Department of Computer Science and Engineering,

PES University, Bangalore, India

Lecture Notes
Problem Solving With C
UE24CS151B

Lecture #6

Qualifiers

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: Qualifiers in C

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

Qualifiers in C

Introduction

Qualifiers are keywords which are applied to the data types resulting in Qualified type. Applied to basic data types to alter or modify its sign or size.

Types of Qualifiers are as follows.

- **Size Qualifiers**
- **Sign Qualifiers**
- **Type qualifiers**

Size Qualifiers

Qualifiers are prefixed with data types to **modify the size of a data type** allocated to a variable. Supports two size qualifiers, **short and long. T**he Size qualifier is generally used with an integer type. In addition, double type supports long qualifier.

Rules regarding size qualifier as per ANSI C standard:

short int <= int <=long int

float <= double <= long double

Note: short int may also be abbreviated as short and long int as long. But, there is no abbreviation for long double.

Coding Example_1:

```
#include<stdio.h>
int main()
{           short int i = 100000; // cannot be stored this info with 2 bytes. So warning
            int j = 100000;  // add more zeros and check when compiler results in warning
            long int k = 100000;
            printf("%d %d %ld\n",i,j,k);
            printf("%d %d %d",sizeof(i),sizeof(j),sizeof(k));
            return 0;
}
```

Sign Qualifiers

Sign Qualifiers are used to specify the signed nature of integer types. It specifies whether a variable can hold a negative value or not. It can be used with int and char types

There are two types of Sign Qualifiers in C: s**igned and unsigned**

A signed qualifier specifies a variable which can hold both positive and negative integers

An unsigned qualifier specifies a variable with only positive integers.

Note: In a t-bit signed representation of n, the most significant (leftmost) bit is reserved for the sign, "0" means positive, "1" means negative.

Coding Example_2:

```
#include<stdio.h>
int main()
{
        unsigned int a = 10;
        unsigned int b = -10;   // observe this
        int c = 10;  // change this to -10 and check
        signed int d = -10;
        printf("%u %u %d %d\n",a,b,c,d);
        printf("%d %d %d %d",a,b,c,d);
        return 0;
}
```

Type Qualifiers

A way of expressing additional information about a value through the type system and ensuring correctness in the use of the data.

Type Qualifiers consists of two keywords i.e., **const** and **volatile.**

const

The **const** keyword is like a normal keyword but the only difference is that once they are defined, their values can't be changed. They are also called as literals and their values are fixed.

Syntax: const data_type variable_name

Coding Example_3:

```
#include <stdio.h>
int main()
{       const int height = 100; /*int constant*/
        const float number = 3.14; /*Real constant*/
        const char letter = 'A'; /*char constant*/
        const char letter_sequence[10] = "ABC"; /*string constant*/
        const char backslash_char = '\?'; /*special char cnst*/
        //height++; //error
        printf("value of height :%d \n", height );
        printf("value of number : %f \n", number );
        printf("value of letter : %c \n", letter );
        printf("value of letter_sequence : %s \n", letter_sequence);
        printf("value of backslash_char : %c \n", backslash_char); return 0;
}
```

Output:

```
value of height : 100
value of number : 3.140000
value of letter : A
value of letter_sequence : ABC
value of backslash_char : ?
```

Note: In detail explanation of Pointers, constant Pointer and Pointer to constant will be discussed in Unit-2 wrt the Functions Topic.

volatile

It is intended to prevent the compiler from applying any optimizations. Their values can be changed by the code outside the scope of current code at any time. A type declared as volatile can't be optimized because its value can be easily changed by the code. The declaration of a variable as volatile tells the compiler that the variable can be modified at any time by another entity that is external to the implementation, for example: by the operating system or by hardware. **Syntax:** volatile data_type variable_name

Applicability of Qualifiers to Basic Types

The below table helps us to understand which Qualifier can be applied to which basic type of data.

No.	Data Type	Qualifier
1.	char	signed, unsigned
2.	int	short, long, signed, unsigned
3.	float	No qualifier
4.	double	long
5.	void	No qualifier

Happy Coding with Qualifiers!