Department of Computer Science and Engineering,
PES University, Bangalore, India

Lecture Notes
Problem Solving With C
UE24CS151B

Lecture #5
Variables and Data types

By,
Prof. Sindhu R Pai,
Theory Anchor, Feb-May, 2025
Assistant Professor
Dept. of CSE, PESU

Many Thanks to
Dr. Shylaja S S (Director, CCBD and CDSAML Research Center, PES University)
Prof. Nitin V Poojari (Dean, Internal Quality Assurance Cell, PES University)

Unit #: 1

Unit Name: Problem Solving Fundamentals

Topic: Variables and Data types

Course objectives: The objective(s) of this course is to make students

- Acquire knowledge on how to solve relevant and logical problems using computing Machine.
- Map algorithmic solutions to relevant features of C programming language constructs.
- Gain knowledge about C constructs and its associated ecosystem.
- Appreciate and gain knowledge about the issues with C Standards and it's respective behaviours.

Course outcomes: At the end of the course, the student will be able to:

- Understand and Apply algorithmic solutions to counting problems using appropriate C Constructs.
- Understand, Analyze and Apply sorting and Searching techniques.
- Understand, Analyze and Apply text processing and string manipulation methods using Arrays, Pointers and functions.
- Understand user defined type creation and implement the same using C structures, unions and other ways by reading and storing the data in secondary systems which are portable.

Sindhu R Pai

Theory Anchor, Feb - May, 2025

Dept. of CSE,

PES University

Basic constructs in C

C program consists of various tokens and a **token can be any identifier -> a keyword, variable, constant or any symbol.** For example, the following C statement consists of five tokens.

printf("HelloFriend
s");The individual tokens
are − printf
(
"HelloFriends"
)
;

Identifiers

An identifier is a **name used to identify a variable, function, or any other user-defined item**. An identifier **starts with a letter A to Z, a to z, or an underscore '_' followed by zero or more letters, underscores and digits (0 to 9)**. C does not allow few punctuation characters such as #, @ and % within identifiers. As C is **a case-sensitive programming language**, Manpower and manpower are two different identifiers in C.

Here are some examples of acceptable identifiers

_sum stud_name a_123 myname50 temp j count123 printf

Keywords

The keywords are identifiers which have special **meaning in C** and hence cannot be used as constants or variables. There are **32 keywords in C Language**.

Keywords in C Programming			
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

Variables

This is the most important concept in all languages. Variable is **a name given to a storage area that our programs can manipulate.** It has a **name, a value, a location and a type**. It also has something **called life, scope, qualifiers** etc. Variables are used to store information to be referenced and manipulated in a computer program. It provides a way of labeling data with a descriptive name, so our programs can be understood more clearly by the reader and ourselves. It is helpful to think of Variables as **containers that hold data or information**. **Purpose of Variables is to label and store data in memory, which then can be used throughout the program.**

Sometimes, the runtime makes variables with no names. These are called temporary variables. We cannot access them by name as we have no name in our vocabulary**. A variable has a type.** In 'C'**, type should be specified before a variable is ever used**. We define the variable with respect to type before using it. We cannot define the variable more than once.

Examples: int a; double b;

Rules for naming a Variable aka User defined Identifier:

A Variable name **cannot be same as the keyword** in c Language. A Variable name **should not be same as the standard identifiers** in c Language. A Variable name can **only have letters (both uppercase and lowercase letters), digits and underscore**. The **first letter** of a Variable should be either a **letter or an underscore**. There is no rule on how long a variable name aka user defined identifier can be. Variable name may run into problems in some compilers, if the variable name is longer than 31 characters. C is a **strongly typed language**, which means that the variable type cannot be changed once it is declared.

The type of a variable can never change during the program execution. The type decides what sort of values this variable can take and what operations we can perform. Type is a compile time mechanism. **The size of a value of a type is implementation dependent and is fixed for a particular implementation.**

Variable is associated with three important terms: **Declaration, Definition and Initialization**.

- **Declaration of a variable** is for informing to the compiler about the **name of the variable and the type of value it holds.** Declaration gives details about the properties of a variable.
- **Definition of a variable**: The variable gets stored. i.e., memory for the variable is allocated during the definition of the variable.

  If we want to only declare variables and not to define it i.e. we do not want to allocate memory, then the following declaration can be used.

  extern int a; // We will discuss this in Unit – 2
- **Initialization of a variable:** We can initialize a variable at the point of definition. **An uninitialized variable within a block has some undefined value**. 'C' does not initialize any default value.

```
int c = 100;
int d; // undefined value !!  variable can be assigned a value later in the code
int c = 200;
int c;  // Declaration again throws an error
d = 300;
```

Note: Assignment is considered as an expression in 'C' .

Data Types

In programming, data type is a classification that specifies type of value and what type of mathematical, relational or logical operations can be applied to it without causing an error. It deals with the **amount of storage to be reserved for the specified variable**. **Significance** of data types are given below:

1) Memory allocation
2) Range of values allowed
3) Operations that are bound to this type
4) Type of data to be stored

Size of a type:

The size required to represent a value of that type. Can be found using an operator called sizeof. **The size of a type depends on the implementation.** We should never conclude that the size of an int is 4 bytes. Can you answer this question –How many pages a book has? What is the radius of Dosa we get in different eateries?

The **sizeof operator** is the most common operator in C. It is a **compile-time unary operator** and is used to compute the size of its operand. It returns the size of a variable/value/type. It can be applied to any data type, float type, pointer type variables When sizeof() is used with the data types, **it simply gives the amount of memory allocated to that data type**. The output can be different on different machines like a 32-bit system can show different output while a 64-bit system can show different of same data types.

The size of a type is decided on an implementation based on efficiency. C standards follow the below Rules.

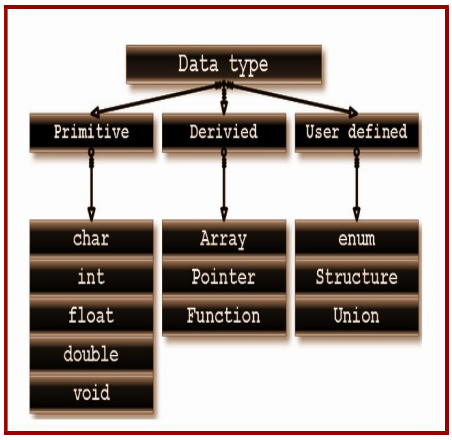
sizeof(short int)<=sizeof(int)<=sizeof(long int)<=sizeof(long long int)<= sizeof(float)<=sizeof(double)<=sizeof(long double)

Classification of Data types

Data types are categorized into **primary and non-primary (secondary) types**.

Primary ---> int, float, double, char, void

Secondary---> Derived and User-defined types



Data types can be extended using qualifiers:

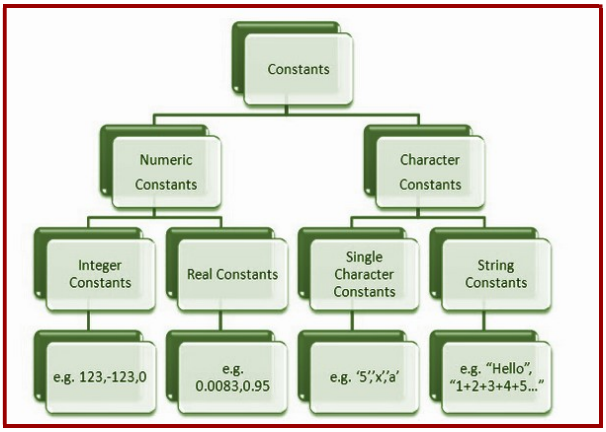
Size and Sign Qualifiers. - This will be discussed in Lecture-6.

Literals

Literals are the constant values assigned to the constant variables. There are four types of literals that exist in c programming:

1. **Integer literal**
2. **Float literal**
3. **Character literal**
4. **String literal –** Will be discussed in Unit – 2

Constant is basically a named memory location in a program that holds a single value throughout the execution of that program



Integer Literal: It is a numeric literal that represents only integer type values. Represents the value neither in fractional nor exponential part. Can be specified in the following three way.s

Decimal number (base 10): It is defined by representing the digits between 0 to 9. Example: 1,3,65 etc

Octal number (base 8): It is defined as a number in which 0 is followed by digits such as 0,1,2,3,4,5,6,7. Example: 032, 044, 065, etc

Hexadecimal number (base 16): It is defined as a number in which 0x or 0X is

followed by the hexadecimal digits (i.e., digits from 0 to 9, alphabetical characters from (a-f) or (A-F))

Example: 0XFE, oxfe etc..

Float Literal: It is a literal that contains only **floating-point values or real numbers**. **These real numbers contain the number of parts such as integer part, real part, exponential part, and fractional part.** The floating-point literal must be specified either in **decimal or in exponential form.**

Decimal form: Must contain decimal **point, real part, or both.** If it does not contain either of these, then the compiler will throw an error. The decimal notation can be prefixed either by '+' or '-' symbol that specifies the positive and negative numbers. Example: +9.5, -18.738

Exponential form: The exponential form is useful when we want to represent the number, which is having a big magnitude. It contains **two parts, i.e., mantissa and exponent**. Example: the number is 3450000000000, and it can be expressed as 3.45e12 in an exponential form

Rules for creating an exponential notation: The mantissa can be specified either in decimal or fractional form An exponent can be written in both uppercase and lowercase, i.e., e and E.

We can use both the signs, i.e., positive and negative, before the mantissa and exponent. Spaces are not allowed.

Character Literal: Contains a **single character enclosed within single quotes**. If we try to store more than one character in a character literal, then the warning of a multi-character character constant will be generated. A character literal can be represented in the following ways:

It can be represented by specifying a single character within single quotes.

Example: 'x', 'y', etc.

We can specify the escape sequence character within single quotes to represent a character literal.

Example, '\n', '\t', '\b'.

We can also use the ASCII in integer to represent a character literal.

Example: the ascii value of 65 is 'A'.

The octal and hexadecimal notation can be used as an escape sequence to represent a character literal.

Example, '\043', '\0x22'.

Think! What is the maximum and minimum integer value I can store in a variable?

The limits.h header file determines various properties of the various variable types. The macros defined in this header, limits the values of various variable types like char, int and long. These limits specify that a variable cannot store any value beyond these limits The values indicated in the table are implementation-specific and defined with the #define directive, but these values may not be any lower than what is given in the table.

Macro	Value	Description
CHAR_BIT	8	Defines the number of bits in a byte.
SCHAR_MIN	-128	Defines the minimum value for a signed char.
SCHAR_MAX	+127	Defines the maximum value for a signed char.
UCHAR_MAX	255	Defines the maximum value for an unsigned char.
CHAR_MIN	-128	Defines the minimum value for type char and its value will be equal to SCHAR_MIN if char represents negative values, otherwise zero.
CHAR_MAX	+127	Defines the value for type char and its value will be equal to SCHAR_MAX if char represents negative values, otherwise UCHAR_MAX.
MB_LEN_MAX	16	Defines the maximum number of bytes in a multi-byte character.
SHRT_MIN	-32768	Defines the minimum value for a short int.
SHRT_MAX	+32767	Defines the maximum value for a short int.
USHRT_MAX	65535	Defines the maximum value for an unsigned short int.
INT_MIN	-2147483648	Defines the minimum value for an int.
INT_MAX	+2147483647	Defines the maximum value for an int.
UINT_MAX	4294967295	Defines the maximum value for an unsigned int.
LONG_MIN	-9223372036854775808	Defines the minimum value for a long int.
LONG_MAX	+9223372036854775807	Defines the maximum value for a long int.

The **float.h** header file of the c Standard Library contains a **set of various platform-dependent constants** related to floating point values. These constants are proposed by ANSI C. They allow making more portable programs.

Coding Example_1:

```
#include<stdio.h>
#include<limits.h>
#include<float.h>
int main()
```

```
{
        int a = 8;
        char p = 'c';
        double d = 89.7;
        int e = 0113; // octal literal
        /*printf("%d\n",e);
        printf("%o\n",e);
        printf("%X\n",e);
        printf("%x\n",e);
        */
        //int h = 0xRG; // error
        int h = 0xA;
        //printf("%d",h);
        int i = 0b111;
        //printf("%d",i);
        //printf("%d %d %d %d\n",sizeof(int),sizeof(short int),sizeof(p),sizeof(long));
        //int g = 787878787888888787;
        int g = 2147483649;
        /*printf("%d\n",INT_MAX);
        printf("%d\n",INT_MIN);
        printf("%d\n",INT_MIN);
        printf("%d\n",INT_MIN);*/
        printf("%d\n",CHAR_MAX);        printf("%g\n",DBL_MAX);
        printf("%g\n",FLT_MAX);         printf("%d\n",g);
        return 0;
}
```

Question for you to think!

- What is header file?
- What is Library file?
- Is there any difference between the above two?

Happy Coding using Identifiers in C!!