C Language Basics

- Libraries, Header files, Preprocessor
- Constants, Variables, Data Types
- Standards for Comments, Spacing, Indentation
- Keywords, Operators, Expressions, Syntax
- Structures, Control Flow, Memory Management
- Lines end with semicolon (;)
- main() function is entry point
- Solve the problem at hand

Basic Program

```c
// Libraries and Header Files
#include <stdio.h>

// Constants and Macros
#define MAX 1000

/* Prints "Hello World!" to the screen. Note
   the indentation and spacing for readability. */
int main()
{
  // printf() function contained in stdio.h library
  printf("Hello World!\n");

  // Return 0 to indicate success
  return 0;
}
```

Data Types

Types

- `char` : character
- `int` : integer
- `float` : real number
- `double` : real number *(double precision)*

Qualifiers

- `signed` (default) / `unsigned` : `char`, `int` data types
- `short` / `long` / `long long` : `int` data type
- `long` : `double` data type
- `double double` : available but not standard

Data Type Sizes

Type	Bytes	Lower Bound	Upper Bound
`(signed) char`	1	-128	127
`unsigned char`	1	0	255
`(signed) int`	2	-32768	32767
`unsigned int`	2	0	65536
`long int`	4	-2^31	(2^31)-1
`float`	4	1.17e-308	3.4e38
`double`	8	2.2e-308	1.79e208

64-bit systems : float 8 bytes, double 16 bytes

Constant Declaration

- `const` qualifier or `#define`
- Standard constant names in uppercase

Scope of Data in a Program

- Variables are private and local to the function, values are released after the function quits.
- `extern` : Variables exist outside of any function, values are retained and available to all functions.
- `static` : Variables are local to a specific function, but retain their values after a function quits.

Variable Declaration

- Declare with type and initialize the values
- Name format : lowercase with underscores, CamelCase
- No data type for strings
- Name to context or use
- Null character '\0' not same ar '0'
- ' ' for single characters
- " " for strings

Consider the following pseduo code:

```
main()
{
   ....
}

int what_scope;
float end_of_scope[10];

void what_global()
{
   ....
}

char alone;

float fn()
{
   ....
}
```

Printf()

- `int printf(char *format, arg list ...);` -- prints to stdout the list of arguments according specified format
  Returns number of characters printed.
- The format string has 2 types of object:
  - ordinary characters -- these are copied to output.
  - conversion specifications -- denoted by `%` and listed

```
printf("Hello World!\n");          Output: Hello World!
printf("%-2.3f\n",17.23478);       Output:  17.235
printf("VAT=17.5%%\n");            Output: VAT=17.5%
```

Format	Type	Result
`%c`	`char`	single character
`%i` or `%d`	`int`	decimal integer
`%o`	`int`	octal integer
`%x` or `%X`	`int`	hexadecimal integer
`%u`	`int`	unsigned integer
`%s`	`char` array	string terminated by \0

Format	Type	Result
`%f`	`float` / `double`	real number -m.ddd..
`%e` or `%E`	`float` / `double`	scientific format m.dddd..
`%g` or `%G`	`float` / `double`	%f or %e format *(compact)*
`%p`	pointer	address in hexadecimal
`%%`		`%` character

Sequence	Name	Description
`\a`	Alarm or Beep	generate a bell sound
`\b`	Backspace	move the cursor one place backward
`\f`	Form Feed	start of the next logical page
`\n`	New Line	start of the next line
`\t`	Horizontal Tab	insert whitespace to the left
`\v`	Vertical Tab	insert vertical space

Sequence	Name	Description
`\\`	Backlash	insert backslash character
`\'`	Single Quote	display single quotation mark
`\"`	Double Quote	display double quotation marks
`\?`	Question Mark	display question mark
`\0`	NULL	NULL character

Scanf()

- int scanf(char *format, args....) -- reads from stdin and puts input in address of variables specified.
- Format control string similar to printf
- ADDRESS of variable or a pointer to one is required
- Name of an array or string to scanf

```
int number = 0;
float realnumber = 0.0;
char string[80];
scanf("%d",&number);
scanf("%f",&realnumber);
scanf("%s",string);
```

Operators

- Primary:

  . and -> are used to access members of a `struct`

  [] is used to access elements of an array

  () is used to override precedence and to call functions

- Unary:

  * and & are used to access values through pointers

  - and ! are used to negate

  ++ and -- are used to increment and decrement

  sizeof is used to determine the size of a data type

  () is used to cast a value to a different type

```c
// Postincrement / Postdecrement
int c = 5;
printf("%d is c", c);
printf("%d is c++", c++);
printf("%d is c", c);
printf("%d is c--", c--);
printf("%d is c", c);
// Preincrement / Predecrement
c = 5;
printf("%d is c", c);
printf("%d is ++c", ++c);
printf("%d is c", c);
printf("%d is --c", --c);
printf("%d is c", c);
```

Operators

- Order of operations : () * / % + - =
- Addition and Subtraction : +, -
- Pre and Post Increment and Decrement : --, ++
- Multiplicative : *, /, %
- Assignment : =, +=, -=, *=, /=, %=
- Compare : ==, !=, >, <, >=, <=
- Bitwise : &, ^, |, <<, >>, ~ (One's Complement)
- Logical : &&, ||, !
- Ternary : *condition* ? *true* : *false*

int a = 5, b = 21, c = 3, d = 5, e = 4, f = 6, g = 12;

Operator	Sample	Explain	Assign
++	a++	a = a + 1	6 to a
--	--b	b = b - 1	20 to b
+=	c += 7	c = c + 7	10 to c
+=	a += e	a = a + e	10 to a
-=	d -= 4	d = d - 4	1 to d
*=	e *= 5	e = e * 5	20 to e
/=	f /= 3	f = f / 3	2 to f
%=	g %= 9	g = g % 9	3 to g

a = 10, b = 20, c = 10, d = 1, e = 20, f = 2, g = 3

Operator	Statement	Assign
? :	(c >> e) ? a = c : a = e	15 to a
? :	(a == b) ? c++ : d++	2 to d
<<	if (d << g) { d+=g; }	5 to d
!=	if (a != (c+d)) { b = a; }	None
||	if ((a==c) || (c!=d)) { f+=2; }	4 to f
||	if ((a!=c) || (c==d)) { d+=5; }	10 to d
&&	if ((a << c) && (c == d)) { a+=5; }	None
&&	if ((a >> c) && (c == d)) { a+=5; }	20 to a

Questions