**GANPAT UNIVERSITY**

**U.V.PATEL COLLEGE OF ENGINEERING & TECHNOLOGY**

**Department of Computer Science and Engineering**

**B.Tech 1ST Semester   Subject –ESFP-1**

**First Name:Punit       Enrollment Number:T14012111001                      Stream :MA**

**Assignment -1                                                                                  Date:27/8/2014**

Goal:To Learn about data types,c tokens,backslash characters

Requirements :c++

Q-1) Write a program to print ASCII values of all backslash characters and white space on screen in following format :

The ASCII value of ‘\t’ is : 9

(Hint : you can verify your ASCII values with the help of a table for ASCII values from text book)

**Ans-1):**

#include<stdio.h>

#include<conio.h>

void main()

{

char a;

clrscr();

a='\a';

printf("\nthe ascii value of'\\a' is:%d",a);

a='\b';

printf("\nthe ascii value of '\\b' is:%d",a);

a='\f';

printf("\nthe ascii value of '\\f' is:%d",a);

a='\n';

printf("\nthe ascii value of '\\n' is:%d",a);

a='\r';

printf("\nthe ascii value of '\\r' is:%d",a);

a='\t';

printf("\nthe ascii value of '\\t' is:%d",a);

a='\v';

printf("\nthe ascii value of '\\v' is:%d",a);

a='\?';

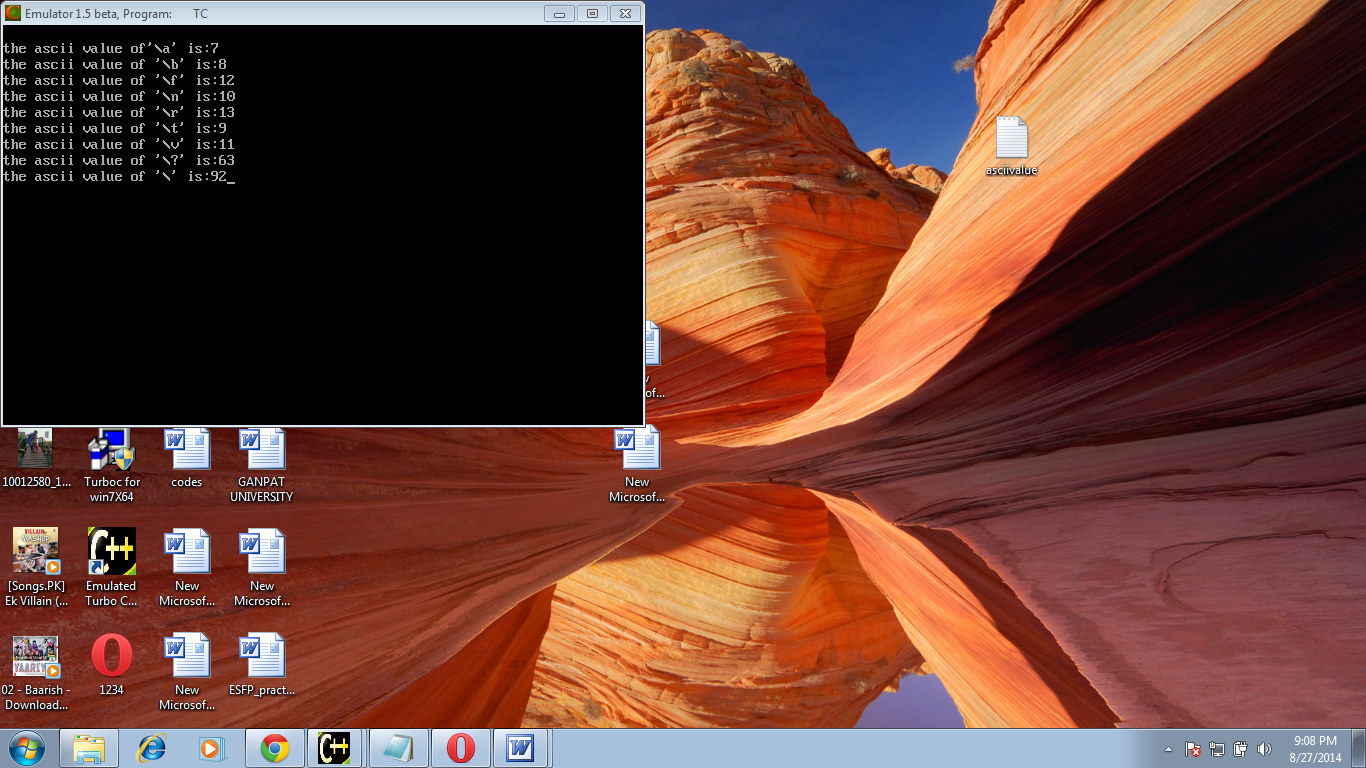
printf("\nthe ascii value of '\\?' is:%d",a);

a='\\';

printf("\nthe ascii value of '\\\' is:%d",a);

getch();

}

output: 

Q-2) Explain different data types available in C.

#### Ans-2):

#### C – data types:

There are four data types in C language. They are,

|  |  |  |
| --- | --- | --- |
| S.no | Types | Data Types |
| 1 | Basic data types | int, char, float, double |
| 2 | Enumeration data type | enum |
| 3 | Derived data type | pointer, array, structure, union |
| 4 | Void data type | void |

**1. Basic data types in C:**

**1.1. Integer data type:**

* Integer data type allows a variable to store numeric values.
* “int” keyword is used to refer integer data type.
* The storage size of int data type is 2 or 4 or 8 byte.
* It varies depend upon the processor in the CPU that we use.  If we are using 16 bit processor, 2 byte  (16 bit) of memory will be allocated for int data type.
* Like wise, 4 byte (32 bit) of memory for 32 bit processor and 8 byte (64 bit) of memory for 64 bit processor is allocated for int datatype.
* int (2 byte) can store values from -32,768 to +32,767
* int (4 byte) can store values from -2,147,483,648 to +2,147,483,647.
* If you want to use the integer value that crosses the above limit, you can go for “long int” and “long long int” for which the limits are very high.

**Note:**

* We can’t store decimal values using int data type.
* If we use int data type to store decimal values, decimal values will be truncated and we will get only whole number.
* In this case, float data type can be used to store decimal values in a variable.

**Example:**

{

int Count;

Count = 5;

}

**1.2. Character data type:**

* Character data type allows a variable to store only one character.
* Storage size of character data type is 1. We can store only one character using character data type.
* “char” keyword is used to refer character data type.
* For example, ‘A’ can be stored using char datatype. You can’t store more than one character using char data type.
* Please refer C-Strings topic to know how to store more than one characters in a variable.

**Example:**

{

char Letter;

Letter = 'x';

}

**1.3. Floating point data type:**

Floating point data type consists of 2 types. They are,

* 1. float
  2. double

**1. float:**

* Float data type allows a variable to store decimal values.
* Storage size of float data type is 4. This also varies depend upon the processor in the CPU as “int” data type.
* We can use up-to 6 digits after decimal using float data type.
* For example, 10.456789 can be stored in a variable using float data type.

**Example:**

{

float Miles;

Miles = 5.6;

}

**2. double:**

* Double data type is also same as float data type which allows up-to 10 digits after decimal.
* The range for double datatype is from 1E–37 to 1E+37.

**Example:**

{

double Atoms;

Atoms = 2500000;

}

**1.3.1. Modifiers in C:**

* The amount of memory space to be allocated for a variable is derived by modifiers.
* Modifiers are prefixed with basic data types to modify (either increase or decrease) the amount of storage space allocated to a variable.
* For example, storage space for int data type is 4 byte for 32 bit processor. We can increase the range by using long int which is 8 byte. We can decrease the range by using short int which is 2 byte.
* There are 5 modifiers available in C language. They are,
  + - 1. short
      2. long
      3. signed
      4. unsigned
      5. long long
* Below table gives the detail about the storage size of each C basic data type in 16 bit processor.  
  Please keep in mind that storage size and range for int and float datatype will vary depend on the CPU processor (8,16, 32 and 64 bit)

|  |  |  |  |
| --- | --- | --- | --- |
| S.No | C Data types | storage Size | Range |
| 1 | char | 1 | –127 to 127 |
| 2 | int | 2 | –32,767 to 32,767 |
| 3 | float | 4 | 1E–37 to 1E+37 with six digits of precision |
| 4 | double | 8 | 1E–37 to 1E+37 with ten digits of precision |
| 5 | long double | 10 | 1E–37 to 1E+37 with ten digits of precision |
| 6 | long int | 4 | –2,147,483,647 to 2,147,483,647 |
| 7 | short int | 2 | –32,767 to 32,767 |
| 8 | unsigned short int | 2 | 0 to 65,535 |
| 9 | signed short int | 2 | –32,767 to 32,767 |
| 10 | long long int | 8 | –(2power(63) –1) to 2(power)63 –1 |
| 11 | signed long int | 4 | –2,147,483,647 to 2,147,483,647 |
| 12 | unsigned long int | 4 | 0 to 4,294,967,295 |
| 13 | unsigned long long int | 8 | 2(power)64 –1 |

**2. Enumeration data type in C:**

* Enumeration data type consists of named integer constants as a list.
* It start with 0 (zero) by default and value is incremented by 1 for the sequential identifiers in the list.
* Enum syntax in C:

enum identifier [optional{ enumerator-list }];

* Enum example in C:

enum month { Jan, Feb, Mar }; or  
/\* Jan, Feb and Mar variables will be assigned to 0, 1 and 2 respectively by default \*/  
enum month { Jan = 1, Feb, Mar };  
/\* Feb and Mar variables will be assigned to 2 and 3 respectively by default \*/  
enum month { Jan = 20, Feb, Mar };  
/\* Jan is assigned to 20. Feb and Mar variables will be assigned to 21 and 22 respectively by default \*/

* The above enum functionality can also be implemented by “#define” preprocessor directive as given below. Above enum example is same as given below.

#define Jan 20;  
#define Feb 21;  
#define Mar 22;

**Example:**

#include <stdio.h>

int main()

{

enum MONTH { Jan = 0, Feb, Mar };

enum MONTH month = Mar;

if(month == 0)

printf("Value of Jan");

else if(month == 1)

printf("Month is Feb");

if(month == 2)

printf("Month is Mar");

}

***3. Derived data typ*e in C:**

* Array, pointer, structure and union are called derived data type in C language.
* To know more about derived data types, please visit “[C – Array](http://fresh2refresh.com/c/c-array/)“ , “[C – Pointer](http://fresh2refresh.com/c/c-pointer/)” , “[C – Structure](http://fresh2refresh.com/c/c-structures/)” and “[C – Union](http://fresh2refresh.com/c/c-union/)” topics in this tutorial.

**4. Void data type in C:**

* Void is an empty data type that has no value.
* This can be used in functions and pointers.
* Please visit “[C – Function](http://fresh2refresh.com/c/c-function/)” topic to know how to use void data type in function with simple call by value and call by reference example programs.

# 5. Qualifiers

A type qualifier is used to refine the declaration of a variable, a function, and parameters, by specifying whether:

* The value of a variable can be changed.
* The value of a variable must always be read from memory rather than from a register

Standard C language recognizes the following two qualifiers:

* const
* volatile

The *const* qualifier is used to tell C that the variable value can not change after initialisation.

const float pi=3.14159;

Now *pi* cannot be changed at a later time within the program.

Another way to define constants is with the *#define* preprocessor which has the advantage that it does not use any storage

The volatile qualifier declares a data type that can have its value changed in ways outside the control or detection of the compiler (such as a variable updated by the system clock or by another program). This prevents the compiler from optimizing code referring to the object by storing the object's value in a register and re-reading it from there, rather than from memory, where it may have changed. You will use this qualifier once you will become expert in "C". So for now just proceed.

# 6. What are Arrays:

We have seen all baisc data types. In C language it is possible to make arrays whose elements are basic types. Thus we can make an array of 10 integers with the declaration.

|  |
| --- |
| int x[10]; |

The square brackets mean subscripting; parentheses are used only for function references. Array indexes begin at zero, so the elements of x are:

Thus Array are special type of variables which can be used to store multiple values of same data type. Those values are stored and accessed using subscript or index.

Arrays occupy consecutive memory slots in the computer's memory.

|  |
| --- |
| x[0], x[1], x[2], ..., x[9] |

If an array has n elements, the largest subscript is n-1.

Multiple-dimension arrays are provided. The declaration and use look like:

|  |
| --- |
| int name[10] [20];  n = name[i+j] [1] + name[k] [2]; |

Subscripts can be arbitrary integer expressions. Multi-dimension arrays are stored by row so the rightmost subscript varies fastest. In above example**name** has 10 rows and 20 columns.

Same way, arrays can be defined for any data type. Text is usually kept as an array of characters. By convention in C, the last character in a character array should be a `\0' because most programs that manipulate character arrays expect it. For example, printf uses the `\0' to detect the end of a character array when printing it out with a `%s'.

Here is a program which reads a line, stores it in a buffer, and prints its length (excluding the newline at the end).

|  |
| --- |
| main( ) {  int n, c;  char line[100];  n = 0;  while( (c=getchar( )) != '\n' ) {  if( n < 100 )  line[n] = c;  n++;  }  printf("length = %d\n", n);  } |

## Array Initialization

* As with other declarations, array declarations can include an optional initialization
* Scalar variables are initialized with a single value
* Arrays are initialized with a list of values
* The list is enclosed in curly braces

|  |
| --- |
| int array [8] = {2, 4, 6, 8, 10, 12, 14, 16}; |

The number of initializers cannot be more than the number of elements in the array but it can be less in which case, the remaining elements are initialized to 0.if you like, the array size can be inferred from the number of initializers by leaving the square brackets empty so these are identical declarations:

|  |
| --- |
| int array1 [8] = {2, 4, 6, 8, 10, 12, 14, 16};  int array2 [] = {2, 4, 6, 8, 10, 12, 14, 16}; |

An array of characters ie string can be initialized as follows:

|  |
| --- |
| char string[10] = "Hello"; |

Q-3) Explain about C Tokens in brief.

**Ans-3):**

# C Tokens

Tokens are individual words and punctuation marks in passage of text. In C, program the smallest individual units are known as C Tokens. C has **Six types of Tokens. The Tokens are shown in figure.**

## Key words & Character set

**Instructions in C language are formed using syntax and keywords. It is necessary to strictly follow C language Syntax rules. Any instructions that mismatches with C language Syntax generates an error while compiling the program. Keywords should not be used either as Variable or Constant names. The character set in C Language can be grouped into the following categories.**

**1. Letters**

**2. Digits**

**3. Special Characters**

**4. White Spaces**

**White Spaces are ignored by the compiler until they are a part of string constant.**

**Auto Extern Sizeof**

**Break Float Static**

**Case For Struct**

**Char Goto Switch**

**Const If Typedef**

**Continue Int Union**

**Default Long Unsigned**

**Do Register Void**

**Double Return Volatile**

**Else Short while**

**Enum Signed**

Q-4)Explain about backslash characters in C.

**Ans:**

String values in the C language are terminated with null characters (CHAR(0)) and can contain nonprintable characters (such as backspace).

Nonprintable characters are specified by escape sequences. An escape sequence is denoted by using the backslash (\) as an escape character, followed by a single character indicating the nonprintable character desired.

This type of string is specified by using a standard string constant followed by the character C. The standard string constant is then interpreted as a C-language constant. Backslashes are treated as escapes, and a null character is automatically appended to the end of the string (even if the string already ends in a null character).

[Table 3-1](http://h21007.www2.hp.com/portal/download/files/unprot/fortran/docs/lrm/lrm0035.htm#esc_seq_tab) shows the escape sequences that are allowed in character constants.

**Table 3-1****C-Style Escape Sequences**

|  |  |
| --- | --- |
| **Escape Sequence** | **Represents** |
| \a or \A | A bell |
| \b or \B | A backspace |
| \f or \F | A formfeed |
| \n or \N | A new line |
| \r or \R | A carriage return |
| \t or \T | A horizontal tab |
| \v or \V | A vertical tab |
| \x*hh* or \X*hh* | A hexadecimal bit pattern |
| \*ooo* | An octal bit pattern |
| \0 | A null character |
| \\ | A backslash (\) |

If a string contains an escape sequence that isn't in this table, the backslash is ignored.

A C string must also be a valid Fortran string. If the string is delimited by apostrophes, apostrophes in the string itself must be represented by two consecutive apostrophes ('' ).

For example, the escape sequence \'string causes a compiler error because Fortran interprets the apostrophe as the end of the string. The correct form is\''string .

If the string is delimited by quotation marks, quotation marks in the string itself must be represented by two consecutive quotation marks ("").

The sequences \ooo and \x*hh* allow any ASCII character to be given as a one- to three-digit octal or a one- to two-digit hexadecimal character code. Each octal digit must be in the range 0 to 7, and each hexadecimal digit must be in the range 0 to F. For example, the C strings '\010'C and '\x08'C both represent a backspace character followed by a null character.

The C string '\\abcd'C is equivalent to the string '\abcd' with a null character appended. The string ''C represents the ASCII null character.