C

31 May 2023 12:59

1. A procedural language
2. Literals
   a.

Integers	3	-902

   b.

Floating	3.1313	6.02e23

   c.

Characters	'#'	'\n'	'A'

   d.

Strings	"Michael Jordan"	"priv7@rvb.com"

1.

Identifiers	ENG2	ee_2a

2.

Keywords	auto	break	int	return	unsigned	static	const

3.

Operators	+ - / % : ?

4. Data types
   a. int 2B in 16-bit, 4B in 32-bit
   b. short 2B
   c. Long 4B
   d. float 4B
   e. double 8B
   f. char 1B
5. A variable can be declared many times but can be defined only once.

```
int i = 5; //initialization
float a {1.792}, b;
char c = 'a';
char str[] = "penguin";   //stored as p e n g u i n \0
```

6. operators:

arithmetic	logical || && !	relational == <=	bitwise << >> ~ | & ^
increment ++ --	assignment += *= =	conditional (: ?)	dot . comma , arrow ->

7. Operator Precedence
8. Qualifiers:

size qualifiers	sign qualfiers

9. Type Conversion: implicit, explicit (char)67 //yields 'C'
10. String formatting scanf("%d**%*c**%d", &a, &b); //if we enter 2,3 then comma(,) will be ignored
    "%10d" justifies the number to left with whire space padded to the right
    "%-10d" justifies the number to right with whire space padded to the left
    "%.3f", prints floating number with 3 decimal places

11. enumerated types:

```
enum flag{
      warn, crash, clean};

flag f1 = warn; //0
flag f2 = clean;//2

enum values are by default initialized starting from 0

enum hardness{
      beginner=800,
      mediocre=1500,
      advanced=2500};
```

12.

typedef char * string;	typedef int& INTREF;	typedef struct student STU;

13. Structures: is a composite data type declaration that defines a physically grouped list of variables under one na
    memory,

```
struct book{
      char name;
      float price;
      int pages;
      };
struct book b1;
b1={'A',10.99,200};    //initialization
b1.price=12;           //accessing data

Struct book[10];   //Array of structures

//structures can be nested
```

14. Unions: It is a collection of variables of different datatypes in the same
    memory location. We can define a union with many members, but at a given point
    of time only one member can be updated or read.

```
union Shakespeare_play
```

Operator	Description	Associativity
() [] . ->>> ++ --	Parentheses or function call Brackets or array subscript Dot or Member selection operator Arrow operator Postfix increment/decrement	left to right
++ -- + - ! ~ (type) * & sizeof	Prefix increment/decrement Unary plus and minus not operator and bitwise complement type cast Indirection or dereference operator Address of operator Determine size in bytes	right to left
* / %	Multiplication, division and modulus	left to right
+ -	Addition and subtraction	left to right
<< >>	Bitwise left shift and right shift	left to right
< <= > >=	relational less than/less than equal to relational greater than/greater than or equal to	left to right
== !=	Relational equal to or not equal to	left to right
&&	Bitwise AND	left to right
^	Bitwise exclusive OR	left to right
|	Bitwise inclusive OR	left to right
&&	Logical AND	left to right
||	Logical OR	left to right
? :	Ternary operator	right to left
= += -= *= /= %= &= ^= |= <<= >>=	Assignment operator Addition/subtraction assignment Multiplication/division assignment Modulus and bitwise assignment Bitwise exclusive/inclusive OR assignment	right to left
,	comma operator	left to right

```
cook@pop-os:~$ ascii -d
  0 NUL    16 DLE    32      48 0    64 @    80 P    96 `   112 p
  1 SOH    17 DC1    33 !    49 1    65 A    81 Q    97 a   113 q
  2 STX    18 DC2    34 "    50 2    66 B    82 R    98 b   114 r
  3 ETX    19 DC3    35 #    51 3    67 C    83 S    99 c   115 s
  4 EOT    20 DC4    36 $    52 4    68 D    84 T   100 d   116 t
  5 ENQ    21 NAK    37 %    53 5    69 E    85 U   101 e   117 u
  6 ACK    22 SYN    38 &    54 6    70 F    86 V   102 f   118 v
  7 BEL    23 ETB    39 '    55 7    71 G    87 W   103 g   119 w
  8 BS     24 CAN    40 (    56 8    72 H    88 X   104 h   120 x
  9 HT     25 EM     41 )    57 9    73 I    89 Y   105 i   121 y
 10 LF     26 SUB    42 *    58 :    74 J    90 Z   106 j   122 z
 11 VT     27 ESC    43 +    59 ;    75 K    91 [   107 k   123 {
 12 FF     28 FS     44 ,    60 <    76 L    92 \   108 l   124 |
 13 CR     29 GS     45 -    61 =    77 M    93 ]   109 m   125 }
 14 SO     30 RS     46 .    62 >    78 N    94 ^   110 n   126 ~
 15 SI     31 US     47 /    63 ?    79 O    95 _   111 o   127 DEL
```

```
#include <iostream>
using namespace std;

    struct book
    {float price;
    int pages;
    char name;};

    void show(struct book &x){
        printf("%d", x.pages);
    }

int main(){
    struct book b1={20, 100, 'a'};
    show(b1);
    return 0;
}
```

Pass by reference

```
#include <stdio.h>

struct book
    {float price;
    int pages;
    char name;};

void show(struct book* x){
    printf("%d",x->pages);}

void main(){
    struct book b1={20,100,'a'};
    show(&b1);
}
```

pass by address

```
#include <stdio.h>

struct book
    {float price;
```

```
{
        char narration[1000];
        char dialogue[50000];
};
```

15. CONTROL FLOW:

```
if(test)
        {---}
else if(test)
        {---}
else
        {---}

for(int i=1, float j=0.002 ; i<10 && j!=0; j+=0.03)
        {---}

while(test)
        {---}

do
        {---}
while(test);

switch(test var){
        case a: ---
        case b: ---
        default: ---}
```

note that only char and int vars can be tested in switch

break	*continue-skips past follwing statements to next iteration*

16. Arrays: elements stored contiguously

```
int n[10] = {1,2,3,4,5,6,7,8,9,0};
n1[0]; //access 0th element
/*Caution C/C++ may warn but does not offer array bound validation n[20] can be read even though it is not a part
of the array*/

int n2[][2]={{2,3},
        {3,-2}};    //a 2*2 array

int n3[2][1][2]={{3,2},
                 {4,3},
                 {6,4}};  //a 2*1*2 array

/*when accessing n1[3] it means *(n+3)
  when accessing n2[1][0] it means *(*(n1+1)+0)
  This essentially means that just the name of an array is actually a pointer variable,
  so n1, n2, n3 are actually pointers (to the first elements)*/

/*by default arrays are passed by reference*/
```

17. Strings:

```
char s[]="monkey magic!";  /sizeof(s) yields 13 bytes but strlen(s) yields 12
```

builtin functions in string.h

strlen()	*strcat(target,source,optional int n)*	*strcmp(a,b)*	*strcpy(target,source,optional int n))*
strupr()	*strlwr()*		

18. functions: a block of code which only runs when it is called. You can pass data, known as parameters, into a function. parameters can be passed by value, address, reference

```
int mail(int ID); //function prototype

// int mail(int); function signature

int mail(int ID)
{---
return 0;}
int mail(int ID = 29030); //default value
/* a function returns only once. If a return statement is encountered, the function exits
and the stack space is cleared !!*/
inline add(int x, int y)
    {return x+y;}
```

/*inline functions are like macros. They are replaced by their body at places of incocation by preprocessor*/

Scope and extent of variables:

- global variables are defined outside all functions and live as long as the script
- Local variables are defined inside blocks {} and have scope local to those blocks. Variables declared inside the functions live untill the function is revoked and stack space cleared.

Storage Specifier	Storage	Initial value	Scope	Life
auto	Stack	Garbage	Within block	End of block
extern	Data segment	Zero	global	Till end of program

```
#include<stdio.h>
struct book{
    int pages;
    char name;};

void show(struct book x){
    printf("%d",x.pages);}

void main(){
    struct book b1={20,100,'a'};
    show(b1);
}
```

pass by value

			Multiple files	
static	Data segment	Zero	Within block	Till end of program
register	CPU Register	Garbage	Within block	End of block

Storage classes

- If a local and a global variable has the same name, the local variable is given priority. To refer to the global variable, use scope resolution operator :: before variable name.

```
Int a = 999;
{
Int a = 1;
Print ( a );    //prints 1
Print(  ::a );   //prints 999
```

- Static variables live across various function calls
- Recursions: when a function invokes itself
- default paramter values can be specified either in prototypes or deinition

give_halloween(char name[10], int chocolates = 2)

- Variable number of arguments:

19. Pointers: pointers are just integers that point to a memory location. A pointer to a specific data type needs to be the same type as the variable.
Reference Variables: can be also be used in place of pointers (although pointer can be much more beneficial). Refernces are just another name of a variable, just like we have diiferent usernames on internet but refer to same person.

```
int a=5;
int &aref = a;  //a reference, just another name
int * ap = &a;  //a pointer to a

aref = 9000;    //a itself changed
*ap = -200;     //a changed

std::cout<<*ap;
```

```
int n = 67;
void *p = &n;
cout<<*(int*)p<<endl;        //yields 67
cout<<*(float*)p<<endl;      //yields 9.3887e-4
cout<<*(char*)p<<endl;       //yields 'C'
```

19. Macros

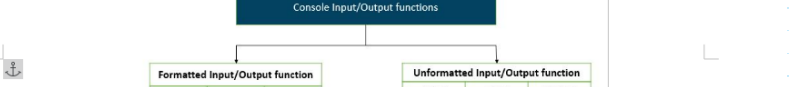
```
#define PI 3.1412                 //object like macro

#define SQUARE(x) (x*x)           //function like macro
#define CUBE(x) (x*SQUARE(x))     //chain macro

//multiline macro
#define YELL for(int i=0;i<9;i++)\
                printf("Haan Bhai Qubul Hai!!");
```

20. File inclusion

```
/*main.c*/
#include <stdio.h>
#include "myfiles.c"

void main(){
    int k=f1(1,2);
    f2();
    printf("\nThe value of G = %.11f\n",G);

}
```

```
/*myfiles.c*/
float G=6.606e-8;

int f1(int x,int y )
{
    return x+y;
}

void f2()
{
    printf("\nYes I love you too!\n");
}
```

21. I/O facilities



Formatted Input/Output function

Type	Input	Output
char	scanf()	printf()
int	scanf()	printf()
float	scanf()	printf()
string	scanf()	printf()

Unformatted Input/Output function

Type	Input	Output
char	getch() getche() getchar()	putch() putchar()
int	-	-
float	-	-
string	gets()	puts()

Function	Description
fopen()	Create a new file or open a existing file
fclose()	Closes a file
getc()	Reads a character from a file
putc()	Writes a character to a file
fscanf()	Reads a set of data from a file
fprintf()	Writes a set of data from a file
getw()	Reads a integer from a file
putw()	Writes a integer to a file
fseek()	Set the position to desire point
ftell()	Gives current position in the file
rewind()	Set the position to the beginning point

r open a text file for reading

w truncate to zero length or create a text file for writing

a append; open or create text file for writing at end-of-file

rb open binary file for reading

wb truncate to zero length or create a binary file for writing
ab append; open or create binary file for writing at end-of-file
r+ open text file for update (reading and writing)
w+ truncate to zero length or create a text file for update
a+ append; open or create text file for update
r+b or rb+ open binary file for update (reading and writing)
w+b or wb+ truncate to zero length or create a binary file for update
a+b or ab+ append; open or create binary file for update