Programming in C

Nepal College of Information Technology

Course Instructor : Er. Rabina Chaudhary

Character set in C programming language:

- The characters that can be used to form words
- It can be single alphabet or a single digit or single special symbol
- C character are
  - Alphabet a-z, A-Z
  - Digits 0-9
  - Symbols *,(),{},+,;,etc
  - White space : blank space, new line

Constant:

- Constant is an entity whose value remains fixed, constant during program execution
- Constant are stored in particular memory location and that memory location is called variable
- X=100
- Here X is a variable
- 100 is a constant

Types of Constant:

1. Primary constant
    a. integer constant
    b. real constant
    c. character constant
2. Secondary constant
    a. array
    b. pointer
    c. structure
    d. union
    e. enum

Integer Constant:

- It is number without any fractional part or decimal form
- Example: 10,20,30
- Rules for constructing integer constant
  - It must have at least one digit
  - It must not contain decimal or fractional form
  - It can be both negative and positive
  - If no sign is specified, it is positive by default
  - No commas or space is allowed within integer constant

Real Constant:

- It is a number with decimal or fractional form or floating point form
- Example:

  3.14, -6.89007
- Two types of real constant
  a. Fractional form
  b. Exponential form

Real Constant:

a. **Fractional form:**

Rules for constructing fractional form

1. Real constant must have a decimal form
2. Real constant must have at least one digit
3. It can be both negative and positive
4. If no sign is specified, it is positive by default
5. No commas or space is allowed within real constant
6. Example: 3.1467, 31.56001, -1213.3012

Real Constant:

b**. Exponential form:**

Rules for constructing exponential form

- If the value of real constant is too small or too large, exponential form of representation of real constant is used
- Exponential form consists of two parts
  - Mantissa
  - Exponent
- General form is mantissa e exponent
  - Example:
    - 0.000345 can be represented as 3.45 e -4 in exponential form

b**. Exponential form:**

- the mantissa and exponent part must be separated by ‘e’
- Mantissa part may be positive or negative
- Default sign of mantissa part is positive
- Exponent part must have at least one digit
- Exponent part may be positive or negative
- Default sign of Exponent part is positive

Character constant:

- It can be single alphabet, single digit or a single special symbols enclosed in a single quotation ‘ ‘
- Rules

1. Character constant can hold single character at a time which must be enclosed in single inverted comma

   Example:

   ‘a’ , ‘1’, ‘@’

Keywords:

- Reserved words used in programming are called keywords
- Their meaning is already predefined and reserved
- There are 32 keywords in C programming language

example:

char
int
break
case
goto
switch
if
else
void
continue

etc

Identifiers in C:

- Identifiers are the names given to variables, constants, functions, structures, labels in our program

Rules for naming identifiers

1. Keyword cannot be used as identifiers
2. Identifiers can have alphabets (both lowercase and uppercase),digits and underscore
3. The first letter of identifier can be an alphabet or an underscore
4. It must not be longer than 32 characters
5. no commas, blank space or any other symbols are allowed in an identifier

example: int number;

float testFunction( );

here number and testFunction is identifier

Tokens in C:

- C tokens are the basic building blocks of C programming language
- Tokens are smallest elements of a program

C tokens are of six types:

1. keywords (int, if, switch, case)
2. identifiers (numbers, testFunction)
3. constants (100,200,'a')
4. strings ("Hello world");
5. special symbols ({}, ())
6. operators (+,-,&)

Variables:

- Variables is the name given to memory location in which constant is stored
- Value of variable can change during the execution of program

Rules for naming variables:

1. Variable name must start with an alphabet or underscore

   Example:

   num -> valid

   _num -> valid

   1num -> invalid

Variables:

2. Variable name must not include commas, blank space or any special symbols except underscore

example:

num 1-> invalid

num,1 -> invalid

num_1-> valid

num@1 ->invalid

Variable:

3. Variable name cannot be any keyword

int ->invalid

4. Length of variable name must be up to 32 characters

5. Variable name is case sensitive

i.e number and Number are different

6. Variable must be declared before using it in a program

e.g. The variable number1 and number2 must be declared before taking input for number1 and number2 and calculation of sum

Basic Structure of C program

Documentation section
Link section
Definition section
Global declaration section

main function section
{
declaration part
executable part
}

Subprogram section user defined function1 user defined function2

Data types in C:

- In C programming, data types are used for declarations for variables
- This determines the type and size of data associated with variables

Basic data types in C are

1. int
2. float
3. char
4. double

Data types in C :

Derived data types :

- Data types that are derived from fundamental data types are derived types.

  For example: arrays, pointers, function, structures, etc.

1. int data type

- Integer are numbers without decimal or fractional form
- Integer is a number that can be zero, positive or negative number but cannot have decimal values

  Example: 1024, 425

- int is used to declare integer variable

  int number

Here number is a variable of integer type

int :

- There are three types of integer

1. Integer (int)
2. Short integer (short int)
3. Long integer (long int)

It can be both in signed and unsigned forms

Signed and unsigned integers:

Signed Integers

- It can represent both positive and negative integers
- The data type qualifier is

  signed int or **int**

Vairables are defined as :

int number;

signed int number;

- it reserves 2 bytes or 4 bytes of memory depends on the processor in CPU

Unsigned Integers

- It represents only positive integers
- The data type qualifier is **unsigned int**

  example: unsigned int number;
- it reserves 2 bytes or 4 bytes of memory

Signed Integers

- Format specifiers (conversion character) is %d
- Range is -2^{15} to $2^{15}-1$

  i.e -32,768 to 32,767 if we use 16 bit processor
- Range is -2^{31} to $2^{31}-1$

  i.e -2,147,483,648 to 2,147,483,647 if we use 32 bit processor

Unsigned integers

- Format specifier (conversion character) is %u
- Range is 0 to $2^{16} – 1$ i.e 0 to 65,535 for 16 bit processor
- Range is 0 to $2^{32}-1$ i.e 0 to 4,294,967,295 for 32 bit processor

Signed and Unsigned short Integers:

Signed short integers

- It can represent both positive and negative integers
- Data type qualifier is

  signed short int or **short int**

  or short

Variable are defined as:

signed short int number;

Unsigned short Integers

- It can represent positive and integers
- Data type qualifier is

  unsigned short int or

  unsigned short

Variable are defined as:

unsigned short int number;

Signed short integers

- It reserves 2 bytes of memory
- Range is -2^{15} to $2^{15}-1$

  i.e -32,768 to 32,767
- Format specifier (conversion character) is %i or %d

Unsigned short integers

- It reserves 2 bytes of memory
- Range is 0 to $2^{16} – 1$ i.e 0 to 65,535
- Format specifier (conversion character) is %u

Signed and unsigned long integers:

Signed long Integers

- It can represent both positive and negative integers
- The data type qualifier is

signed long int or **long int or long**

Vairables are defined as :

long int number;

signed long int number;

- it reserves 4 bytes of memory

Unsigned long Integers

- It represents only positive integers
- The data type qualifier is **unsigned long int or unsigned long**

example: unsigned long int number;

- it reserves 4 bytes

Signed long Integers

- Format specifiers (conversion character) is %ld
- Range is -2^{31} to $2^{31}-1$

  i.e -2,147,483,648 to 2,147,483,647

Unsigned long integers

- Format specifier (conversion character) is %lu
- Range is 0 to $2^{32}-1$ i.e 0 to 4,294,967,295

2. float data type

- Floating point numbers are real numbers in decimal form
- They are defined using **float** keyword
- It reserves 4 bytes in memory
- It gives six digits of precision
- Format specifier is %f
- Range is $-3.4*10^{-38}$ to $3.4*10^{38}$
- Variables are defined as

  float number;

3. Double data type :

- A double type number use 8 bytes and is used when accuracy of float is not sufficient
- Data type qualifier is keyword **double**
- Reserves 8 bytes of memory
- Range is $1.7*10^{-308}$ to $1.7*10^{308}$
- Its format specifier is %lf
- Variables are defined as :

  double number;

long double data type:

- Is the accuracy of double is not sufficient then long double is used
- Reserves 10 bytes of memory
- Range is $3.4*10^{-4932}$ to $1.1*10^{4932}$
- Format specifier is %Lf
- Variable is declared as: **long double number;**

4. char data type:

- Single character can be defined as character data
- Data type qualifier is keyword **char**
- It occupies 1 byte in memory
- Format specifier is %c
- Range is 0 to 255
- Character value is written in single quote as:

```
char a='Z';
```

Assignment:

Write a program to explain each data type by declaration of variable of each type, initialization of each variable at compile time and run time, display space allocated by each variable.

ASCII values:

- American Standard Code for Information Interchange (ASCII) is equivalent numerical value of each character
- ASCII equivalent of ‘a’ is 97, ‘b’ is 98 and so on for ‘z’ is 122
- ASCII equivalent of ‘A’ is 65, ‘B’ is 66 and so on for ‘Z’ is 90
- ASCII equivalent of ‘0’ is 48, ‘1’ is 49 and so on for ‘9’ is 57

ASCII values:

- A character is represented by an ASCII number internally
- When a character is displayed using format specifier %d, it will display the ASCII value
- Example:

```
#include < stdio.h >
#include < conio.h >
 void main()
        {
                char a;
                printf (“Enter a character :”);
                scanf (“%c”,&a);
                printf(“\nThe equivalent ASCII value of %c is %d”,a,a);
                getch();
        }
```