Keywords in C

- Keywords are specific reserved words in C each of which has a specific feature associated with it.
- Almost all of the words which help us use the functionality of the C language are included in the list of keywords.
- There is a total of 32 keywords in C:

1. auto
2. break
3. case
4. char
5. const
6. continue
7. default
8. do
9. double
10. else
11. enum
12. extern
13. float
14. for
15. goto
16. if
17. int
18. long
19. register
20. return
21. short
22. signed
23. sizeof
24. static
25. struct
26. switch
27. typedef
28. union
29. unsigned
30. void
31. volatile
32. while

Identifier in C

- Identifier refers to name given to entities such as variables, functions, structures etc.
- These are user defined names.
- Identifier names must differ in spelling and case from any keywords.

For example:

```
int money;
double accountBalance;
```

Here, `money` and `accountBalance` are identifiers.

Also remember, identifier names must be different from keywords. You cannot use `int` as an identifier because `int` is a keyword.

Rules for naming identifiers

1. A valid identifier can have letters (both uppercase and lowercase letters), digits and underscores.
2. The first letter of an identifier should be either a letter or an underscore.
3. You cannot use keywords like int, while etc. as identifiers.
4. There is no rule on how long an identifier can be. However, you may run into problems in some compilers if the identifier is longer than 31 characters.

Variables & Constants

- In programming, a variable is a container (storage area) to hold data.
- Variables has some memory allocated to it
- Different types of variables require different amounts of memory

Variable Declaration:

for single variable:

```
type variable_name;
```

for multiple variables:

```
type variable1_name, variable2_name, variable3_name;
```

Example: Height, age, are the meaningful variables that represent the purpose it is being used for. Height variable can be used to store a height value. Age variable can be used to store the age of a person

Following are the rules that must be followed while creating a variable:

1. A variable name should **consist of only characters, digits and an underscore**.
2. A variable name should **not begin with a number**.
3. A variable name should **not consist of whitespace**.
4. A variable name should **not consist of a keyword**.
5. 'C' is a **case sensitive language** that means a variable named 'age' and 'AGE' are different.

Following are the examples of **valid variable names** in a 'C' program:

```
height or HEIGHT
_height
_height1
Ha_sh
```

Following are the examples of **invalid variable names** in a 'C' program:

```
1height
Hei$ght
Ha sh
```

For example, we declare an integer variable **height** and assign it the **value 48**:

```
int height;
```

```
height = 6;
```

or

```
int height = 6;
```

Constants:

- If you want to define a **variable whose value cannot be changed**, you can use the const keyword. This will create a constant. For example,

```
const double PI = 3.14;
```

- Notice, we have added **keyword const**.
- Here, PI is a symbolic constant; its value cannot be changed.

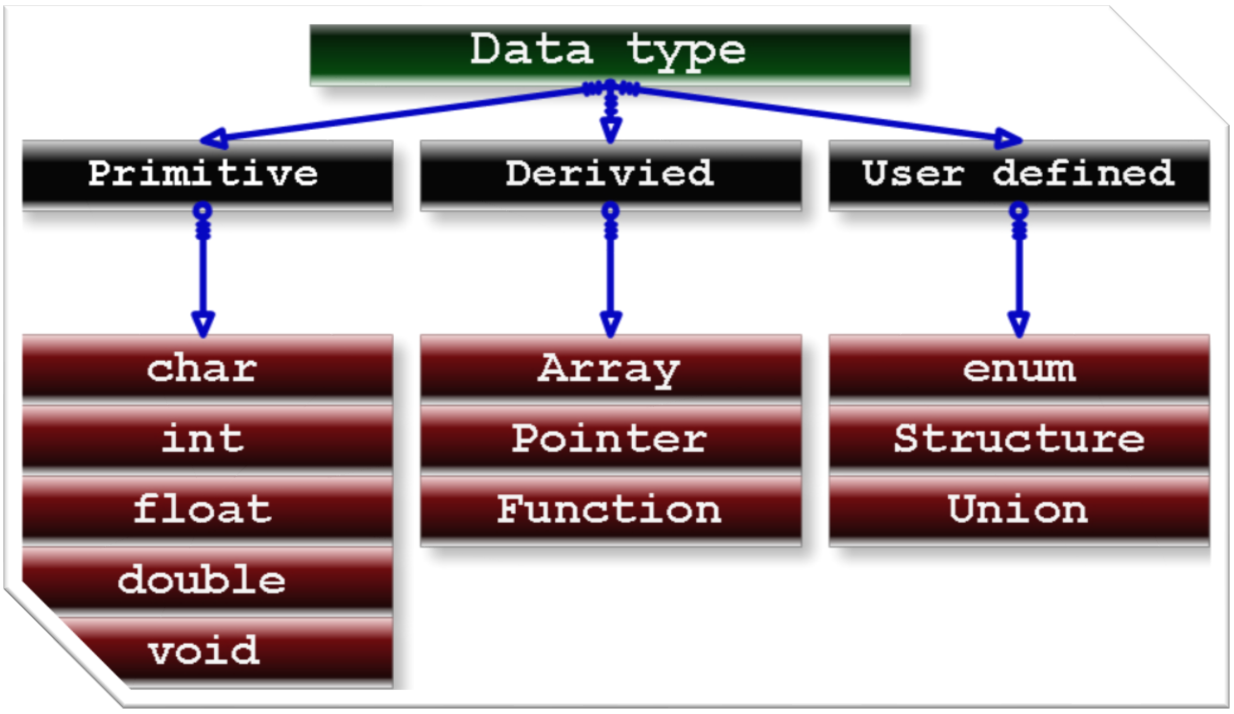
```
const double PI = 3.14;

PI = 2.9; //Error
```

Data types

'C' provides various data types to make it easy for a programmer to select a suitable data type as per the requirements of an application. Following are the three data types:

1. Primitive data types
2. Derived data types
3. User-defined data types



Each data type differs from one another in size and range. Following table displays the size and range of each data type.

Data type	Size in bytes	Range	Format Specifier
Char or signed char	1	-128 to 127	%c
Unsigned char	1	0 to 255	%c
int or signed int	2	-32768 to 32767	%d, %i
Unsigned int	2	0 to 65535	%u
Short int or Unsigned short int	2	0 to 255	%hd
Signed short int	2	-128 to 127	%hd
Long int or Signed long int	4	-2147483648 to 2147483647	%ld, %li
Unsigned long int	4	0 to 4294967295	%lu
float	4	3.4E-38 to 3.4E+38	%f
double	8	1.7E-308 to 1.7E+308	%lf
Long double	10	3.4E-4932 to 1.1E+4932	%Lf

There are five basic data types which we mostly going to use are,

1. **int** for integer data

- This data type is used to define an integer **number** (-….-3,-2,-1,0,1,2,3….).
- A single integer occupies **2 bytes**.
- An integer data type is further divided into other data types such as short int, int, and long int.
- Whenever we want to use an integer data type, we have place int before the identifier such as,

```
int age;
```

- Here, age is a variable of an integer data type which can be used to store integer values.

2. **float** for floating point numbers

   - The 'float' keyword is used to represent the floating-point data type.
   - It can hold a floating-point value which means a number is having a **fraction and a decimal part**.
   - float and double are used to hold real numbers.

```
float salary;
double price;
```

   - In C, floating-point numbers can also be used in **exponential**. For example,

```
float normalizationFactor = 22.442e2;
```

   - What's the difference between **float** and **double**?
   - The size of float (single precision float data type) is 4 bytes. And the size of double (double precision float data type) is 8 bytes.

3. **char** for character data

- Used to define **characters**.
- A single character occupies **1 byte**.
- Character value enclosed in single quotes.

  Example,

```
char letter;
char event = ‘hash’;
```

4. **void**

- void is an **incomplete type**. It means "nothing" or "no type".
- You can think of void as absent.
- For example, if a function is **not returning anything**, its return type should be void.
- Note that, you cannot create variables of void type.

  Example,

  void sum (int a, int b); – This function won’t return any value to the calling function.

  int sum (int a, int b); – This function will return value to the calling function.

C Output PRINTF FUNCTION

In C programming, **printf()** is one of the **main output function**. The function sends formatted output to the screen. For example,

Example

```
#include <stdio.h>
int main()
{
    // Displays the string inside quotations
    printf("HASH 1.0");
    return 0;
}
```

Output

```
HASH 1.0
```

How does this program work?

- C programs must contain a **main()** function. Every C program code execution begins from the **main()** function.
- **printf()** is a library function to send output to the screen. The function prints the string inside quotations **“ ”**.
- To use **printf()** in your program, we need to include **<stdio.h>** header file.
- The **return 0;** statement inside the **main()** function is the "Exit status" of the program. It's optional.

C Input SCANF FUNCTION

In C programming, **scanf()** is one of the **commonly used function to take input** from the user. The **scanf()** function reads formatted input from the standard input such as keyboards.

Example:

```
#include <stdio.h>
int main()
{
    int a;
    printf("Enter an integer: ");
    scanf("%d", &a);
    printf("Number = %d",a);
    return 0;
}
```

Output

```
Enter an integer: 4
Number = 4
```

C Operators

An operator is a **symbol which tells the compiler to perform any specific mathematical or logical functions**.

C language has a lot of in built-in operators and provides the following are the types of operators −

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators

C Arithmetic Operators

An arithmetic operator **performs mathematical operations** such as **addition, subtraction, multiplication, division** etc. on numerical values (constants and variables).

Operator	Meaning of Operator
+	addition or unary plus

-	subtraction or unary minus
*	multiplication
/	division
%	remainder after division (modulo division)

Example:

```
#include <stdio.h>
int main()
{
    int a = 9,b = 4, c;

    c = a+b;
    printf("a+b = %d \n",c);
   return 0;
}
```

Output

```
a+b = 13
```

Relational Operators

They are **used to show relation between two Variables**. The following table shows all the relational operators supported by C. Let's, assume variable **A** holds 10 and variable **B** holds 20 then −

Operator	Description	Example
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.	(A <= B) is true.

Example:

```
#include <stdio.h>
int main()
{
    int a = 5, c;
    c = a;
    printf("c = %d\n", c);
return 0;
}
```

Output

```
c = 5
```

Logical Operators

These operators are **used for Decision making**. Logical operator **returns either 0 or 1** depending upon whether expression results true or false. Following table shows all the logical operators supported by C language. Assume variable **A** holds 1 and variable **B** holds 0, then –

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A && B) is true.

Example:

```
#include <stdio.h>
int main()
{
    int a = 5, b = 5, c = 10, result;

    result = (a == b) && (c > b);
    printf("(a == b) && (c > b) is %d \n", result);
return 0;
}
```

Output

```
(a == b) && (c > b) is 1
```

Bitwise Operators

Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ is as follows –

p	**q**	**p & q AND**	**p | q OR**	**p ^ q NOT**
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

The following table lists the bitwise operators supported by C. Assume variable 'A' holds 60 and variable 'B' holds 13, then –

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, i.e., 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, i.e., 0011 0001
~	Binary One's Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = ~(60), i.e,. -0111101
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240 i.e., 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15 i.e., 0000 1111

Example:

```
#include <stdio.h>
int main()
{
    int a = 12, b = 25;
    printf("Output = %d", a&b);
    return 0;
}
```

Output

```
Output = 8
```

Assignment Operators

Assignment Operators are **<u>used to assign values to a variable</u>**. The following table lists the assignment operators supported by the C language −

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand	C = A + B will assign the value of A + B to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C - A
***=**	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	Bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	Bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

Example:

```
#include <stdio.h>
int main()
{
    int a = 5, c;
    c = a;
    printf("c = %d\n", c);
return 0;
}
```

Output

```
c = 5
```