Elemen Penyusun Program

Agenda

- This session will deals with basic elements, which are used to create a program, in specific a C program.
- These elements are - the valid character set, identifiers, keywords, basic data types and their representation, constants and variables.

Character Set

- C uses the uppercase English alphabets A to Z, the lowercase letters a to z, the digits 0 to 9, and certain special characters as building blocks to form basic program elements such as constants, variables, operators, expressions and statements.

Special Character Set

!	*	+	\	"	<
#	(	=	|	{	>
%	)	~	;	}	/
^	-	[	:	,	?
&	-	]	'	.	(blank)

Escape sequence:

- “\n” : newline
- “\t” : tab
- “\b” : backspace

Identifiers and Keywords

- Identifiers are names given to various items in the program, such as variables, functions and arrays.
- An identifier consists of letters and digits, in any order, except that the first character must be a letter.
- Both upper and lowercase letters are permitted. Upper and lowercase letters are however not interchangeable (i.e., an uppercase letter is not equivalent to the corresponding lowercase letter).
- The underscore character (_) can also be included, and it is treated as a letter.
- Keywords like if, else, int, float, etc., have special meaning and they cannot be used as identifier names.

example

The following are examples of valid identifier names: **A, ab123, velocity, stud_name, circumference, Average, TOTAL**

The following names are not valid identifiers:

```
1st          - the first character must be a letter
"Jamshedpur" - illegal characters (")
stud-name    - illegal character (-)
stud name    - illegal character (blank space)
```

Identifier Length

- Although an identifier can be arbitrarily long, most implementations recognize typically 31 characters.
- There are some implementations, which recognize only eight characters. The ANSI standard recognizes 31 characters.
- If a system recognizes only 8 characters and if you use a variable name with more than 8 characters, only the first 8 characters will be taken, the rest will be ignored. Thus, average_of_numbers and average_ will both be recognized as average_.

exercise

Identify which of the following are valid identifiers.

```
(a)  stud 1   (d)  switch   (g) Average of Numbers      (j) 123-45-678
(b)  1stud    (e)  %calc    (h) Average_of_Numbers
(c)  stud_1   (f)  _x       (i) Average-of-Numbers
```

Data Types and Sizes

Data type	Description	Size	Range
char	single character	1 byte	0 - 255
int	integer number	4 bytes	-2147483648 to 2147483647
float	single precision floating point number (number containing fraction & or an exponent)	4 bytes	3.4E-38 to 3.4E+38
double	double precision floating point number	8 bytes	1.7E-308 to 1.7E+308

Data Types and Sizes

- The list of data types can be increased by using the data type qualifiers such as - short, long, and unsigned.
- The memory requirement of an integer data varies depending on the compilers used.

Data type	Size	Range
short int	2 bytes	-32768 to 32767
long int	4 bytes	-2147483648 to 2147483647
unsigned short int	2 bytes	0 to 65535
unsigned int	4 bytes	0 to 4294967295
unsigned long int	4 bytes	0 to 4294967295
long double (extended precision)	8 bytes	1.7E-308 to1.7E+308

Char type

- The char type is used to represent individual characters, and occupies one byte of memory.
- Each character has an equivalent integer representation (since each stores internally the ASCII value for the corresponding character).

  e.g. 'A' corresponds to the integer 65. The ASCII value of 'a' (small) is 97.
- So char variables or constants can be used as integer data in arithmetic expressions.

Program Expressions

- The data objects to be manipulated in a C program are classified as *variables* and *constants*.
- The type of all the variables to be used in the program must be declared before they can be used.
- The operations that can be performed on the data objects are specified by a set of operators.
- Expressions used in a program combine the variables, constants and operators to produce new values.

Constants

- The constants in C can be classified into four categories
  - integer constants,
  - floating point constants,
  - character constants and
  - string constants.

Character and String Constants

- A character constant is written as for example - 'A' (always enclosed in single quotes).
- Examples of string constants are - "Jamshedpur", "A", etc. Note that a string constant is always enclosed within double quotes.

Non-printing Characters

- Some character constants are of non-printing type which can be expressed by a combination of a back-slash (\) and a character.
- They are known as escape sequences.
- Each of them represents a single character even though they are written in terms of two or more characters.

Character	Escape Sequence	ASCII Value
Bell	\a	007
Backspace	\b	008
Null	\0	000
Newline	\n	010
Carriage return	\r	013
Vertical tab	\v	011
Horizontal tab	\t	009
Form feed	\f	012

Integer Constants

- A normal integer constant is written as 1234.
- A long int is recognized by the presence of L (uppercase or lowercase) at the end of the constant, e.g. 2748723L.
- The suffix u or U signifies the int to be an unsigned one.
- The UL or ul at the end indicates the int quantity is of unsigned long type

Floating Point Constants

- Floating point constants contain a decimal point (167.903) or an exponent (1e-2) or both.
- Their type is double unless suffixed.
- The suffix of f or F indicates float; 1 or L indicates long double.

Numeric Constants

- In numeric constants e.g. integer or floating point constants, blanks and any non-numeric characters cannot be included.
- The range of these constants will be limited by the maximum and minimum bounds (usually machine dependent).

Octal and Hexadecimal Data

- C also supports octal and hexadecimal data.
- The value of an integer data can be specified in either octal or hexadecimal form.
- A hexadecimal constant must begin with 0x or 0X.
- A leading 0 indicates the octal representation.
- Octal and hexadecimal constants may also be followed by U to indicate unsigned or L to determine long.

Hexadecimal Example

The number 0x2A5 is an example of a hexadecimal number. Internally the number is represented by the following bit patterns,

$$0x2A5 = \underset{\begin{array}{cccc}2 & A & 5\end{array}}{\begin{array}{cccc}0010 & 1010 & 0101\end{array}} = 2 * 16^2 + 10 * 16^1 + 5 * 16^0 = 677$$

The number 677 is the decimal equivalent of the number 0x2A5.

Octal Example

Example of an octal number can be 0347. To represent each digit of an octal number in the form of binary, we need maximum of three bits since the last digit in the octal number system is 7.

$$0347 = \underset{\substack{\text{---}\\ 3}}{011}\ \underset{\substack{\text{---}\\ 4}}{100}\ \underset{\substack{\text{---}\\ 7}}{111} = 3 * 8^2 + 4 * 8^1 + 7 * 8^0 = 231 \text{ (in decimal)}$$

exercise

Identify which of the following numerical values are valid constants. If a constant is valid, specify whether it is integer or real. Also, specify the base for each valid integer constant.

(a)	0.7	b)	39,634	c)	16.3e18	d)	16.3e-18
(e)	123456789	f)	123456789l	g)	0.3E+0.4	h)	0.3E4
(i)	0412	j)	018ACF	k)	0xABCDE	l)	0x97e334

exercise

Which of the following are valid character constants?

```
(a)    'a'      (b)    '/n'          (c)    'F'      (d)    '\0492'
(e)    'x'      (f)    '\\'          (g)    '\0'     (h)    '\n'
(i)    '\b'     (j)    '\x-y-'
```

exercise

Which of the following are valid string constants?

```
(a) '9:25 A.M.'
(b) "Blue, Green and Indigo"
(c) "Name:
(d) "Section 4 (Next \'d')"
(e) "1.6e-18"
(f) "JAMSHEDPUR BR 831 001"
(g) "The Station master announced, "Down Geetanjali express is running
    late"
```

Symbolic Constants

- Constants which play crucial roles in a program can be made more meaningful by assigning them appropriate names to make the program more readable and easily changeable.
- The salient feature of a symbolic constant is that the value assigned to a symbolic constant cannot be changed (unlike variables) subsequently in the program.
- These constants are called symbolic constants and are defined as follows.

  Examples:

  ```
  #define PI 3.141593
  #define TRUE 1
  #define PROMPT "Enter Your Name :"
  ```

Variable Declaration

- In a C program all variables must be declared before they are used.
- A declaration determines the type of the data, and contains a list of one or more variables having the same type.

Example

```
int           count, index;
char          flag, text[80];
short int     a,b;
unsigned int  p;
double        d;
```

Variable Declaration

- A variable can be initialized with values at the time of declaration.

  Example

```
int    c = 5;
char   reply = 'Y';
double d = 4.64386237445675;
char   state[] = "ANDHRA PRADESH";
float  eps = 1.0e-5;
```

exercises

Write appropriate declaration for each group of variables and character array (strings):

```
(a) Integer variables: x, y
(b) Integer variable: count
    Unsigned integer variable: employee no
    Double-precision variables: net sales, tax, profit
(c) Character variables: first name, last name
(d) 70 element character array: message
```

exercises

Write appropriate declaration and assign the given initial values to each group of variables and array:

```
(a) Floating-point variables: x = -9.5, y = 1.0004
    Integer variables: a = 734, b = 49, c = -63
    Character variables: c1 = 'a', c2 = '$'
(b) Double-precision variable: d1 = 3.94 * 10-12, d2 = -9.89 * 107
    Integer variable: i = 437 (octal), h = 6AFF (hexadecimal)
(c) Long integer variable: large = 123456789
    Double-precision variables: d = 0.6666666
    Character variable: eol = newline character
(d) One-dimensional character array: message = "OVERFLOW"
```

Program Structure in C

main() Function

- A program in C consists of one or more functions, and one of them must be called **main()**, which is the controlling function.
- Program execution always begins by executing the main function.
- Additional function definitions may precede or follow main.

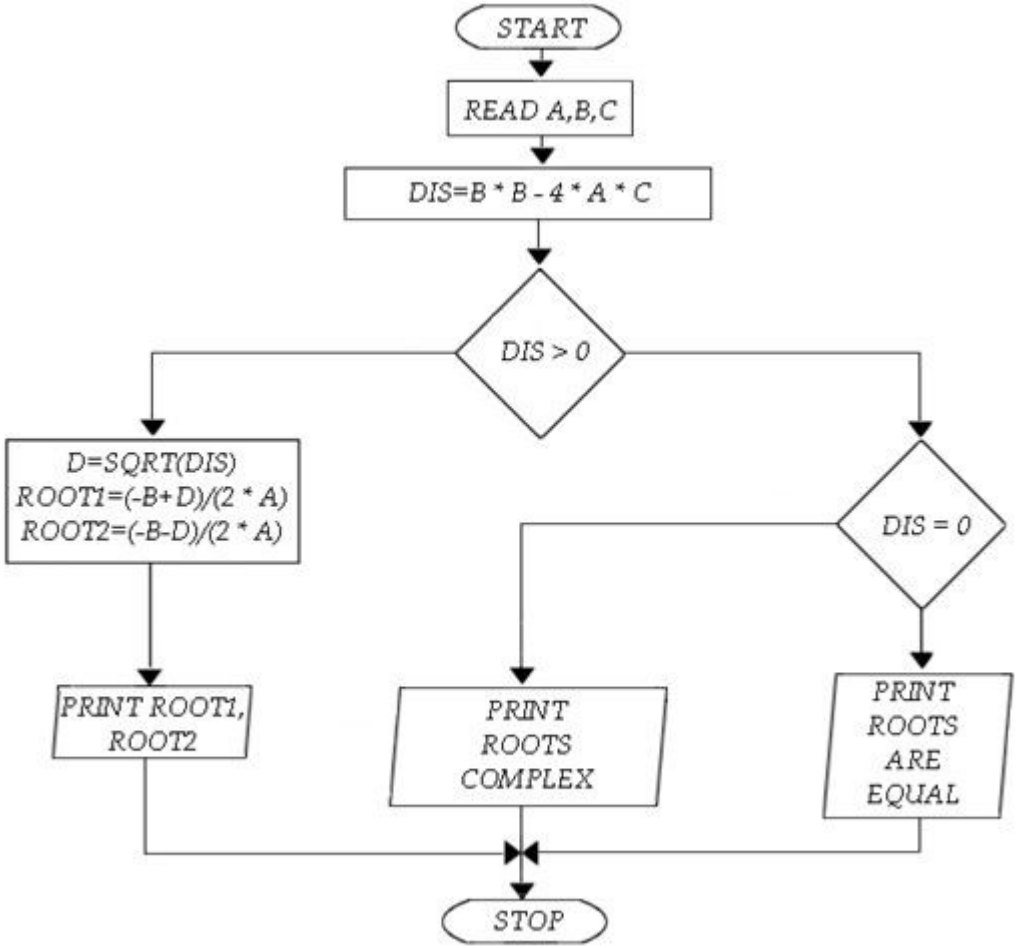
Function

- A function definition consists of the following components:
  - A function heading, which consists of the return type, the function name, followed by an optional list of arguments and their declarations, enclosed in parentheses.
  - A compound statement, which comprises the body of the function containing the function code.

The structure

- The arguments (or parameters) are symbols that represent information being passed between the function and other parts of the program.
- Each compound statement is enclosed within a pair of braces, i.e., { }.
- The braces may contain combinations of elementary statements (called expression statements) and other compound statements. Thus, compound statements may be nested, one within another.
- Each expression statement must end with a semicolon (;).
- Comments (remarks) may appear anywhere within a program, as long as they are placed within the delimiters /* and */ (e.g.,/* this is a comment */). Such comments are useful for identifying the principal features of the program and also for explaining the underlying logic.

example


START
READ A,B,C
DIS=B * B - 4 * A * C
DIS > 0
D=SQRT(DIS)
ROOT1=(-B+D)/(2 * A)
ROOT2=(-B-D)/(2 * A)
DIS = 0
PRINT ROOT1, ROOT2
PRINT ROOTS COMPLEX
PRINT ROOTS ARE EQUAL
STOP

example

```
#include <stdio.h> /* a preprocessor directive to include the file stdio.h */
 #include <math.h>
 /* program to find the real roots of a quadratic polynomial */
 main()
 {
        /* type declaration of the variable */
        float a, b, c, discriminant, d, w, root1, root2;  /* function declaration */
        float compute_discriminant(float x, float y, float z);
        printf("Give the values of the coefficients a,b,c of the polynomial:");
        scanf("%f,%f,%f", &a, &b, &c);
        fflush(stdin); /* flushing the buffer associated with standard input */
        discriminant = compute_discriminant(a,b,c);  /* invoking a function */
        d = 2 * a;
        if(discriminant == 0)
                printf("The roots are equal %f\n", - b / d);
        else
                if(discriminant < 0)
                        printf("The roots are complex\n");
                else
                {
                        w = sqrt(discriminant);
                        printf("The roots are real and different\n");
                        root1 = (-b - w) / d;
                        root2 = (-b + w) / d;
                        printf("Root1 = %f, Root2 = %f\n",root1,root2);
                }
    return;
 }
 /* Function definition */
 float compute_discriminant (float x, float y, float z)
 {
        float d;                  /* local variable declaration */
        d = y * y - 4 * x * z;
        return d;
 }
```

End

Questions?