by SAHIL



# Constants, Variables and Data Types

# The C Character Set

The characters in C are grouped into the following categories.

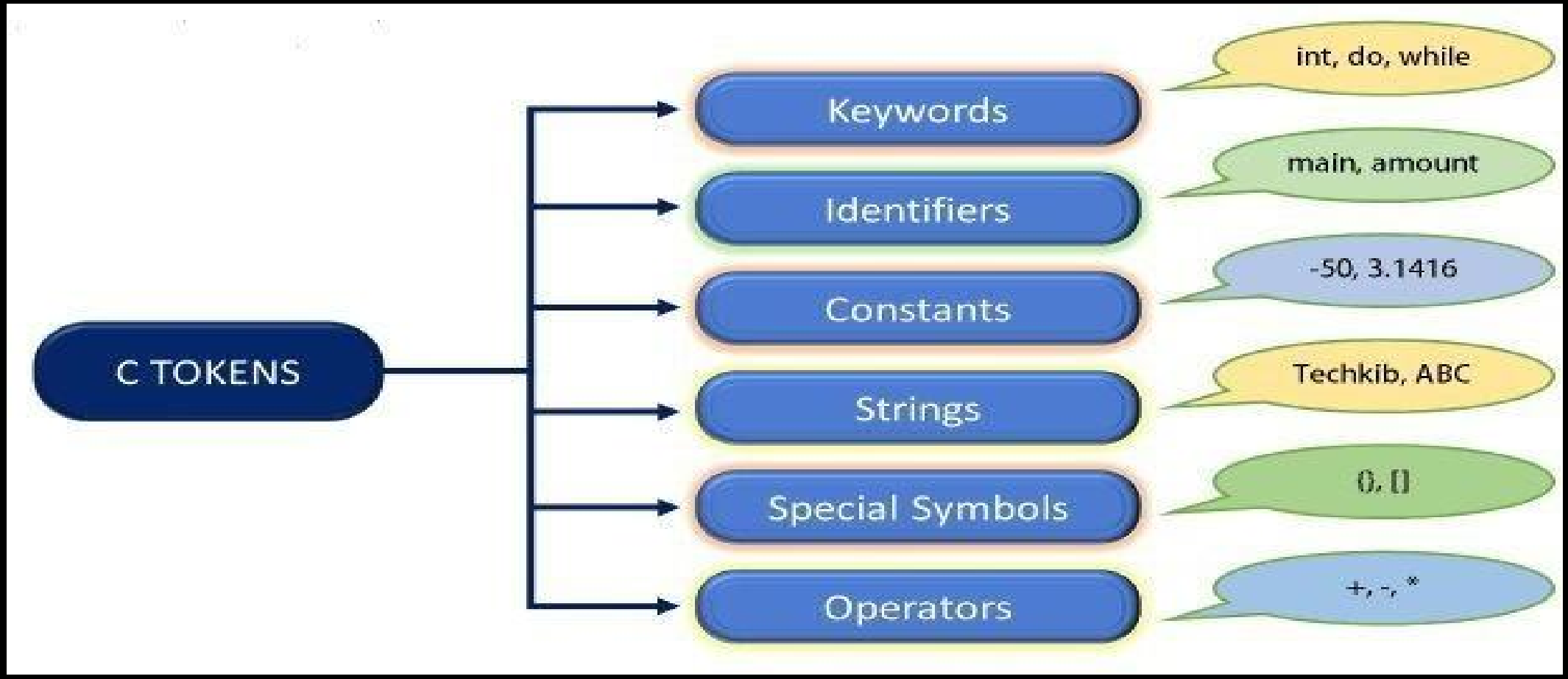
1. Letters [A-Z] [a-z]
2. Digits [0-9]
3. Special characters [@,#,$,%]
4. White spaces

**Note:**

1. The compiler ignores the white spaces unless they are a part of string constant.
2. These are prohibited between the characters of keywords and identifiers.

# C Tokens

**The *smallest individual units* used in a C program are known as *C tokens*. C has *six* types of tokens.**



# KEYWORDS AND IDENTIFIERS

1. Every C word is classified as either a *keyword* or *identifier*.
2. All keywords have fixed meanings and these meanings cannot be changed.
3. All keywords must be written in lowercase.

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

# KEYWORDS AND IDENTIFIERS

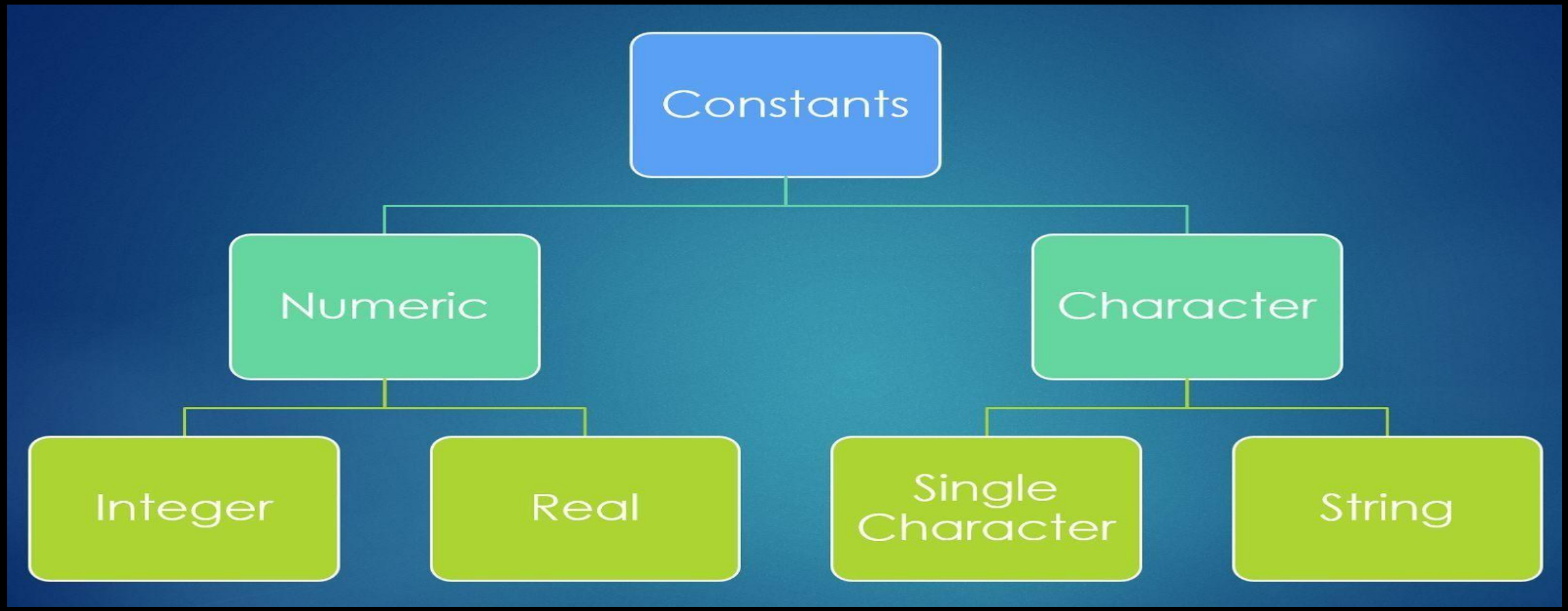
4. Identifiers refer to the names of variables, functions and arrays.
5. These are user defined names and consist of sequence of letters and digits.

## Rules for Identifiers

1. First character must be an alphabet (or underscore).
2. Must consist of only letters, digits or underscore.
3. Only first 31 characters are significant.
4. Cannot use keyword.
5. Must not contain whitespace.

# Constants in C

**Constants in C refer to fixed values that do not change during execution of a program.**



# Variables

1. A *variable* is a data name that may be used to store a data value.
2. A variable may take a different values at different times during execution.
3. It is advised to use meaningful names for variables.
4. *Example:* Average, height, Total, Counter_1, class_strength
5. As variable names are known as identifiers, therefore a programmer must adhere to the rules of identifiers for variable names.
6. Examples of valid variable names: *John, x1, mark, Delhi, ph_value.*
7. Examples of invalid variable names: *123, 25th, (area), %*

# DATA TYPES

1. ANSI C supports three classes of data types;
    a. Primary (or fundamental) data types
    b. Derived data types
    c. User-defined data types
2. All C compiler supports five fundamental data types;
    a. Integer (int)
    b. Floating point (float)
    c. Character (char)
    d. Double precision floating point (double)
    e. Void

# Integer Types

1. The number without any decimal point known as Integers.
2. They are also known as whole numbers.
3. (int) is a general keyword to declare the integer variable.
4. To give better memory manipulation c have Integer Family,

| Data Type | keyword | Memory size |
|---|---|---|
| Short Integer | short int | 2 bytes |
| Integer | int | 4 bytes |
| Long Integer | long int | 8 bytes |
| Long Long Integer | long long int | 10 bytes |

# Floating point Data Type

1. The number with any decimal point is known as floating point numbers.
2. The number consist ‘ . ’ anywhere.
3. float is a general keyword to declare floating point variable.
4. To give better memory manipulation c have Integer Family,

| Data type | keyword | Memory size |
|---|---|---|
| Floating Point | float | 4 bytes |
| Double Floating Point | double | 8 bytes |

# Void Data Type

1. The *void* type has no values.
2. Generally used to declare functions which do not return anything.
3. void is a general keyword.

# Character Data Type

1. A single character is known as Character Constant.

1. They always written inside single quotes [ ‘ ’ ].

3. char is general keyword to declare character type varables.

4. It takes 1 bytes memory size.

5. char is a general keyword to declare character type variable.

# DECLARATION OF VARIABLES

1. The declaration of a variable does two things,
   a. *It te ls the compiler what the variable name is.*
   b. *It specifies what type of data wi l the variable hold.*
2. The declaration of variables must be done before they are used.

**Primary Type Declaration:**

1. The declaration of variables take the general form as,

```
data-type v1,v2,v3…….vn;
```

# DECLARATION OF VARIABLES

2. Where v1, v2, v3….vn are the variables of the type mentioned before them.
3. The variables must be separated by commas.
4. The declaration statements must be terminated by semicolon.

**Example:**

```
int count;
int number, total;
double ratio;
```

# ASSIGNING VALUES TO VARIABLES

1. The values to the variables can be assigned in two ways,
2. Using Assignment Statement
3. Through keyboard using *scanf* function

**Using Assignment Statement:**

1. It takes the general form as,

```
variable_name = constant;
```

# ASSIGNING VALUES TO VARIABLES

**Examples:**

```
final_value     =  100;
balance         =  75.84;
yes             =  'x';
```

2. C permits the assignment of a variable with its declaration. It take the general form as:

```
data-type    variable_name    =    constant;
```

**Examples:**

```
int final_value     =  100;
double balance      =  75.84;
char yes            =  'x';
```

# ASSIGNING VALUES TO VARIABLES

3. The process of giving initial values to variables is called initialization.
4. C permits the initialization of more than one variables in one statement.

**Examples:**

```
p = q = s;
x = y = z = max
```

**Through Keyboard:**

1. The values to the variables can be given using a built-in general input function *scanf()*.

```
scanf("formate specifiers",&variable);
```

# ASSIGNING VALUES TO VARIABLES

**Examples:**

```
int age;
scanf("%d",&age);
```

**Examples:**

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

**2. When scanf statement is encountered by the compiler the execution comes to the halt till the respective data value is typed in and return key is pressed.**

# Accessing VALUES of VARIABLES

1. Generally we can access the values of variables using printf() function.
2. It is a general purpose output function.

**Examples:**

```
int age;
scanf("%d",&age);
printf("%d",age);
```

# Accessing VALUES of VARIABLES

**Examples:**

```
int a,b,c;
scanf("%d %d %d",&a,&b,&c);
printf("%d %d %d",a,b,c);
                or
printf("%d\n%d\n%d\n",a,b,c);
                or
printf("%d\n",a);
printf("%d\n",b);
printf("%d\n",c);
```

# Declaring symbolic constants

1. Defining symbolic constants helps improve *modifiability* and *understandability*.
2. They are defined with another *preprocessor* directive *#define*.
3. It takes the general form as follows:

```
#define    name_of_constant    value_of_constant
```

**Examples:**

```
#define STRENGTH 100
#define PASS_MARK 50
#define MAX 200
#define PI 3.1459
```

# Declaring variable constants

**The value of a variable can be kept constant during execution of a program if needed by introducing a qualifier *const* at the start of declaration statement.**

```
const  data-type  variableName = value;
```

**Examples:**

```
const int size = 100;
const char ch = 'A'
const float temp = 35.5;
```

return 0;



# Thank You

by SAHIL