Introduction To C Programming

Lesson 02

Data types, operands, expressions & precedence

Objectives

- Understand C data types
- Understand C operands
- Understand C expressions
- Understand operator precedence

Memory Concepts

- Computer memory can be thought of as a sequence of bytes
  - If you've used a spreadsheet, you can think of these bytes as cells in the spreadsheet.
- What values are in these bytes?
  - Before our program runs?
  - After our program is loaded into memory?

Memory Use

- The interpretation of that memory is up to us
- What do we do?
  - Group 1, 2, 4, or 8 contiguous bytes together
  - Interpret the bits in these bytes as integers, characters, or floating point numbers
    - Each type interprets the bits differently
- C helps us do this by providing data types

Data Types

- Fundamental data types
  - Integer types
  - char type (really a one byte integer)
  - Floating point types

Integers

- Whole numbers (no fractional part)
- Must be specified without a decimal
- Usually range from -32768 to 32767
- Actual range is machine and vendor dependent

10

1234

-5000

Floating Point Numbers

- Computer representation of a real number
- Requires the decimal
- Can be written using scientific notation

10.

1234.567

1.23E23

Data Type Keywords

- Keywords used to describe data types:

  char

  int

  short

  long

  float

  double

  unsigned

  signed

Integer Types

- There are 8 integer types:

  char, unsigned char

  int, unsigned int

  short int (usually just ‘short’), unsigned short

  long int (usually just ‘long’), unsigned long

int

- Size is machine dependent:
  - 16 bit machine: 2 bytes, -32768 to 32767
  - 32 bit machine: 4 bytes, -2,147,483,648 to 2,147,483,647
- High bit is the sign bit unless int is declared as unsigned
  - Consequences (e.g., int, unsigned int)
    - signed int range (-32768 to 32767)
    - unsigned int range (0 to 65535)

char

- char is an *integer* type.
- Can be either signed or unsigned
- 16 bit machine: 1 byte, signed range is -128 to 127, unsigned range is 0 to 255
- 32 bit machine: 1 byte, signed range is -128 to 127, unsigned range is 0 to 255

short int

- Size is machine and vendor dependent
- Guaranteed to be no bigger than an int
- 16 bit machine: 2 bytes, -32,768 to 32,767
- 32 bit machine: 2 bytes, -32,768 to 32,767

long int

- Size is machine and vendor dependent
- Guaranteed to be no smaller than an int
- 16 bit machine: 4 bytes, -2,147,483,648 to 2,147,483,647
- 32 bit machine: 4 bytes, -2,147,483,648 to 2,147,483,647

Floating Point Types

- Size is vendor dependent
- Two types:

  float

  double

- On some systems the range of a double is the same as a float, but the precision is increased
- Some implementations may allow ‘long double’

Floating Point Ranges

- float:

  16 bit machine: 4 bytes

  range: + or - -3.4E-38, + or - 3.4E+38

  32 bit machine: 4 bytes, range machine dependent

- double:

  16 bit machine: 8 bytes

  range: + or - -1.7E-308, + or - 1.7E+308

  32 bit machine: 8 bytes (range machine dependent)

sizeof(arg)

- Returns the number of bytes in its argument.
  - Can specify a *type name* or *variable name* as arg
- Looks like a function but it’s an operator
- Results are machine dependent
- Examples
  - sizeof(int) - evaluates to 4 on 4 byte machine
  - sizeof(char) - evaluates to 1
  - float value; sizeof(value) - returns number of bytes used to store a float on machine

limits.h and float.h

- Standard says:
  - sizeof(short) <= sizeof(int) <= sizeof(long)
  - sizeof(float) <= sizeof(double) <= sizeof(long double)
- In a portable C program, how do you know how big or small of a number you can use?
- Limits.h and float.h are ANSI standard header files which contain constants for common limits on integer and floating point number

Constants

- Specify an unchanging value
- Have a type
- Four types:
  - *Integer*
  - *Floating Point*
  - *Character*
  - *String*

Constants (example)

- ***Integer***: 0 32000 -123 (if too big will make it a long; specify a long by putting the letter ‘L’ after the number: 21567L)
  - Can specify octal constant by prefixing with 0
    - Example: 034
  - Can specify hex constant by prefixing with 0x
    - Example: 0xffff
- ***Floating point***: 3.14159 -123.05 1.5E-02 (by default, it is a double; put an ‘F’ after the number to make it a float)

Constants (character examples)

- ***Character***: ‘A’ ‘q’ ‘$’ ‘\723’ (non-printable characters can be written this way, eg., ‘\015’ for carriage return)
- ***String***: “Hello world!”

Variables

- Each variable has a name, type, and value
- A symbolic reference to a memory location holding program data, subject to change
- Must be declared before they are used
- Declaration statement:
  - <type> <identifier>
    - type - one of the data types
    - identifier - begin with letter or underscore; use letters, numbers or underscores; can’t be keyword; only first 31 chars are ‘significant’

Variables (examples)

```
int    num;
char   character;
float  x;
double sum;
```

- Multiple variables of the same type may be declared on one line:

```
int    num1, num2, num3;
char   in_char, out_char;
float  x, y, z;
```

Variables

- Must be declared at the beginning of a function or code block
- Un-initialized variables have garbage values
- Occupy space on the ‘stack’

Operators

- Specify an action to perform
- Five basic types:

  sizeof

  Arithmetic

  Assignment

  Relational

  Logical

Operators

- C has a rich set of operators
- Style: It’s usually better to leave a space on each side of the operator:

```
     num = x * y


    num=x*y
```

Arithmetic Operators

- + (addition)
- - (subtraction)
- * (multiplication)
- / (division)
- % (modulo or ‘remainder’)

  5 / 2.0 yields 2.5

  5 / 2 yields 2

  5 % 2.0 undefined

  5 % 2 yields 1

Assignment Operators

- = (assignment)

```
num = 3
```

- Makes things like this possible:

```
x = y = z = 0
```

- Evaluated as:

```
x = (y = (z = 0))
```

Compound Arithmetic Operators

- +=
- -=
- *=
- /=
- %=
- Treated as: lvalue = lvalue <op> <expr>

```
num = num + 100;
num += 100;
```

Initialization During Declaration

- Variables may have values assigned to them at declaration time

  int num = 3;

- Multiple declarations can be put on one line

  int   num1 = 3, num2 = 1, num3 = 0;

- Mix and match: int x = 3, y, z = 1;
- The keyword ***`const`*** tells the compiler that the variable can't be modified:

  const int num = 3;

Relational Operators

- Relational operators are binary operators used to compare one operand to another
- < (less than)
- > (greater than)
- <= (less than or equal)
- >= (greater than or equal)
- == (equal to)
- != (not equal to)

Relational Operators

- Given:

```
int     x = 3, y = 5;


y > x          yields 1
x == y         yields 0
x < y          yields 1
x != 3         yields 0
```

Logical Operators

- Relational expressions can be combined using logical operators
- && (logical AND)
- || (logical OR)
- ! (logical NOT)
- Logical expressions are evaluated left to right. As soon as an element is found which invalidates the expression as a whole, the evaluation stops.

Logical Operators

- Given:

```
int    x = 0, y = 5;
-3 < x && y > 2            yields 1
-4 > x || y == 5            yields 1
x != 0 && 20 / x > 5     yields 0
```

- The second expression in example 3 is evaluated only if x is non-zero

Expressions

- An *expression* in C is a combination of zero or more operators with one or more operands
  - An operand is a variable or constant
- All C expressions have a value. The value of an assignment is the value being assigned
  - x = 3; /* value of x is 3; value of expression x = 3 is 3 */
  - x == 1; /* value is 1 if x is 1; 0 otherwise */
  - y = x = 3; /* value of y is 3 since value of x = 3 is 3

Increment & Decrement

- A common need is to increment a variable by one or decrement by one:

```
x = x + 1 (could also be:  x += 1)
y = y - 1  (could also be: y -= 1)
```

- Increment:

```
++x    or    x++
```

- Decrement:

```
--y    or    y--
```

Increment & Decrement

- Consider:

```
int     a = 2, b = 3, c = 0;

c = a * ++b;       (c is assigned 8)

b = 3;

c = a * b++;       (c is assigned 6)
```

- What are advantages and disadvantages of writing code such as the above?

Precedence

Op. Type	Precedence	Operators	Associativity
primary	15	() []	l to r
unary	14	! ++ -- -	r to l
		(type) * & sizeof()	
arithmetic	13	* / %	l to r
	12	+ -	l to r
relational	10	< <= > >=	l to r
	9	== !=	l to r

Precedence

Op. Type	Precedence	Operators	Associativity
logical	5	&&	l to r
	4	||	l to r
assign.	2	= += -= *= /= %= etc.	r to l
comma	1	,	l to r

Expression Evaluation

- Usually follows rules of algebra
- Constants and variables of different types are converted to a common type:
  - All char’s and short’s converted to int; float converted to double
  - If one operand in a pair is a long double, the other is converted to long double

    Else, if one operand is a double, other converted to double

Expression Evaluation

Else, if one operand is a long, the other is converted to long

Else, if one operand is unsigned, the other is converted to unsigned

Else, both operands are int’s

Expression Evaluation

- You can force an operand to be a certain type by means of a *cast*:

(type) expression

Example:

To ensure that x / 2 evaluates to float to prevent truncation of the remainder:

(float) x / 2

(Or just: x / 2.0)

printf() Format Codes

Code	Format
%c	a single character
%d	decimal integer
%i	decimal integer
%e	scientific notation
%f	decimal notation
%g	use the shortest of %e or %f
%o	octal integer
%s	string of characters
%u	unsigned decimal integer

printf() Format Codes

- The letter ‘l’ means print a long data type. %ld prints a long int, %lf prints a long float (double)
- ‘h’ means print a short. %hd prints a short int.
- %x.yf prints a field width and precision

  %10.4f   (10 chars, 4 after decimal point)

  %.2f   (2 chars after decimal)