Introduction to Computer Programming

Recompiled -
Dr. Jaideepsinh Raulji
Assistant Professor,
CSE, SET,
Navrachana University

Algorithms

- **Algorithm** is a procedure or step by step process for solving a problem.

- Any problem can be **expressed** using different kinds of notations, including algorithm, pseudocode, flowcharts, programming languages and natural languages.

- Computer algorithm is a kind of logic written in computer software by the programmer to solve a specific problem.

C-Programming

- Dennis Ritchie – AT&T Bell Laboratories – 1972
- Widely used today
    - Embedded Systems, OS, IoT, Kernel Programming, etc.
- Extends to newer system architectures
- Efficiency / performance
- Low-level access to the system components

Systems programming:

- OSes, like Linux
- microcontrollers: automobiles and airplanes
- Embedded processors: phones, portable electronics, etc.
- DSP processors: digital audio and TV systems...

C-Programming

- Evolved over the years: 1972 – C invented
- 1978 – *C Programming Language specification* published
- 1989 – C89 standard (known as ANSI C or Standard C)
- 1990 – ANSI C adopted by ISO, known as C90
- 1999 – C99 standard
    - mostly backward-compatible but not completely implemented in many compilers
- 2007 – new C standard C1X announced (also known as C11)
- The latest Standard - ISO/IEC 9899:2018
- In this course: ANSI/ISO C onwards standard

Reference: https://en.wikipedia.org/wiki/C11_(C_standard_revision)

Latest Standard: https://www.iso.org/standard/74528.html

C-Programming

- Evolved over the years: 1972 – C invented
- 1978 – *C Programming Language specification* published
- 1989 – C89 standard (known as ANSI C or Standard C)
- 1990 – ANSI C adopted by ISO, known as C90
- 1999 – C99 standard
    - mostly backward-compatible but not completely implemented in many compilers
- 2007 – new C standard C1X announced (also known as C11)
- The latest Standard - ISO/IEC 9899:2018
- In this course: ANSI/ISO C onwards standard

Reference: https://en.wikipedia.org/wiki/C11_(C_standard_revision)

Latest Standard: https://www.iso.org/standard/74528.html

Identifier Naming Rules

- In C language any name is called identifier.
- This name can be
  - variable name,
  - function name,
  - enum constant name,
  - goto label name
- Any other data type name like
  - structure, union or typedef name

C-Programming

Identifier Naming Rules

Rule 1: **Name of identifier includes alphabets, digit and underscore.**

Rule 2: **First character of any identifier must be either alphabets or underscore.**

Rule 3: **Name of identifier can’t be any keyword of c program.**

Rule 4: **Name of identifier is case sensitive i.e. num and Num are two different variables.**

Rule 5: **Must not contain white space.**

Constants/Literals in C

- **Constants:** The constants refer to fixed values that the program may not alter during its execution.
- These fixed values are also called **literals**.
- Constants can be of any of the basic data types like
  - an **integer constant**,
  - **a floating constant,**
  - **a character constant,** or
  - **a string literal.**
- There are also enumeration constants as well.
- The constants are treated just like regular variables except that their values cannot be modified after their definition.

Integer literals

- An integer literal can be a **decimal, octal, or hexadecimal** constant.
- A **prefix** specifies the base or radix: **0x** or **0X** for **hexadecimal**, **0** for **octal**, and **nothing** for **decimal**.
- An integer literal can also have a **suffix** that is a combination of U and L, for **unsigned** and **long**, respectively.
- Here are some examples of integer literals:

```
85        /* decimal */
0213      /* octal */
0x4b      /* hexadecimal */
30        /* int */
30u                 /* unsigned int */
```

Floating-point literals

- A floating-point literal has an integer part, a decimal point, a fractional part, and an exponent part.
- You can represent floating point literals either in decimal form or exponential form.
- While representing using decimal form, you must include the decimal point, the exponent, or both and while representing using exponential form, you must include the integer part, the fractional part, or both. The signed exponent is introduced by e or E.
- Here are some examples of floating-point literals:

  - **3.14159   /* Legal */**
  - **314159E-5L   /* Legal */**
  - **510E   /* Illegal: incomplete exponent */**

Character constants

- Character literals are enclosed in single quotes,
- e.g., 'x' and can be stored in a simple variable of char type.
- A character literal can be
  - a plain character (e.g., 'x'),
  - an escape sequence (e.g., '\t'),
- There are certain characters in C when they are preceded by a backslash they will have special meaning and they are used to represent like newline (\n) or tab (\t).

Escape sequence	Meaning
\\	\ character
\'	' character
\"	" character
\?	? character
\a	Alert or bell
\n	New line
\t	Horizontal tab

C Language - Character Set

Character set in C language are grouped under following major categories:

- ❑ **Letters (A…Z, a…z)**
- ❑ **Digits (0, 1, …9)**
- ❑ **Special Characters (; : , “ “ & * ! ,etc.)**
- ❑ **White spaces (Blank space, Tab, New line, etc.)**

➢ **C Tokens, Keywords and Identifiers**

Examples / Demonstration

Variables

A ***variable*** is named link or reference to a value stored in the memory.

Consider following declarations:

int rollno, marks;

In this statement:

- *rollno* and *marks* are variable name.
- *int* is a data type.

Examples / Demonstration

Variables

Declaring variables

Must declare variables before use

- Examples of Variable declaration:

```
int number;

float percentage;
```

- int - integer data type
- float -floating-point data type
- Many other types (will be discussing soon. . .)

Variables

Initializing variables

- Uninitialized variable assumes a default value/may results in inappropriate value **(Demonstration)**
- Variables can be initialized using **assignment operator (=)**

```
number = 3;
```

Can also initialize at declaration:

```
float percentage = 81.60;
```

- Possible to declare/initialize multiple variables at once:

```
int a, b, c = 5, d = 10;
```

Data types

The **data type** of an object in memory determines

- the set of values it can have and
- what operations that can be performed on such objects (variable)

C has a small family of data types

- Numeric (int, float, double)
- Character (char)
- User defined (struct, union)

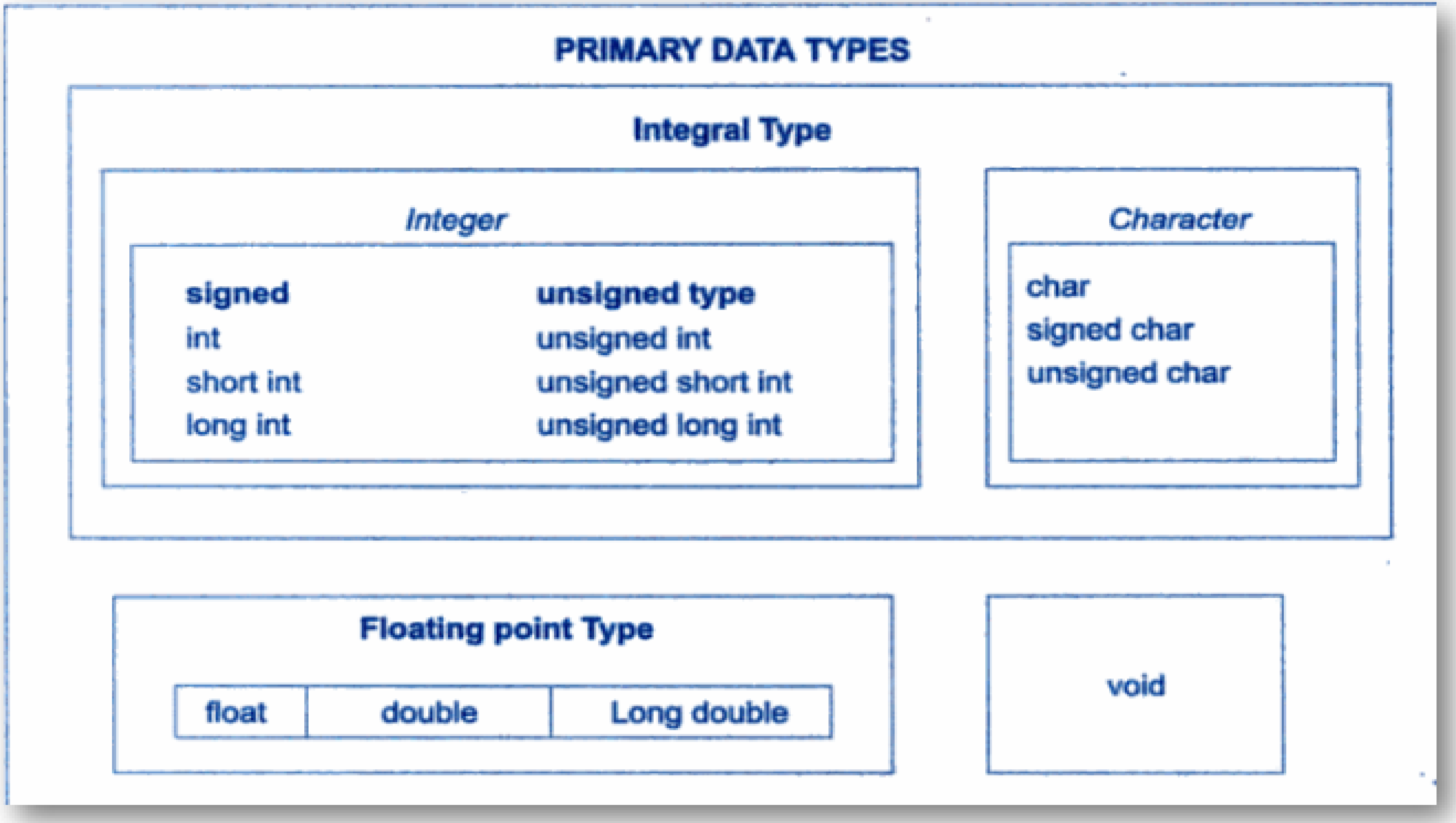
Data types

ANSI C Supports three classes of Data types:

- ➢ Primary data types
- ➢ Derived data types
- ➢ User-defined data types

Example / Demonstration and Discussion

Data types – Primary data type



PRIMARY DATA TYPES

Integral Type

Integer

signed	unsigned type
int	unsigned int
short int	unsigned short int
long int	unsigned long int

Character

char
signed char
unsigned char

Floating point Type

float	double	Long double

void

Reference: Programming in ANSI C by E Balagurusamy

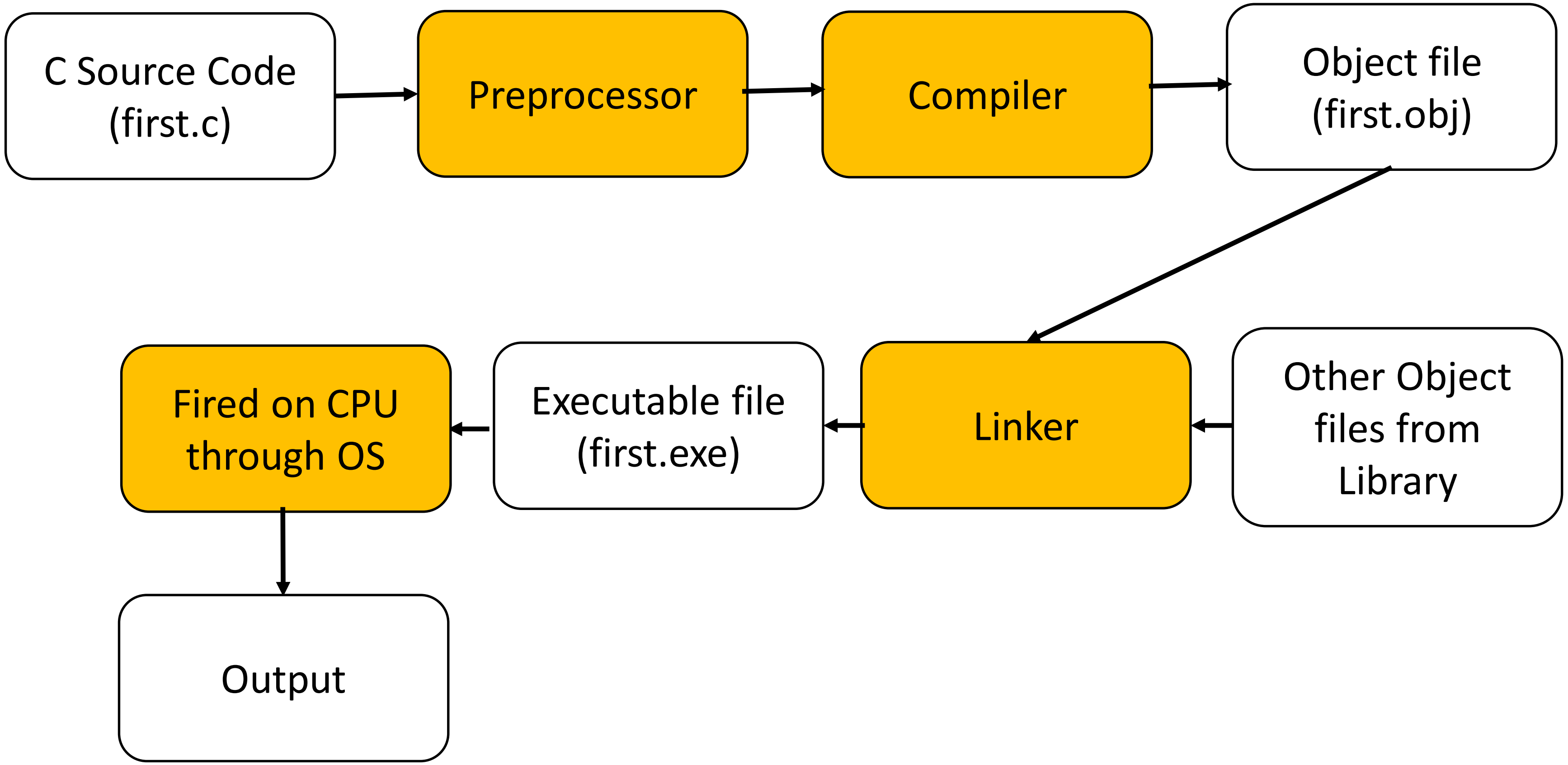
“C” Language

- General Purpose Language
- It is Procedural and Structured.
- Suitable and efficient for Machine Instruction code types hence utilized in development of System software like compiler, interpreter, operating systems, driver development, programming embedded systems etc.
- It is ANSI and ISO standard.
- Developed originally at Bell Labs by Dennis Ritchie around 1973.
- As the “C” footprint is small, it is most appropriate for computational intensive task and constrained memory conditions.

“C” – Process of Compilation

- “C” is a compiled language.
- A compiler checks whole source code from beginning to end. If the code is error-free than converts to machine code.
- Different compilers for different Operating System and CPU architectures are available in the market.
- Eg GNU or GCC compiler, Borland Turbo C Compiler, Borland C Compiler for Windows OS, etc.

“C” – Process of Compilation



"C" – Process of Compilation

- **Preprocessor** : It is simple text substituting tool done before compilation. It aids programmer to define short-names for complex identifiers. It also directs compiler for important task so as to optimize the process of compilation.

  All preprocessor directive starts with # (hash) sign.

  eg. #define pi 3.14   Here pi is substituted by 3.14

  eg. #include <stdio.h> Here stdio.h library is added to current source code.

- **Compiler** : The compiler is a program which generates equivalent machine understandable code from after reading the whole source code. The C compiler generates Object code. The Object code contains relocation information of other object files so as to provide information to linker.
- **Linker** : Linker links several object files along with the main linker file to create absolute machine code stored in Executable file (.exe).

“C” – Program Structure

```
/* Comment */
# Preprocessor Directive
# Another Preprocessor Directive
Global Declarations;
User Define Function Prototype();
User Defined Function Definition()
{
   statement 1; statement 2; statement n;
}
Structure definition
{
}
Union definition
{
}
void main()//Starting point of C program
{
     statement 1; statement 2; statement n;
}
```

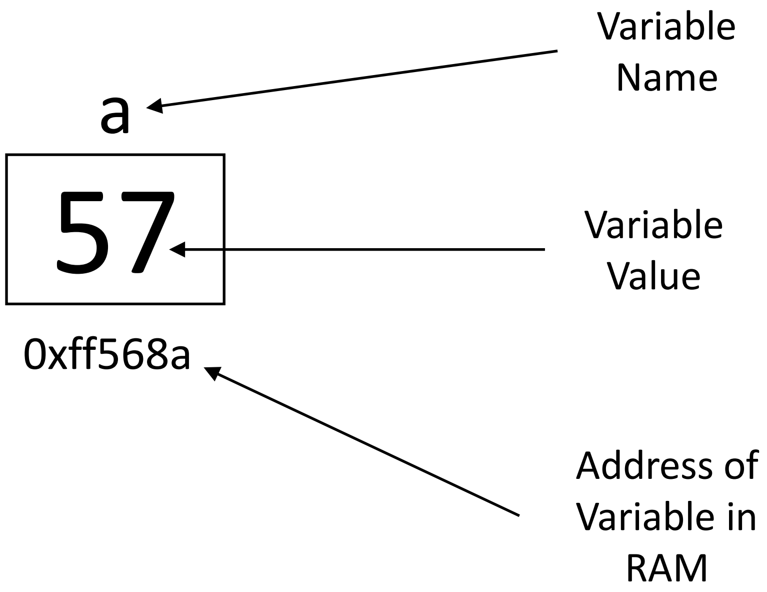
“C” – Program Structure

```
/* First C Program */  //This is Comment
# include<stdio.h> //file inclusion preprocessor
# define pi 3.14 //macro expansion preprocessor
int g=300; //Global variable
void callme(); //function prototype
void main()  //Starting point of C program
{
     printf(“Hello, This is my first C Program”);
     callme(); //user defined function call
     int x=30; //local variable
}
void callme() //user defined function definition
{
    printf(“User Defined Function”);
}
```

Variables and Datatypes

Variable : A variable is a named memory location in RAM whose value can be altered throughout life time of the execution of program.

Eg. int a=57;



a

57

0xff568a

Variable Name

Variable Value

Address of Variable in RAM

Variables and Datatypes

Variable Naming Rules :

1. Variable names include alphabet, numbers and underscore.
2. It should always start with alphabet or underscore. The following letters can be number.
3. No symbols are allowed in variable name.
4. No keywords/reserved words are allowed as variable name.
5. Names are case sensitive , hence age=12 and Age=12 are two different variables.
6. Spaces are not allowed in variable name.
7. Only first 31 characters are significant in variable name.

Variables and Datatypes

DataTypes of C Language :

C provides 3 categories of datatype.

1. Primary(Built-in) Data Types:
   void, *int*, long, *char*, *double* and *float*.
2. Derived Data Types:
   Array, *References*, and *Pointers*.
3. User Defined Data Types:
   Structure, *Union*, and *Enumeration*.

Variables and Datatypes

DataTypes of C Language :

The size of datatype in C language varies from compiler to compiler. The size of datatypes mentioned below is for GCC compiler.

Primary(Built-in) Data Types:
void, *int*, long, *char*, *double* and *float*.

1. void : It is a datatype in ‘C’ for which storage size is not visible in RAM.

   Variables with void type cannot be declared. But yes, void pointers can be declared to store address of function returning void types.

   eg . void i; //Invalid
   void *i; // valid

2. short : It is used to store integer data. It occupies 2 bytes in memory. %hd is the format specifier.

Variables and Datatypes

DataTypes of C Language :

3. int and long : It is used to store integer data. Both occupies 4 bytes in memory. Few compilers allocates 8 bytes for long datatype. %d and %ld are format specifier for int and long respectively.

   Eg. int i=56;

4. float and double : Used to store real nos. Float and double occupies 4 and 8 bytes respectively. %f and %lf are format specifier for float and double types respectively.

   Eg float x=45.5;

5. char : It occupies 1 bytes in memory. It stores characters and integers too.

   Eg. char ch=‘A’; or char ch=93;

Variables and Datatypes

```
//Demo for datatypes and its size on GCC compiler.
#include<stdio.h>
#include<conio.h>
void main()
{    short g=45;
     printf("\n short is %hd",g);
     printf("\n size of short is %d",sizeof(g));
     int i=668;
     printf("\n int i is %d",i);
     printf("\n size of int is %d",sizeof(i));
     long v=56666;
     printf("\n long value is %ld",v);
     printf("\n size of long is %d",sizeof(v));
     float j=45.7888;
     printf("\n float is %.2f",j);
     printf("\n size of float is %d",sizeof(j));
     double k=34.788;
     printf("\n Double is  %g",k);
     printf("\n size of double is %d",sizeof(k));
     char ch='A';
     printf("\n Character is %c  ",ch);
     printf("\n size of character is %d",sizeof(ch));
    getch();
}
```

Variables and Datatypes

```
#include<stdio.h>
void main()
{
  printf("First C Program");
  printf("\n\nAhmedabad University");
  int a=12; //4 bytes
  int b=60;
  printf("\nEnter first value:");
  scanf("%d",&a);
  printf("\nEnter another value:");
  scanf("%d",&b);
  printf("\nValue of a is %d",a);
  printf("\nValue of b is %d",b);
  int s=a+b;
  printf("\n\nAddtion of %d and %d is %d",a,b,s);
  float pi=3.14;
  printf("\nFloat value is %.2f",pi);
  char ch='a';
  printf("\n\nCharacter is %d",ch);
  getch();//waits at output screen
}
```

Variables and Datatypes

```
#include<stdio.h>
#include<conio.h>
void main()
{
   char name[40];
   printf("Enter your name:");
   scanf("%[^#]",name); //Accepts string until # is entered
   fflush(stdin); //clears input buffer
   printf("\nGood Day %s",name);
   getch();
}
```

Decision Making

Control Statements : Control statements are statements which are responsible for directing and controlling flow of execution of program.
Control statements are condition or data driven.

Conditions : Conditions are expressions which evaluates to true or false. They are also known as Boolean or logical expressions.
To create a condition Relational and Logical operators are used.

Relational Operators : Sets the relation between raw data or stored in variables. Relational operators are >, <, >=, <=, == (equal to), != (not equal to).

```
 eg.  5>3 (condition is true)
        int a=5,b=3
        a<b (condition is false)
```

Decision Making

Logical Operators : Logical Operators are used to check multiple conditions simultaneously. They are && (logical AND), || (logical OR) and ! (NOT).
Eg.   5>3 && 9>2 (condition is true)

Truth Table for &&
True && True   -> True
True && False -> False
False && True   -> False
False && False -> False

Truth Table for ||
True || True -> True
True || False -> True
False || True -> True
False || False -> False

Truth Table for !
! True -> False
! False -> True

Decision Making

If statement : If is considered as decision making statement. Its different forms are mentioned below.

Form 1:

```
if ( condition )
{
   statement 1;
   statement 2;
}
```

Here if condition is true then if block gets executed, otherwise not.

Eg.

```
Int a=6,b=2;
If(a>b)
{
    printf(“C programming”);
}
```

Output :

```
C Programming
```

Decision Making

Form 2:

```
if ( condition )
{
     statement 1;
     statement 2;
}
else
{
    statement 3;
    statement 4;
}
```

Here in form 2, if condition is true then if block gets executed other wise else block gets executed. Hence any one block out of two gets executed. The control moves in else only if condition is false.

Eg

```
if(5 % 2 ==0)
{
    printf(“Even number”);
}
else
{
     printf(“Odd Number”);
}
```

Output:

```
Odd Number
```

Decision Making

Form 3: (else if ladder chain)

```
if( condition 1 )
{
     statement block 1:
}
else if( condition 2 )
{
    statement block 2;
}
else if( condition 3 )
{
    statement block 3;
}
else
{
    statement block 4;
}
```

Here out of 4 blocks only one block get executed. Here control moves to else only if condition is false otherwise its executes any one block whose condition is true and moves out completely from else if ladder chain.

Decision Making

Eg.

```
int x=2;
if( x==1 )
{
    printf(“ONE”);
}
else if( x==2 )
{
    printf(“TWO”);
}
else if( x==3 )
{
    printf(“THREE”);
}
else
{
   printf(“No matching if );
}
```

Output :

```
TWO
```

Decision Making

Form 4: (nested if)

```
if( condition 1 ) //outer if
{
   statement A; //statement of outer if
      if( condition 2 ) //inner if
      {
             statement 1;
             statement 2;
      }
       else   //else of inner if
       {
            statement 3;
            statement 4;
       }
      statement B; //statement of outer if
}
else  //else if outer if
{
      statement C; //statement of outer if..else
      statement D;
}
```

Hence if can be nested in any required manner with proper syntax.

Decision Making

Switch Case Control : Switch Case control behaves similar to else_if ladder. The difference is else if ladder is condition driven while switch case control is value driven.

```
switch (value)
{
      case 1:
             statement 1;
             statement 2;
             break;
      case 2:
            statement 3;
            statement 4;
            break;
     case 3:
            statement 5;
            statement 6;
            break;
      default:
             statement 7;
}
```

Here the case execution depends on the value inputted.

Decision Making

Eg:

```
int x=2;
switch (x) // Here x can be integer or character value
{
     case 1:
           printf(“ONE”);
           break;
     case 2:
          printf(“TWO”);
          break;
    case 3:
          printf(“THREE”);
          break;
     default:
           printf(“No matching case”);
}
```

Here the output is “TWO” as value of x is 2, the control moves to case 2 and after displaying TWO, break statement positions control out of switch. Here break is not mandatory. If break is not mentioned, the control moves to following cases.

Decision Making

goto statement : goto statement forwards control to defined label

Eg

```
#include<stdio.h>
void main()
{
    int x=0;
    uplabel:
    x++;
    printf("%d ",x);
    if(x==10)
    {
      goto downlabel;
    }
    goto uplabel;

    downlabel:
    getch();
}
```

Output

```
1   2   3   4   5   6   7   8   9   10
```

Operator Precedence

Precedence	Operator	Description	Associativity
1	++ --	Suffix/postfix increment and decrement	Left-to-right
	()	Function call	
	[]	Array subscripting	
	.	Structure and union member access	
	->	Structure and union member access through pointer	
	(type){list}	Compound literal(C99)	
2	++ --	Prefix increment and decrement[note 1]	Right-to-left
	+ -	Unary plus and minus	
	! ~	Logical NOT and bitwise NOT	
	(type)	Cast	
	*	Indirection (dereference)	
	&	Address-of	
	sizeof	Size-of[note 2]	
	_Alignof	Alignment requirement(C11)	
	*= /= %=	Assignment by product, quotient, and remainder	
	<<= >>=	Assignment by bitwise left shift and right shift	
	&= ^= |=	Assignment by bitwise AND, XOR, and OR	

Operator Precedence

Precedence	Operator	Description	Associativity
3	* / %	Multiplication, division, and remainder	Left-to-right
4	+ -	Addition and subtraction	
5	<< >>	Bitwise left shift and right shift	
6	< <=	For relational operators < and ≤ respectively	
	> >=	For relational operators > and ≥ respectively	
7	== !=	For relational = and ≠ respectively	
8	&	Bitwise AND	
9	^	Bitwise XOR (exclusive or)	
10	|	Bitwise OR (inclusive or)	
11	&&	Logical AND	
12	||	Logical OR	
13	?:	Ternary conditional	Right-to-left
14	=	Simple assignment	
	+= -=	Assignment by sum and difference	
	*= /= %=	Assignment by product, quotient, and remainder	
	<<= >>=	Assignment by bitwise left shift and right shift	
	&= ^= |=	Assignment by bitwise AND, XOR, and OR	
15	,	Comma	Left-to-right