Data Structures and Program Design in C

Topic 2 : Types, Operators, and Expressions

Dr Manjusri Wickramasinghe

Outline

- Char Types
- Operators
  - Arithmetic Operators
  - Relational and Logical Operators
  - Bitwise Operators
  - Assignment Operators
  - Operator Precedence
- Constants
- Enumerations
- Type Conversions

Char Types ...(1)

- Char types or character types is capable of holding a single ASCII character.

- Char type takes one (1) byte of memory.

- American Standard Code for Information Interchange (ASCII) encodes 128 characters using seven (7) data bits.

Char Types ...(2)

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

UCSC

Assignment and Equal To Operator

- In programming languages the equals (‘=‘) symbol represents the assignment operator.

<L.H.S> = <R.H.S>

- For example, `int a = 5` assigns value five to the variable a.

- To check equivalence the double equal (‘==‘) is used. This is a relational operator.

Arithmetic Operators

- There are two types of arithmetic operators in the C language
  - Binary operators
    - Addition (+), Subtraction (-), Multiplication (*), Division (/), and Modulo (%)
  - Unary operators
    - Increment(++), Decrement(--), Unary Plus (+), Unary Minus (-)

- What is integer division?

Relational and Logic Operators

- These operators, when placed in expressions, provide the logical truth or falsity of the expression.

- Relational operators are greater than (>), greater than or equal (>=), less than (<), equal To (==), not equal to (!=) and less than or equal (<=).

- Logical operators are:
  - AND operator (&&)
  - OR operator (II)

Bitwise Operators

- C provides six (6) operators for bit manipulation.
  - & - bitwise AND
  - | - bitwise OR
  - ^ - bitwise XOR
  - << - left shift
  - >> - right shift
  - ~ - complement of the number

- These operators can only be applied to char, short, int, and long.

Operator Precedence …(1)

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Operator Precedence …(2)

Find the values of variables and the results

Example 1

```
int x = 4, y = 5, z = 2;
int result = x++ * --y + z++ / x;
```

Example 2

```
int a = 3, b = 4, c = 2;
int result = ++a * b-- + c++ - a;
```

Example 3

```
int x = 5, y = 10, z = 4;
int result = x++ & y | z++ ^ --x;
```

Constants …(1)

- Constants are read-only variables whose values cannot be changed once declared.

- Constants are defined in two ways:
  - `const` keyword
  - `#define` preprocessor directive

- Constant can be defined as:

```
const <data_type> <var_name> = value;
```

Constants …(2)

- Constants using #define are macros that behave like constants.
- These constants are handled by the preprocessor.

```
#define <const_name> <value>
```

- Constants are immutable.
- Constants can be named as per the variable naming conventions. However, it is a common practice to use uppercase when naming constants.

Enumerations

- Enumerations are lists that contain constant integer values.

enum <enum_name> { <options>};

- The list of values has to be unique within the scope.

- What is a size of an enum?

Typedef

- Typedef keyword can be used to define an alias for existing data types and enumerations.

typedef <data_type/enum> <new_name>;

- E.g.
  - Enums
    - ```
      typedef enum days
      {
          SUN, MON, TUE, WED, THU, FRI, SAT
      }TEST;
      ```
    - ```
      typedef int Integer;
      ```

Type Conversions …(1)

- When the operator has operands of different types, they are converted to a common type.

- This is also known as *typecasting*.

- Two types of conversions
  - Automatic
  - Explicit

Type Conversions …(2)

- Automatic (implicit) conversions are when narrow operands are converted into wide ones without losing information.
  - E.g. int -> float, float -> double

- Type conversions that convert from wider operands to narrow operands are not prohibited or illegal.
  - Generates a compiler warning
  - Requires explicit type conversion
  - Standard format

```
<type-name> <expression>
```

Type Conversions …(3)

- Why are type conversions important?
  - Type safety
  - Enhancing code readability
  - Improved data manipulation

- Problems with type conversions
  - Loss of precision
  - Overflow or Underflow
  - Unexpected behavior

Mathematical Functions

- The header `<math.h>` declares the mathematical functions and headers.
  - Trigonometric functions - the trigonometric functions are expressed as radians

- To compile source code that uses the `math.h` header requires the use of `-lm`.

Time Functions

- The header `<time.h>` declares types and methods to manipulate date and time.

- `clock()` – returns the process time used by the program since the beginning of the execution of the program.
  - Dividing the returned value by `CLOCKS_PER_SEC` will return the time for execution in seconds.

Questions?