Identifiers

name given to a variable or a funct to identify

ex int a,b,c ;abc are identifiers

keywords in c :

32 of them and they cant be used as identifiers

they are :

1. auto break case char continue do default int long register return signed unsigned ststic sizeof short struct switch typedef union void double else Enum extern for if goto float while volatile

VAriables:

variable name can have digits alpahabets and underscore but cant start with a digit and it csnt be a keyword

Datatypes:

type of data it holds and how much amount of memory it occupies

int - 4bytes

1. Primitive datatypes:

* Integer types:
  + Int -2 or 4 bytes
    - Unsigned
    - Signed
  + Short int – 2 bytes
    - Unsigned short int
    - Signed short int
  + Long int -4 bytes
* floating points:
  + default is taken to be double when taken as decimal
  + float-4 bytes
  + double 8 bytes
  + long double -8 bytes
* character type : ASCII values
  + char – 1 byte
    - unsigned char
    - signed char

**scanf**()-in stdio.h for input

syntax: int scanf(“format specifier ”,&variable);

if all the inputs are successfully read then scanf() returns count of number of inputs read. if it fails then it returns a smaller no or -ve no. strings don’t need an amperscent in scanf

&- an address operator/pointer which tells the compiler where the address of variable is .

Pow()-in math.h used to get power of a variable to a certain power

Syntax:pow(variable,power);

**OPERATORS:**

Types : based on operations performed and based on number of operands it can take

1. 1.operational:
   1. Arithmatic:
      1. +,-,/,%,\*
   2. Operational:
      1. <,>,!=,==,<=,>=
   3. Shorthand:
      1. +=,++,-=,--
   4. Logical: boolean answers
      1. &&-logical and (a Lazy evaluation or short circuit i.e if first condition is false then directly ans is false)
      2. ||-Logical or
      3. !-Logical not
   5. Increment or decrement: only works on a single variable
      1. Unary operators:
         1. Increment(++): increase by 1
         2. Decrement(--):decrease by1
            1. Post inc/decr: it increases the values after the assignment

Ex: x++;x—

* + - * 1. Pre incre/decre:it increases/changes value before the assignment

Ex:++x,--x

* + - * 1. Order of evaluation is unkown
    1. Tereneary operators:
       1. 3 operands:
          1. Ex:a?b:c

Where if a is true then b gets printed and if a is false the c is the result

1. BITWISE:
   1. Doesn’t work on double or float
      1. &-Bitwise AND
      2. |-Bitwise OR
      3. ^-Bitwise XOR
      4. ~-Bitwise NOT
      5. >>-Bitwise RIGHTSHIFT
         1. Shifts bits to left .
            1. Ex: x<<3;
      6. <<-Bitwise LEFTSHIFT
         1. Check if a bit is set or not => c&(1<<i)
         2. To set a bit =>c|(1<<i)
         3. To clear the bit =>c^(1<<i) or c&~(1<<i) where i is position of bit and c is the number
2. SIZEOF:
   1. Format specifiers are %lu,%zu,%u
   2. Used to determine the size in bytes of a datatype or a variable during compile time
   3. Syntax: sizeof(expression/datatype/variable) ; printf(“%lu”,sizeof(“datatype”))
   4. It returns size\_t which Is unsigned integer

Ones complement formula: x=-(x+1).

Its used to subtract numbers

**Branching and conditional statements:**

* + - Simple if
      * Syntax: if(condition)

{statement block;}

* + - If else
      * Syntax: if(condition)

{statement block;}

else

{statement block;}

* + - If else if
      * Syntax: if(condition)

{statement block;}

else if(condition)

{statement block;}

else if(condition)

{statement block;}

……

else

{statement block;}

* + - Nested if
      * syntax : if(condition)

{

if(condition)

{statement block;}

}

else {

if(condition)

{statement block;}

}

* + - Switch
      * Syntax: switch(expression){

case constant1:{statement;break;}

case constant2:{ statement;break;}

………

case constantn:{ statement;break(optional);}

default:

}

* + - * Case constant can have integer ,character, enumerated data
    - Ds
    - While:
      * Syntax :while(condition)

{statements; increment/decrement;}

* + - * + While(1){} is an infinte loop till it encounters a break/exit
    - For:
      * Syntax:for(initialize;condition;incre/decrement)

{}

* + - * For(; ;) is an infinite loop
      * Continue skips one iteration of the loop

Euclidian algo for gcd:

A b a>b a=a-b b>a b=b-a

120 65 true a=55 - -

55 65 false - true b=10

55 10 t a=45 - -

45 10 t a=35 - -

35 10 t a=25 - -

25 10 t a=15 - -

15 10 t a=5 - -

5 10 f - t b=5

Gcd=a=b;

Euclidian algo for gcd using modulo:

A b rem=a%b

120 65 55

65 55 10

55 10 5

10 5 0

5 0 -

1. Unformatted input output functions:
   1. Present in stdio.h
   2. Getchar()
      1. It only gets u one chara at a time from stdin-keyboard
   3. Putchar():
      1. It prints only one char at a time to stdout-terminal
   4. Gets()
   5. Puts()

Unit 2

MACROS:

* + EOF: end of file
    - Predefined macro with default value -1

Flag: variable that gives if the condition u set is true

Typecasting :

(datatype)variable.

POINTERS:

Variable which hold address of another variable

‘\*’Is a dereferencing operator.

Dereferencing an dangling operator is an error

Dangling pointer is a pointer that doesn’t point to anything or the address it was pointing to gets deleted.

ARRAYS:

Initialisations of arrays:

Syntax: datatype variable[size]={values};if partial values are given then rest of the values are 0;

Ex: int a[3]={0} so elements in array are {0,0,0}

int a[3]={2} so elements in array are {2,0,0}

wrong initialisation => int a[7]={}; not allowed in c;

designated initalisation=> int a[7]={[2]=4,[6]=888}; the elements r {0,0,4,0,0,0,888}

traversing an array using pointers:

an array variable is a constant pointer. Therefore we can use the array variable is used to point to its variables.

2d array:

Syntax: datatype variable\_name[size1][size2]

Accessing elements:

Array notation: Pointer notation:

Nested for() \*(\*(arr+i)+j) => i:row,j=column

Outer for() => rows \*(arr+i)+j gives the address

Inner for() => columns arr+I,\*(arr+i)+j are all pointers

Arr[i][j]

Invalids:

Int a[5][]; is invalid as number of columns cant be infinite as it will go on 4ever

2d arrays with pointers:

FUNCTIONS:

Syntax:

Return\_datatype Function\_name(datatype arg1,datatype2 arg2,…..)

{

Block of the function

}

Function call: invoke the function

Function definition : define the function

Function prototype/declaration: when function is defined before main

Functions cannot be defined In another function