

DATA TYPES & TOKENS

by
Dr. Sumit Srivastava
Dept. of Computer Science & Engineering

Unit -I

CS101 PPS @Sumit

1

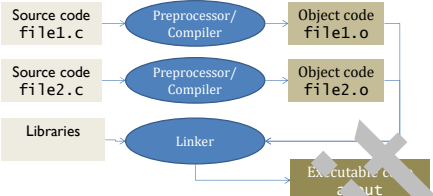
C: Characteristics

- C takes a middle path between low-level assembly language....
 - Direct access to memory layout through pointer manipulation
 - Concise syntax, small set of keywords
- ... and a high-level programming language like Java:
 - Block structure
 - Some encapsulation of code, via functions
 - Type checking (pretty weak)

2

Separate Compilation

- ▶ A C program consists of source code in one or more files
- ▶ Each source file is run through the **preprocessor** and **compiler**, resulting in a file containing object code
- ▶ Object files are tied together by the **linker** to form a single executable program



```

graph LR
    S1[Source code file1.c] --> P1[Preprocessor/Compiler]
    S2[Source code file2.c] --> P2[Preprocessor/Compiler]
    L[Libraries] --> P3[Linker]
    P1 --> O1[Object code file1.o]
    P2 --> O2[Object code file2.o]
    O1 --> P3
    O2 --> P3
    P3 --> E[Executable program]
  
```

3

Separate Compilation

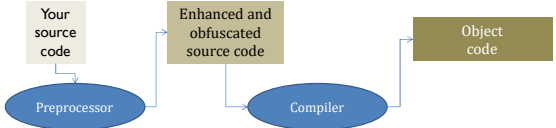
- ▶ Advantage: Quicker compilation
 - When modifying a program, a programmer typically edits only a few source code files at a time.
 - With separate compilation, only the files that have been edited since the last compilation need to be recompiled when re-building the program.
 - For very large programs, this can save a lot of time.

CS 3090: Safety Critical Programming in C

4

The Preprocessor

- ▶ The **preprocessor** takes your source code and – following certain **directives** that you give – tweaks it in various ways before compilation.
- ▶ A directive is given as a line of source code starting with the # symbol
- ▶ The preprocessor works in a very crude, “word-processor” way, simply cutting and pasting – it doesn’t really know anything about C!



```

graph LR
    Y[Your source code] --> P[Preprocessor]
    P --> E[Enhanced and obfuscated source code]
    E --> C[Compiler]
    C --> O[Object code]
  
```

5

Steps in executing a code

- **Write** the code (source code)
- **Compile** the source code
- If **errors**, then go back to edit and **correct** and **repeat compiling**
- If **no errors**, then compiler has **generated the machine code** (object code)
- **Run** the (object) code and so **get the results**

CS101 PPS @Sumit

6

C Program Phases

- **Editor** - code by programmer
- **Compiling** using gcc:
 - Preprocess – expand the programmer’s code
 - Compiler – create machine code for each file
 - Linker – links with libraries and all compiled objects to make executable
- **Running** the executable:
 - Loader – puts the program in memory to run it
 - CPU – runs the program instructions

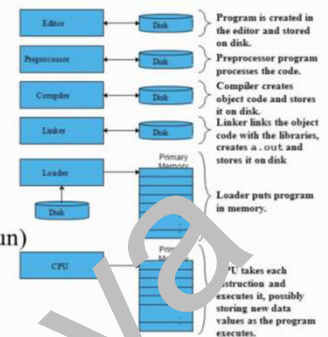
CS101 PPS @Sumit

7

Typical C Development Environment

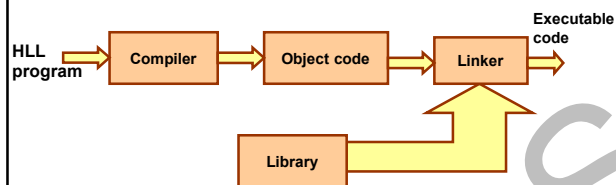
Phases of C Program:

1. Edit
2. Preprocess
3. Compile
4. Link
5. Load & Execute(Run)



8

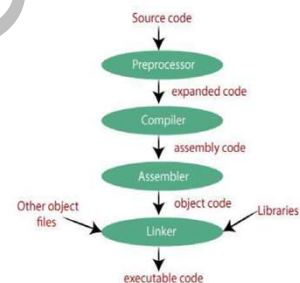
Programming Errors in C



CS101 PPS @Sumit

9

Compilation Process in C



CS101 PPS @Sumit

10

Compilation Process in C

Preprocessor

- The source code is the code which is written in a text editor and the source code file is given an extension ".c".
- This source code is first passed to the preprocessor, and then the preprocessor expands this code.
- After expanding the code, the expanded code is passed to the compiler.

CS101 PPS @Sumit

11

Compilation Process in C

Compiler

- The code which is expanded by the preprocessor is passed to the compiler. The compiler converts this code into assembly code.
- Or we can say that C compiler converts the pre-processed code into assembly code.

CS101 PPS @Sumit

12

Compilation Process in C

13

Assembler

- The assembly code is converted into object code by using an assembler. The name of the object file generated by the assembler is the same as the source file.
- The extension of the object file in DOS is '.obj,' and in UNIX, the extension is '.o'. If the name of the source file is 'hello.c', then the name of the object file would be 'hello.obj'

CS101 PPS @Sumit

13

Compilation Process in C

14

Linker

- linker links the object code of our program with the object code of the library files and other files.
- The output of the linker is the executable file. The name of the executable file is the same as the source file but differs only in their extensions.

CS101 PPS @Sumit

14

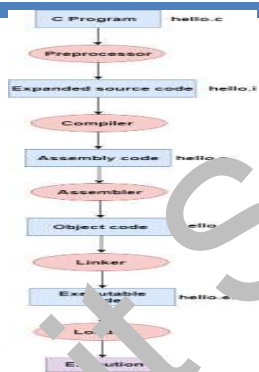
Compilation Process in C

15

Example: Hello.c

```
#include <stdio.h>

int main()
{
    printf("Hello BIT");
    return 0;
}
```



15

Structure of C Program

16

• A C program involves the following sections:

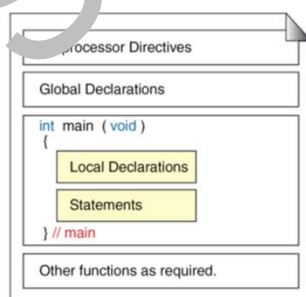
- Documentations (Documentation Section)
- Preprocessor Statements (Link Section)
- Global Declarations (Definition Section)
- The main() function
- Local Declarations
- Program Statements & Expressions
- User Defined Functions

CS101 PPS @Sumit

16

Programming in C

17

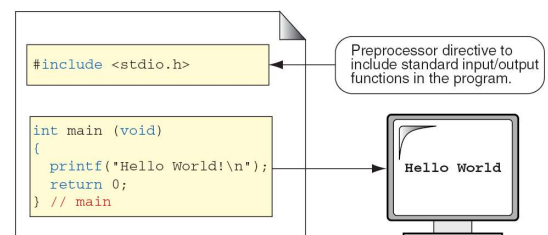


CS101 PPS @Sumit

17

Programming in C

18



CS101 PPS @Sumit

18

Programming in C

19

```

1  /* The greeting program. This program demonstrates
2     some of the components of a simple C program.
3     Written by: your name here
4     Date:      date program written
5  */
6  #include <stdio.h>
7
8  int main (void)
9  {
10 // Local Declarations
11
12 // Statements
13
14     printf("Hello World!\n");
15
16     return 0;
17 } // main

```

CS101 PPS @Sumit

19

First C Program

20

```

/*My first c program*/
#include<stdio.h>

Void main()
{
printf("Hello, World!\n");
}

```

CS101 PPS @Sumit

20

Structure of C Program

21

/* Comments */ - Comments are a way of explaining what makes a program. The compiler ignores comments and used by others to understand the code.

#include<stdio.h> - This is a preprocessor command. That notifies the compiler to include the header file `stdio.h` in the program before compiling the source-code.

Void - the function returns null

main() - The `main()` is the main function where program execution begins. Every C program must contain only one main function.

Braces - Two curly brackets "{...}" are used to group all statements.

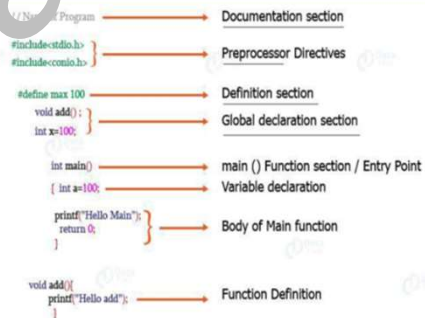
printf() - It is a function in C, which prints text on the screen.

CS101 PPS @Sumit

21

Structure of C Program

22



@Sumit

22

Input and Output Statements

23

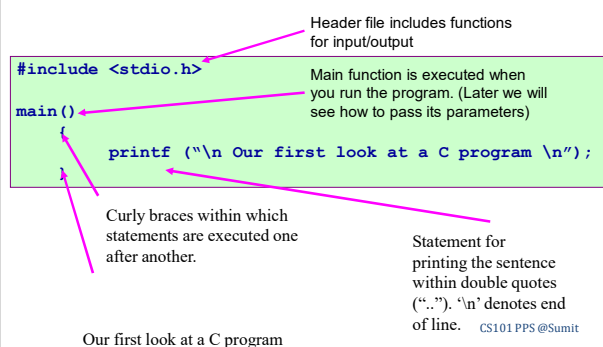
- printf()** is used to display the output and **scanf()** is used to read the input.
- `printf()` and `scanf()` functions are declared in "`stdio.h`" header file in C library.
- All syntax in C language including `printf()` and `scanf()` functions are case sensitive

CS101 PPS @Sumit

23

Sample C Program

24



CS101 PPS @Sumit

24

Sample C Program

25

```
#include <stdio.h>
main()
{
    int a, b, c;
    a = 10;
    b = 20;
    c = a + b;
    printf ("\n The sum of %d and %d is %d\n",
           a,b,c);
}
```

Integers variables declared before their usage.

Control character for printing value of a in decimal digits.

The sum of 10 and 20 is 30

CS101 PPS @Sumit

25

Sample C Program

26

```
#include <stdio.h>

/* FIND THE LARGEST OF THREE NUMBERS */

main()
{
    int a, b, c;
    scanf ("%d %d %d", &a, &b, &c);
    if ((a>b) && (a>c)) /* Composite condition check */
        printf ("\n Largest is %d", a);
    else
        if (b>c) /* Simple condition check */
            printf ("\n Largest is %d", b);
        else
            printf ("\n Largest is %d", c);
}
```

CS101 PPS @Sumit

26

Sample C Program

27

Preprocessor statement. Replace PI by 3.1415926 before compilation.

```
#include <stdio.h>
#define PI 3.1415926

/* Compute the area of a circle */
main()
{
    float radius, area;
    float myfunc (float radius);

    scanf ("%f", &radius);
    area = myfunc (radius);
    printf ("\n Area is %f\n", area);
}
```

Example of a function Called as per need from Main programme.

```
float myfunc (float r)
{
    float a;
    a = PI * r * r;
    return (a); /* return result */
}
```

Function call.

CS101 PPS @Sumit

27

28

SOME TERMINOLOGIES

CS101 PPS @Sumit

28

Some Terminologies

29

- Algorithm / Flowchart
 - A step-by-step procedure for solving a particular problem.
 - Should be independent of the programming language.
- Program
 - A translation of the algorithm/flowchart into a form that can be processed by a computer.
 - Typically written in a high-level language like C, C++, Java, etc.

CS101 PPS @Sumit

29

Variables and Constants

30

- Most important concept for problem solving using computers.
- All temporary results are stored in terms of variables and constants.
 - The value of a variable can be changed.
 - The value of a constant do not change.
- Where are they stored?
 - In main memory.

CS101 PPS @Sumit

30

Variables and Constants

31

- How does memory look like (logically)?
 - As a list of storage locations, each having a unique address.
 - Variables and constants are stored in these storage locations.
 - Variable is like a *house*, and the name of a variable is like the *address* of the house.
 - Different people may reside in the house, which is like the *contents* of a variable.

CS101 PPS@Sumit

31

Address and Values

Every memory location has a **unique** address

0	0
1	11
2	5
3	23
4	12
5	62

Address of byte Value of byte (0...255)

32

32

Memory Map

33

Address 0
Address 1
Address 2
Address 3
Address 4
Address 5
Address 6

Every variable is mapped to a particular memory address

Address N-1

CS101 PPS@Sumit

33

Variables in Memory

34

Instruction executed

Memory location allocated to a variable X

Time ↓

X = 10	→	10
X = 20	→	20
X = X + 1	→	21
X = X * 5	→	105

CS101 PPS@Sumit

34

Variables in Memory

35

Instruction executed

Variable

	X	Y
X = 20	20	?
Y = 15	20	15
X = Y + 3	18	15
Y = X / 6	18	3

Time ↓

CS101 PPS@Sumit

35

Variables in Memory

36

```
void main()
{
    int a = 4, b = 7, c;
    c = a + b;
    printf("sum is %d", c);
}
```

CS101 PPS@Sumit

36

Variables in Memory

37

```
void main()
{
    int a = 4, b = 7, c;
    c = a + b;
    printf("sum is %d", c);
}
```

CS101 PPS @Sumit

37

Variables in Memory

38

```
void main()
{
    int a = 4, b = 7, c;
    c = a + b;
    printf("sum is %d", c);
}
```

CS101 PPS @Sumit

38

Variables in Memory

39

```
void main()
{
    int a = 4, b = 7, c;
    c = a + b;
    printf("sum is %d", c);
}
```

CS101 PPS @Sumit

39

Variables in Memory

40

```
void main()
{
    int a = 4, b = 7, c;
    c = a + b;
    printf("%d + %d is %d", a, b, c);
}
```

CS101 PPS @Sumit

40

Memory Layout of C Programs

41

A typical memory representation of a C program consists of the following sections.

- 1. Text segment** (instructions) - contains executable instructions.
- 2. Initialized data segment** - contains the global variables and static variables that are initialized by the programmer.
- 3. Uninitialized data segment (bss)** - often called the "bss" segment "block started by symbol." contains all global variables and static variables that are initialized to zero or do not have explicit initialization in source code.
- 4. Heap** - Heap is the segment where dynamic memory allocation usually takes place. The heap area begins at the end of the BSS segment and grows to larger addresses from there. Eg malloc, calloc
- 5. Stack** - Stack segment is used to store all local variables and is used for passing arguments to the functions along with the return address of the instruction which is to be executed after the function call is over. Stack grows downwards.

CS101 PPS @Sumit

41

Memory Layout of C Programs

42

(c) goncha.blog.in - Tean

42

Memory Layout of C Programs

43

Example:

```
#include <stdio.h>
```

```
int main() {
    return 0;
}
```

```
~$ gcc file_1.c -o file_1
```

```
~$ size file_1
```

text	data	bss	dec	hex	filename
1418	544	8	1970	7b2	file_1

The size command is used to check the sizes (in bytes) of these different memory segments

CS101 PPS@Sumit

43

Naming Variables

- Variables in C may be given representations containing multiple characters. But there are rules for these representations.
- Variable names in C
 - May only consist of letters, digits, and underscores
 - May be as long as you like, but only the first 31 characters are significant
 - May not begin with a number
 - May not be a C reserved word (keyword)

44

Naming Conventions

- C programmers generally agree on the following **conventions** for naming variables.
 - Begin variable names with lowercase letters
 - Use meaningful identifiers
 - Separate "words" within identifiers with underscores or mixed upper and lower case.
 - Examples: surfaceArea surface_Area surface_area
 - Be consistent!

45

Global and Local Variables

46

Global Variables

These variables are declared outside of functions.

- ☞ Life time of a global variable is the entire execution period of the program.
- ☞ Can be accessed by any function defined below the declaration, in a file.

```
/* Compute Area and Perimeter of a circle */
#include <stdio.h>
float pi = 3.14159; /* Global */

main() {
    float rad; /* Local */

    printf("Enter the radius ");
    scanf("%f", &rad);

    if (rad > 0.0) {
        float area = pi * rad * rad;
        float peri = 2 * pi * rad;

        printf("Area = %f\n", area);
        printf("Peri = %f\n", peri);
    }
    else
        printf("Negative radius\n");

    printf("Area = %f\n", area);
}
```

CS101 PPS@Sumit

46

Global and Local Variables

47

Local Variables

- ☞ These variables are declared inside some functions.
- ☞ Life time of a local variable is the entire execution period of the function in which it is defined.
- ☞ Cannot be accessed by any other function.
- ☞ In general variables declared inside a block are accessible only in that block.

```
/* Compute Area and Perimeter of a circle */
#include <stdio.h>
float pi = 3.14159; /* Global */

main() {
    float rad; /* Local */

    printf("Enter the radius ");
    scanf("%f", &rad);

    if (rad > 0.0) {
        float area = pi * rad * rad;
        float peri = 2 * pi * rad;

        printf("Area = %f\n", area);
        printf("Peri = %f\n", peri);
    }
    else
        printf("Negative radius\n");

    printf("Area = %f\n", area);
}
```

CS101 PPS@Sumit

47

Data Types

48

- Data types are the **type of data** stored in a C program.
- It can be defined as a set of values with similar predefined characteristics. And All the values in a data type have the same properties.
- Data types are **used while defining a variable or functions** in C.
- A data type is an attribute that tells a computer how to interpret the value.
- It determines how much space it occupies in storage and how the bit pattern stored is interpreted.

CS101 PPS@Sumit

48

Data Types

49

- Data types are classified as follows...
 - ▣ **Primary data types** (Basic data types OR Predefined data types)
 - ▣ **Derived data types** (Secondary data types OR User-defined data types)
 - ▣ **Enumeration data types**
 - ▣ **Void data type**

CS101 PPS @Sumit

49

Data Types

50

- 1. **Basic Datatypes (Primary Datatypes)**
Integer, Floating Point
Double & Character
- 2. **Enumerated types**
Used to define variables that can only assign certain integer values
- 3. **void type**
The void type indicates that no value. That means an Empty type (nothing)
- 4. **Derived types**
User created data types like Array, structures, unions...

CS101 PPS @Sumit

50

Primary Data Types

51

- Primary data types are also known as the fundamental data types because they are predefined, or they already exist in the C language.
- The C programming language has four primitive or primary data types as :int, char, float, and double.

CS101 PPS @Sumit

51

Data Types

52

- **Integer** – We use these for storing various whole numbers, such as 5, 8, 67, 2390, etc.
- **Character** – It refers to all ASCII character sets as well as the single alphabets, such as ‘x’, ‘Y’, etc.
- **Double** – These include all large types of numeric values that do not come under either floating-point data type or integer data type.
- **Floating-point** – These refer to all the real number values or decimal points, such as 40.1, 820.673, 5.9, etc.

CS101 PPS @Sumit

52

Data Type Modifiers in C

53

- Modifiers are C keywords that modify the meaning of fundamental data types
- The modifiers help in making the primary or primitive data types much more specific.
- We use these along with all the basic data types for categorizing them further.
- C Programming Language has four data type modifiers as: long, short, signed, unsigned.

CS101 PPS @Sumit

53

Data Types

54

- Different data types can also hold integers in a variety of ranges. These values may differ from one compiler to the next.
- On the **32-bit architecture**, the ranges are listed below, along with the memory requirements and format specifiers.

CS101 PPS @Sumit

54

Data type	Size(bytes)	Range	Format String
char	1	-128 to 127	%c
unsigned char	1	0 to 255	%c
short	2	-32,768 to 32,767	%d
unsigned short	2	0 to 65535	%u
int	2	32,768 to 32,767	%d
unsigned int	2	0 to 65535	%u
long	4	-2147483648 to +2147483647	%d
Unsigned long	4	0 to 4294967295	%u
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

55

Format specifier

56

- The **Format specifier** is a string used in the formatted input and output functions.
- The **format string** determines the format of the input and output.
- The format string always starts with a '%' character.

EXAMPLE:

```
%d - int
%f - float
%c - char
```

CS101 PPS@Sumit

56

Derived Data Types in C

57

- Array Type:** They are a collection of data stored together as a single unit and indexed according to the associated value or index. Here, one or more elements can be arranged under a common name.
- Pointers:** They reference other variables by storing their address in memory. Note that multiple pointers can point to the same object, but they will always have separate addresses.
- Structures Types:** They are composed of many fields declared within the same block of memory that may contain different types or values. Structure types are accessed via dot notation (e.g., struct_name.field_name).
- Unions:** They allow users to combine two different sets of values into one variable, with only one set being active at any given time due to limited resources.
- Functions:** These allow grouping logic statements representing calculation or sequences, which can then be referred to through their name without having to re-write every time it needs to be used.

CS101 PPS@Sumit

57

Enumerated Data Types

58

- Enumerated data types in C define variables that can only take on predefined values.
- These values are stored as constants, and the variable must be assigned one or more of these constant values upon its declaration.
- This is useful when working with sequences such as days of the week, months in a year, etc.

CS101 PPS@Sumit

58

Void Data Types

59

- In C programming, the void data type is an empty data type with no value and cannot be directly assigned to a variable.
- It is commonly used in function declarations as a return type indicating that the function does not return any values and that it simply performs some task without producing any results.
- Void functions are sometimes referred to as "procedures." They may take parameters but do not have a defined set of output values.

CS101 PPS@Sumit

59

60 C TOKENS

CS101 PPS@Sumit

60

C Tokens

61

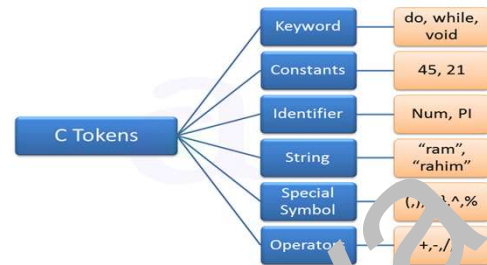
- Tokens in C language are the smallest elements or the building blocks used to construct a C program.
- A compiler breaks a program into the possible minor units known as tokens and proceeds further to the various stages of the compilation.
- Every meaningful character, word, or symbol in this C program is a C token. Compiler groups together these characters of the program into tokens.
- The compilation process:
C Program --> Group characters into C tokens --> Translate tokens into target code.

CS101 PPS @Sumit

61

C Tokens Types

62



CS101 PPS @Sumit

62

Keyword

63

- Keywords in C language are the collection of **pre-defined or reserved words**.
- These are **case-sensitive** and written in lower cases. Their meaning and functionality are already known to the compiler.
- Each Keyword is meant to perform a **specific function** in a C program.
- We can't use these keywords as variable names or function names.
- There are a total of **32 keywords** supported by the C language:

CS101 PPS @Sumit

63

Keyword

64

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

CS101 PPS @Sumit

64

Identifiers

65

- It help to **identify data and other objects** in the program.
- Identifier in C language is **used for naming functions**, variables, structures, unions, arrays, etc.
- The identifier is **user-defined words**. These identifiers can be composed of uppercase, lowercase letters, digits, underscore.
- Identifiers in C are short and informative names that **uniquely** identify variables or function names.

CS101 PPS @Sumit

65

Rules for declaring identifiers:

66

- Identifiers shouldn't begin with any numerical digit and hence, the first character must be either an underscore or an alphabet.
- Identifiers are case-sensitive and hence, both lowercase and uppercase letters are distinct.
- The length of identifiers shouldn't be more than 31 characters.
- Commas and blank spaces are not allowed while declaring an identifier.
- we can't use keywords as identifiers.

CS101 PPS @Sumit

66

Constant

67

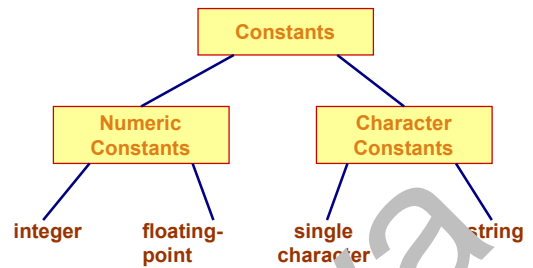
- Constants are the variables whose values are fixed and can not be changed during the execution of a program.
- They are also known as literals.
- We can declare constants in C language using:
 - const keyword** Here, we are using the const keyword to declare a variable and assigning a value to it that can not be modified later. `const variableName;`
 - #define pre-processor** Here, we are using #define pre-processor and constant ll will be an alias-name for long keyword.

CS101 PPS @Sumit

67

Types of Constants in C Language

68



CS101 PPS @Sumit

68

Types of Constants in C Language

69

Type of Constant	Example
Floating-point constant	25.7, 87.4, 13.9, etc.
Integer constant	20, 41, 94, etc.
Hexadecimal constant	0x5x, 0x3y, 0x8z, etc.
Octal constant	033, 099, 077, 011, etc.
String constant	"c++", ".net", "java", etc.
Character constant	'p', 'q', 'r', etc.

CS101 PPS @Sumit

69

Constant

70

- Constants can be classified into two categories:
 - Primary Constants
 - Secondary Constants
- Primary constant**
 - A primary constant is, again, divided into these three types:
 - Numeric
 - Character
 - Logical
 - Numeric is subdivided into two types, Integer and Float.
 - Character is subdivided into three types, Single Character, String and backslash

CS101 PPS @Sumit

70

Constants

- A constant is a value or an identifier whose value cannot be altered in a program.

Integer constants

- A integer constant is a numeric constant (associated with number) without any fractional or exponential part. There are three types of integer constants in C programming:
 - decimal constant(base 10)
 - octal constant(base 8)
 - hexadecimal constant(base 16)

71

Constants

Floating-point constants

- A floating point constant is a numeric constant that has either a fractional form or an exponent form. For example: 2.0, 0.0000234, -0.22E-5

Character constants

- A character constant is a constant which uses single quotation around characters. For example: 'a', 'l', 'm', 'F'

String constants

- String constants are the constants which are enclosed in a pair of double-quote marks. For example: "good", "x", "Earth is round"
- C supports some character constants having a backslash in front of it. The lists of backslash characters have a specific meaning known to the compiler. They are also termed "Escape Sequences".

72

Constants

Backslash constants

- C supports some character constants having a backslash in front of it. The lists of backslash characters have a specific meaning known to the compiler. They are also termed "Escape Sequences".

Logical Constant: Logical Constants in C consist of logical operators and can take either of the two values: true or false.

- They are generally of two types:
 - ▣ logical connectives
 - ▣ quantifiers.

73

Constant

Secondary Constant

The **secondary constant** is divided into the following types:

- Arrays
- Structures
- Union
- Pointer
- Enum etc.

CS101 PPS @Sumit

74

Special Characters in C

- Special characters as the name suggests, are symbols in C language that have special meaning and can not be used for any other purpose.

Types of Special Characters in C

- Square brackets []: Used for single and multi-dimensional arrays.
- Simple brackets (): Used for function declaration.
- Curly braces { }: Used for opening and closing the code.
- The comma (,): Used to separate variables.
- Hash/pre-processor (#): Used for the header file.
- Asterisk (*): Used for Pointers.
- Tilde (~): Used for destructing the memory.
- Period (.): Used for accessing union members.

CS101 PPS @Sumit

75

Operators in C

- The operators in C are the special symbols that we use for performing various functions.
- Operands are those data items on which we apply the operators.
- We apply the operators in between various operands.
- We can classify the operators On the basis of the total number of operands as:
 - Unary Operator
 - Binary Operator
 - Ternary Operator

CS101 PPS @Sumit

76

Operators in C

Unary Operator

- It is applied to one single operand.
- For example: increment operator (++), decrement operator (--), sizeof etc.

CS101 PPS @Sumit

77

Operators in C

Binary Operator

- It is applied between two operands.
- Here is a list of all the binary operators that we have in the C language:
 - Relational Operators
 - Arithmetic Operators
 - Logical Operators
 - Shift Operators
 - Conditional Operators
 - Bitwise Operators
 - Misc Operator
 - Assignment Operator

CS101 PPS @Sumit

78

Operators in C

79

Ternary Operator

- Using this operator would require a total of three operands. For instance, we can use the ?: in place of the if-else conditions.
- Conditional Operator (?) is known as ternary operator.
- Example:
 - `int a = 10, b = 20, c;`
 - `c = (a < b) ? a : b;`
 - `// If a < b is true, then c will be assigned with the value of a else b`
 - `printf("%d", c);`

CS101 PPS@Sumit

79

Strings in C

80

- The strings in C always get represented in the form of an array of characters.
- We have a '\0' null character at the end of any string- thus, this null character represents the end of that string.
- Double quotes enclose the strings, while the characters get enclosed typically within various single characters.
- The number of characters in a string decides the size of that string.

CS101 PPS@Sumit

80

Strings in C

81

- There are different ways in which we can describe a string:
 - `char x[9] = "chocolate";` // Here, the compiler allocates a total of 9 bytes to the 'x' array.
 - `char x[] = "chocolate";` // Here, the compiler performs allocation of memory during the run time.
 - `char x[9] = {'c','h','o','c','o','l','e','t','e','\0'};` // Here, we are representing the string in the form of the individual characters that it has.

CS101 PPS@Sumit

81

Tokens in C (Summary)

- Keywords**
 - These are reserved words of the C language. For example **int, float, if, else, for, while** etc.
- Identifiers**
 - An Identifier is a sequence of letters and digits, but must start with a letter. Underscore (_) is treated as a letter. Identifiers are case sensitive. Identifiers are used to name variables, functions etc.
 - Valid: **Root, _getchar, __sin, x1, x2, x3, x_1, If**
 - Invalid: **324, short, price\$, My Name**
- Constants**
 - Constants like 13, 'a', 1.3e-5 etc.

82 Lectures on Numerical Methods

82

Tokens in C (Summary)

- String Literals**
 - A sequence of characters enclosed in double quotes as "...". For example "13" is a string literal and not number 13. 'a' and "a" are different.
- Operators**
 - Arithmetic operators like +, -, *, /, % etc.
 - Logical operators like ||, &&, ! etc. and so on.
- Special Characters**
 - Spaces, new lines, tabs, comments (A sequence of characters enclosed in /* and */) etc. These are used to separate the adjacent identifiers, keywords and constants.

83 Lectures on Numerical Methods

83

THANK YOU

CS101 PPS@Sumit

84