MPATE-GE 2618:
C Programming for Music Technology

Variables, Operators and Expressions

Reading

- *Programming in C*, Chapter 3

Versions of C

- Many standardized versions of C
  - Initial development of C took place in the late 60s/early 70s
  - K&R C (1978)
  - ANSI C or C89 (1989)
  - ISO C or C90 (1990) – mostly the same as ANSI C
  - C99 (1999)
  - C11 (2011)
- We will primarily use ANSI C, also some aspects of C99
- gcc has most C99 features implemented

C Reference URL

- A good reference for C language
  - –http://en.cppreference.com/w/c

C is a Strongly Typed language

- ALL variables must be declared.
  - Forces you to be aware of variable type
- Compiler may complain when expressions mix variable types
  - This is a hint that something could go wrong

```
long i, j;
float x;
i = j; // OK
i = x; // Will result in loss of fractional part
```

A Few Definitions

- Variable (`a, b, c`)
  - Used to store a value
- Operator (`a + b`)
  - Used to indicate calculations
- Statements
  - A C-language “sentence”

Variable Names

- Some rules on naming variables:
  - Must begin with a letter or underscore (_)
  - Can't begin with a number
  - Can't contain special ($) or operator (+, - , *, /) characters
  - Can't have spaces (the compiler interprets this as two variables)
  - Can't be reserved words like `int, float`
- Variable names are case sensitive
  - `a` and `A` are two different variables

Example Variable Names

- Simple names

```
int i; //loop counter variable
float x; //real number
```

- Underscore enhances readability

```
int parts_count;
```

- “Camel case” enhances readability

```
int partsCount;
```

Statements and variables

- Statements are followed by a semicolon (;)
- A variable is *declared*

  <type> <variablename>;

  ```
  int num;   // declaration
  num = 5;   // assignment
  ```

- It can be initialized on declaration

  ```
  int num = 5;
  ```

- Basic types in C

  `char, short, int, long, float, double, bool`

  http://en.cppreference.com/w/c/language/type

Integers

Types: `int`

- Decimal values – base 10

```
int n = 123; // Decimal value
```

  - Print using %i or %d

- Octal values – base 8 (Seldom used!)

```
int n = o177; // Octal value
```

  - Print using `%o` (lowercase o)

- Hexadecimal values – base 16

```
int n = 0x24; // decimal value 36
```

  - Print using %x or 0x%02x

Binary notation

- Binary = base 2
- Decimal = base 10
- Octal = base 8 (rarely used)
- Hexadecimal = base 16 (widely used)
  - –Useful because:
    - 1 byte = 8 bits
    - 4 bits = 1 hex digits
    - 2 hex digits = 1 byte
- Examples
  - Base 10

    $357 = (10^2 * 3) + (10^1 * 5) + (10^0 * 7)$
  - Base 2

    $1011 = (2^3 * 1) + (2^2 * 0) + (2^1 * 1) + (2^0 * 1)$
  - Base 16

    A = 10; B = 11; C = 12; D = 13; E = 14; F = 15

    $0xFA13 = (16^3 * 15) + (16^2 * 10) + (16^1 * 1) + (16^0 * 3)$

Binary	Decimal	Hex
0000 0001	01	01
0000 0010	02	02
0000 0011	03	03
0000 0100	04	04
0000 0101	05	05
0000 0110	06	06
0000 0111	07	07
0000 1000	08	08
0000 1001	09	09
0000 1010	10	0A
0000 1011	11	0B
0000 1100	12	0C
0000 1101	13	0D
0000 1110	14	0E
0000 1111	15	0F
0001 0000	16	10
0001 0001	17	11

Integers: Two's Complement Notation

- Two's complement is a "computer hardware-friendly" representation of positive and negative integer values
- Representation:
  - Positive: MSB is 0
  - Negative: MSB is 1
  - Negatives are "backwards"
- A = B+C is simple binary addition, discard any carry
- But not symmetric in range

Binary	Decimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

Integer Modifiers

- Unsigned (only positive)
- Short (16 bits -- like many WAV files)
- Long (32 bit)
- Long long (64 bits – like modern CPUs)

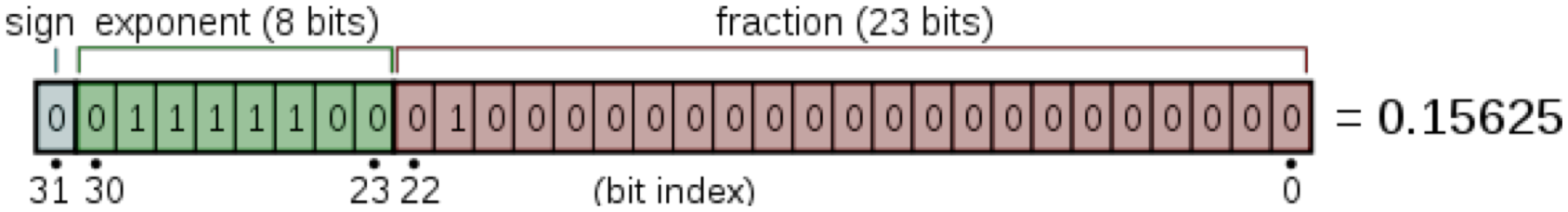
Float

Types: `float` and `double`

- `float` is used to store values with decimal places
- `double` has a larger range than `float` and is more precise
- `printf` format specifiers for doubles and floats: `%f`, `%e`, `%g`
  `%.5f`, for example, displays 5 decimal places
- (Never test floats and doubles for equality!)

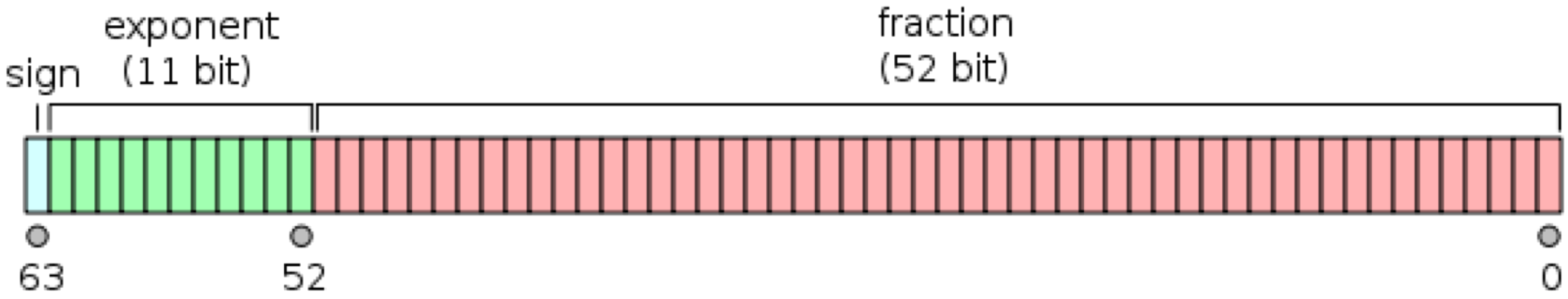
Real Numbers: Floating Point

- In early computing, each machine had its own arithmetic unit architecture!
- IEEE 754 is a standard for floating point numbers
  - Requires a 32-bit word
  - Range is +/- 3.4 x 10^{38}



Float, Double

- Float is single-precision floating point
  - –32 bits of storage
  - –7 decimal digits of precision
- Double is double-precision floating point
  - –64 bits of storage
  - –15 decimal digits of precision



Char

Characters: ASCII encoding

- ASCII: American Standard Code for Information Interchange
  - An industry standard which assigns an 7-bit number to the "printable characters"
  - 7 bits stored in 8-bit variable (i.e. one byte)
  - e.g. the letter 'A' = 65, 'B' = 66, etc.
  - Convert the letters 'A' (decimal 65) and 'm' (decimal 109) to binary and hexadecimal:

'A' = decimal 65 = binary 0100 0001 = hex 0x41

'm' = decimal 109 = binary 0110 1101 = hex 0x6D

ASCII Table

- ASCII codes organized as 4 cols of 32 rows
  - Col 1 – control characters (e.g. '\n', '\r')
  - Col 2 – Special characters and numerals
  - Col 3 – Upper case letters
  - Col 3 – Lower case letters
- MSB of 8-bit ASCII value is always `0`
- Note that 'a' = 'A' + 32
  - A good trick: 'x' - 'a' + 'A' = 'X'
- See asciitable.com
- Also en.cppreference.com/w/c/language/ascii

_Bool or bool

- `_Bool` is built into c99
- From c99, bool is defined by using

```
#include <stdbool.h>

bool flag; //Boolean variable
```

Types: `char` and `bool`

- `char` is a single character and is denoted by single quotes (double quotes are used for strings); use `%c` to print

```
char ch = 'c';
char ch2 = '\n'; // newline character
```

- `bool` has only two states, 0 or 1
  The type `bool` as well as `true` and `false` are defined in the include header file `<stdbool.h>`

```
bool condition = false;
```

  - Print them as you would integers (using `%i` or `%d`)
  - NOTE: `bool` is only available in C99

```
true == 1
false == 0
```

Basic types: review

- Four basic types:
  - char
  - int
  - float/double
  - bool (make sure to add #include <stdbool.h>)

More on type modifiers
`long, short, unsigned`

- Size of `int` depends on the platform
  - `int` is the most efficient length for a platform
  - Typically this is the length of a CPU register
- You can put `long, short`, and `unsigned` before the `int` type specifier :

```
short int i;
short i; /* equivalent to the line above*/
long int i;
long i; /* equivalent to the line above*/
long long int i;
long long i; /* equivalent to the line above*/
unsigned short int i;
unsigned short i; /* equivalent to the line above*/
```

- http://en.cppreference.com/w/cpp/language/types

Literals

- Literals are C tokens that represent values
- Use them in assignments

```
float x = 4 / 8f; //f forces float value
```

- Use them in printf() statements

```
\n //newline
\t //tab
printf("1\tHello\n2\tWorld\n");
```

- http://en.cppreference.com/w/cpp/language/integer_literal
- http://en.cppreference.com/w/cpp/language/floating_literal

Statements

- C-Language statement is
  - –Something that can be evaluated
  - –Always is terminated by semicolon “**;**”

Operators

- Basic arithmetic operators
  - `+` Add
  - `-` Subtract
  - `*` Multiply
  - `/` Divide
- Assignment operators
  - `=`

Operators

- Arithmetic *operators*:
  - + for addition, - for subtraction, * for multiplication, / for division
- *Precedence*:
  - some operators have higher priority
    a * b + c * d is the same as (a * b) + (c * d)
  - When in doubt, use parentheses
- White space not necessary but recommended for readability
- http://en.cppreference.com/w/c/language/operator_precedence

Operators continued

- The unary minus (–) operator negates a value it precedes and has higher precedence than all the arithmetic operators
  (example: `c = -a * b;`)
- The modulus operator % gives the remainder of the first value divided by the second value

```
int a, b, c;
a = 10;
b = 4;
c = a % b; // results in c being 2
```

Relational operators

- Relational operators:

==	equal to
!=	not equal to
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to

Arithmetic Expressions

- Variables and operators are used to build expressions
- Unary expression

```
a = -b;
```

- Binary expression

```
a=b+c;          //white space is not necessary
a = b + c;      //but makes it easier to read
a = b * c;
```

Order of Operations

- Precedence of operators is just like in algebra
  - –Left to right
  - –Multiplication and division over addition and subtraction
- More complex expression

```
a = b+c / d+e;
```

- Evaluated as

```
a = b + (c / d) + e
```

- Use parenthesis to force explicit grouping!

```
c = (a + b) / (c + d);
```

Modulus Operator

- Modulus operator is %
  - –Arguments are integer
  - `a % b` is the remainder when a is divided by b

```
a = 12 % 5; // a is 2
a = 10 % 5; // a is 0
```

Boolean expressions

- Expressions can be true or false

```
int a = 4, b = 10;
```

`a == b` is false

`a != b` is true

`a > b` is false

`b > a` is true

`b - a > a * 2` is false

Increment and Decrement Operators

- `++` and `--` are increment and decrement operators

  `++i` or `i++` is equivalent to `i = i + 1;`

  `--i` or `i--` is equivalent to `i = i - 1;`

- Order of `++` and use:

  `i = ++j;` increments BEFORE assignment

  `i = j++;` increments AFTER assignment

- Operators with `=`

  `i += 2` is equivalent to `i = i + 2;`

  `i -= 2` is equivalent to `i = i – 2;`

  `i *= 5` is equivalent to `i = i * 5;`

  `i /= 3` is equivalent to `i = i / 3;`

Cast Operator

- Explicitly converts one type to another

```
(type) variable
```

- Example

```
int a;
float x = 3.14159;
a = (int) x; //a is 3
```

Implicit Cast

- Expressions always convert to “most inclusive” variable type *under order of parsing*.

```
int i = 10, j;
float x = 0.5, y;
j = 5*i; //j is 50
y = 5*x; //y is 2.5
y = i*x; //y is 5.0
y = i/3 * x; //x is 1.5
```

Assignment to Integer

```
int i;
float x = 1.75;
i = x; //value of i is 1
```

- Integer is always “largest whole number in”

Rounding to Integer

```
#include <math.h>
int i;
float x = 1.75;
i = round(x); //value of i is 2
```

- `round(x)` returns integer value nearest to `x`
- Can also be done by adding 0.5 and using integer cast

```
i = (int)(x + 0.5);
```

Cast and Expressions

- "Fix" mixed type expression

```
int i = 10, j;
float x = 0.5, y;
y = (float)i/3 * x; //x is 1.667
y = i/3.0 * x; //x is 1.667
```

Examples: Integer and floating point conversions and casting

```
float f1 = 123.125, f2;
int i1, i2 = -150;
i1 = f1;          // i1 set to 123
f1 = i2;          // f1 set to –150.000000
f1 = i2 / 100;    // f1 set to –1.000000
f2 = i2 / 100.0; // f2 set to –1.500000
f2 = (float)i2 / 100; // f2 set to –1.500000
```

- Variables may be *promoted* in order to evaluate an expression
- You can explicitly *cast* (e.g `(float)`) to force the desired result
- `(float)` only modifies the variable immediately to the right

```
(float)i2/100; // is -1.5
(float)(i2/100); // is -1.0
```

Code Comments

- Original C comments

```
/* This is a comment */

/* This is a longer comment
   with more to say */
```

- C++ comments are part of C99

```
// A single-line comment
```

Review

- Variable Types

  `char, int, float, double, bool`

- Variable modifiers

  `unsigned, short, long, long long`

- Basic operators

  `= + - * /`

- Relational operators

  `> >= < <= !=`

- Increment operators and = modifiers

  `-- ++ -= += *= /=`

- Promotion and Cast

  `(float)`