
Binary expression=-2
Primary expression=13
Ternary expression=1
```

What are the C library functions?

Library functions are built-in functions that are grouped together and placed in a common location called library.

Each function here performs a specific operation. We can use this library functions to get the pre-defined output.

All C standard library functions are declared by using many header files. These library functions are created at the time of designing the compilers.

We include the header files in our C program by using **#include<filename.h>**. Whenever the program is run and executed, the related files are included in the C program.

Header File Functions

Some of the header file functions are as follows –

- **stdio.h** – It is a standard i/o header file in which Input/output functions are declared
- **conio.h** – This is a console input/output header file.
- **string.h** – All string related functions are in this header file.
- **stdlib.h** – This file contains common functions which are used in the C programs.
- **math.h** – All functions related to mathematics are in this header file.
- **time.h** – This file contains time and clock related functions.Built functions in stdio.h

Built functions in stdio.h

Let's see what are the built functions present in stdio.h library function.

Sl.No	Function & Description
1	**printf()** This function is used to print the all char, int, float, string etc., values onto the output screen.
2	**scanf()** This function is used to read data from keyboard.
3	**getc()** It reads character from file.
4	**gets()** It reads line from keyboard.
5	**getchar()** It reads character from keyboard.

Sl.No	Function & Description
6	**puts()** It writes line to o/p screen.
7	**putchar()** It writes a character to screen.
8	**fopen()** All file handling functions are defined in stdio.h header file.
9	**fclose()** Closes an opened file.
10	**getw()** Reads an integer from file.
11	**putw()** Writes an integer to file.
12	**fgetc()** Reads a character from file.
13	**putc()** Writes a character to file.
14	**fputc()** Writes a character to file.
15	**fgets()** Reads string from a file, one line at a time.
16	**f puts()** Writes string to a file.
17	**feof()** Finds end of file.
18	**fgetchar** Reads a character from keyboard.
19	**fgetc()** Reads a character from file.
20	**fprintf()** Writes formatted data to a file.
21	**fscanf()** Reads formatted data from a file.
22	**fputchar** Writes a character from keyboard.
23	**fseek()** Moves file pointer to given location.

Sl.No	Function & Description
24	**SEEK_SET** Moves file pointer at the beginning of the file.
25	**SEEK_CUR** Moves file pointer at given location.
26	**SEEK_END** Moves file pointer at the end of file.
27	**ftell()** Gives current position of file pointer.
28	**rewind()** Moves file pointer to the beginning of the file.
29	**putc()** Writes a character to file.
30	**sprint()** Writes formatted output to string.
31	**sscanf()** Reads formatted input from a string.
32	**remove()** Deletes a file.
33	**flush()** Flushes a file.

C - Program Structure

tutorialspoint.com/cprogramming/c_program_structure.htm

Before we study the basic building blocks of the C programming language, let us look at a bare minimum C program structure so that we can take it as a reference in the upcoming chapters.

Hello World Example

A C program basically consists of the following parts –

- Preprocessor Commands
- Functions
- Variables
- Statements & Expressions
- Comments

Let us look at a simple code that would print the words "Hello World" –

Live Demo

```
#include <stdio.h>

int main() {
   /* my first program in C */
   printf("Hello, World! \n");

   return 0;
}
```

Let us take a look at the various parts of the above program –

- The first line of the program *#include <stdio.h>* is a preprocessor command, which tells a C compiler to include stdio.h file before going to actual compilation.
- The next line *int main()* is the main function where the program execution begins.
- The next line /*...*/ will be ignored by the compiler and it has been put to add additional comments in the program. So such lines are called comments in the program.
- The next line *printf(...)* is another function available in C which causes the message "Hello, World!" to be displayed on the screen.
- The next line **return 0;** terminates the main() function and returns the value 0.

Compile and Execute C Program

Let us see how to save the source code in a file, and how to compile and run it. Following are the simple steps –

- Open a text editor and add the above-mentioned code.
- Save the file as *hello.c*
- Open a command prompt and go to the directory where you have saved the file.
- Type *gcc hello.c* and press enter to compile your code.
- If there are no errors in your code, the command prompt will take you to the next line and would generate *a.out* executable file.
- Now, type *a.out* to execute your program.
- You will see the output *"Hello World"* printed on the screen.

```
$ gcc hello.c
$ ./a.out
Hello, World!
```

Make sure the gcc compiler is in your path and that you are running it in the directory containing the source file hello.c.

Advertisements