GNG1106
Fundamentals of Engineering Computation

Instructor: **Hitham Jleed**

University of Ottawa

Fall 2023 ~

Outline

1. Conversion Specifiers
2. More on Types
3. Arithmetics
4. Math Library
5. Cast Operator
6. Symbolic Constant

Conversion Specifiers for `scanf` and `printf`

Type	printf	scanf
short	%d, %i, %hd, %hi	%hd, %hi
int	%d, %i	%d, %i
long	%ld, %li	%ld, %li
unsigned short	%hu	%hu
unsigned int	%u	%u
unsigned long	%lu	%lu
float	%f, %e, %E, %g, %G	%f, %e, %E, %g, %G
double	%f, %e, %E, %g, %G	%lf, %le, %lE, %lg, %lG
long double	%Lf, %Le, %LE, %Lg, %LG	%Lf, %Le, %LE, %Lg, %LG

- Conversion specifiers for `scanf` and for `printf` are not always the same (check type `double`).
- "%e" and "%E" are for scientific notations: 1.3e3 and 1.3E3
- You do not need to remember this entire table. Just the common ones (those for int, float, double) would be good enough.

Width and Precision Formatting in `printf`

By running this code, can you figure out what `%x.yf` does?

```
#include <stdio.h>
int main()
{
  printf("%5.3f\n", 3.14);
  printf("%7.3f\n", 3.14);
  printf("%7.3f\n", 3.14345);
  printf("%7.3f\n", 3.14365);
  printf("%5.3f\n", 321.14);
  return 0;
}
```

- `x` specifies the maximum width of the number; `y` specifies the precision (the number of digits after the decimal).
- It is good enough to know such things exist; no need to memorize them. Google them as you need.

Outline

1. Conversion Specifiers
2. More on Types
3. Arithmetics
4. Math Library
5. Cast Operator
6. Symbolic Constant

Binary Representation

- Data and programs are both represented in the binary format inside a computer.
- The binary representation uses strings consisting of 0 and 1.
- A single digit taking value 0 or 1 is called a bit.
- A string of 8 bits is called a byte.
- How many different configures can a byte take?
    - counting: 00000000, 00000001, 00000010, 00000011, ... , 11111111
    - answer: $2^8 = 256$.
- Byte is the basic unit of representation in a computer

- A set of rules are built-in to convert between numbers and bytes.
- Converting numbers to bytes is often called “encoding”.
- Encoding rules are machine-dependent.
- Two main classes of encoding rules:
    - Fixed-point encoding
    - Floating-point encoding

Fixed-Point Encoding

- It essentially converts between bytes and integers.
- Ideas:
  - 2 bits can represent 4 integers,
    - e.g. 00 → 0, 01 → 1, 10 → 2, 11 → 3
      (such an encoding is sometimes called "natural binary encoding")
      "*There are 10 kinds of people in this world: those who know binary and those who do not*".
    - e.g. 00 → −2, 01 → −1, 10 → 0, 11 → 1
  - One byte can represent $2^8 = 256$ integers, e.g. in the range of $\{0, 1, \ldots, 255\}$ or $\{-128, -127, \ldots, 127\}$
  - Two bytes can represent $2^{16} = 65,536$ integers, e.g., in the range of $\{0, 1, \ldots, 65535\}$, or $\{-32768, -32766, \ldots, 32767\}$

- In standard C, fixed-point types are `char`, `short`, `int`, `long`, which are 1-byte, 2-byte, 4-byte and 8-byte encoding of integers. The range of their encoded integers include negative numbers, 0, and positve numbers. (The `char` type is mostly used to represent characters, which we will discuss later)
- Each of these type also has an `unsigned` version (`unsigned char` `unsigned short`, `unsigned int`, `unsigned long`), each encoding only non-negative numbers.
- In C, numbers (formally, "literal values") without including the decimal point, such as "31", "-10", "0", etc. are interpreted as fixed-point numbers.

A code to check the sizes of types

```
#include <stdio.h>
int main()
{
    printf("Size of char is %ld bytes\n", sizeof(char));
    printf("Size of short is %ld bytes\n", sizeof(short));
    printf("Size of int is %ld bytes\n", sizeof(int));
    printf("Size of long is %ld bytes\n", sizeof(long));
    printf("Sizes of unsigned long is %ld bytes\n",
sizeof(unsigned long));
    return 0;
}
```

Floating-Point Encoding

- It essentially converts between bytes and rational numbers
- More complex encoding methods
- Encoding with more bytes gives larger range and higher precision
- In standard C, floating-point types are `float` (4 bytes), `double` (8 bytes), `long double` (16 bytes).
- In C, numbers (formally, "literal values") including the decimal point, such as "31.0", "-10.135", ".567" , "1.32e3", are interpreted as floating-point numbers.
- Note: "1.32e13" represents 1.32×10^{13}.

Outline

1. Conversion Specifiers
2. More on Types
3. Arithmetics
4. Math Library
5. Cast Operator
6. Symbolic Constant

Operation	C Binary Operator	Example Expression
addition	+	a + b
subtraction	-	a - b
multiplication	*	a * b
division	/	a / b
modulus	%	a % b

- The modulus operator `%` applies to integers only. The expression `a % b` gives the remainder of dividing `a` by `b`, for example, `23 % 5` gives value `3`.
- Be very careful with the division operator:
  - If either `a` or `b` is floating-point, then `a/b` is the regular division. For example, `3.0/5` gives 0.6.
  - if both `a` and `b` are fixed-point, then `a/b` is gives the quotient of dividing `a` by `b`. For example, `3/5` gives 0 and `8/5` gives 1.

Operator Precedence

- Contents of parentheses `(...)` are evaluated first.
  - If there are many levels of parentheses, then the innermost pair is evaluated first; the next innermost pair is evaluated second ...
  - If there are many pairs of parentheses on the same level then they are evaluated left to right.
- Unary operator (such as negation) is evaluated next.
- Multiplications, divisions and moduli are evaluated next.
  - If there are many, they are evaluated from left to right.
- Additions and subtractions are evaluated last.
  - If there are many, they are evaluated left to right.

Implicit Rule of Promotion

- An expression that contains variables of several different types will be converted according to the “implicit rule of promotion”.
- The basic idea/rationale is the following.
    - A high-range high-precision type can store the same information as a low-range low-precision type. For example, a float-typed value can also be represented as a double-typed value without losing information.
    - When a low-byte-number variable interacts with a high-byte-number variable via an arithmetic operation, the result is automatically represented using the high-byte-number representation.
- The same rule of promotion applies when a low-range low-precision value is assigned to a high-range high-precision variable, in which case, the value is automatically promoted to the high-byte-number representation.

Play with this kind of code and investigate!

```
#include <stdio.h>
int main()
{
     long double x;
     printf("1 is represented by %ld bytes\n", sizeof(1));
     printf("5.0 is represented by %ld bytes\n", sizeof(5.0));
     printf("1+5.0 is represented by %ld bytes\n", sizeof(1+5.0));
     x=1;
     printf("x is represented by %ld bytes\n", sizeof(x));
     printf("(1+5.0)*x is represented by %ld bytes\n", sizeof((1+5.0)*x));
     return 0;
}
```

The result:

```
1 is represented by 4 bytes
5.0 is represented by 8 bytes
1+5.0 is represented by 8 bytes
x is represented by 16 bytes
(1+5.0)*x is represented by 16 bytes
```

Examples

Expression	Value
`3/4`	`0`
`3.0/4`	`0.75`
`0.3+3/4`	`0.3`
`0.3+3/4.0`	`1.05`
`(0.3+3)/4`	`0.825`

Outline

1. Conversion Specifiers
2. More on Types
3. Arithmetics
4. Math Library
5. Cast Operator
6. Symbolic Constant

Standard Math Library

- There is a rich library of built-in standard math functions we can use in C.
- To use these functions, we need to use pre-processor directive `#include <math.h>`
- Most of these functions have their input variables declared as `double`, and also have return type `double`.

Math Function	C function
$\sqrt{x}$	`sqrt(x)`
x^y	`pow(x,y)`
e^x	`exp(x)`
$\log_e x$	`log(x)`
$\log_{10} x$	`log10(x)`
$\|x\|$	`fabs(x)`
$\sin(x)$	`sin(x)`
$\cos(x)$	`cos(x)`
$\tan(x)$	`tan(x)`
$\arcsin(x)$	`asin(x)`
. $\arccos(x)$	`acos(x)`
$\arctan(x)$	`atan(x)`
$\sinh(x)$	`sinh(x)`
$\cosh(x)$	`cosh(x)`
$\tanh(x)$	`tanh(x)`

Note

All functions taking an angle as input or output use radian (rather than degree) as the unit for angle.

- The function `ceil(x)` rounds `x` to the smallest integer that is greater than or equal to `x`.
    - “ceil” stands for “ceiling”
- The function `floor(x)` rounds `x` to the largest integer that is less than or equal to `x`.
- Note that these two functions return an integer value but with type `double`.

Outline

1 Conversion Specifiers

2 More on Types

3 Arithmetics

4 Math Library

5 Cast Operator

6 Symbolic Constant

The Cast Operator

- A cast operator is a unary operator, i.e., acting only on one value/variable.
    - An unary operator has higher precedence than binary operators.
- A cast operator forces a value/variable to be interpreted as a different type ("casts" the value/variable to a different type).
- A cast operator is in the form of "`(aType)`", and used in expressions like "`(aType) aValue`", where `aValue` is a value or a variable, and `aType` is the type to which `aValue` is forced.

    -
      ```
      int a=1, b=2;
      float c;
      c=(float)a/b;
      ```
    - In this example, the value of `c` will be 0.5.

Caution with Casting

- It is fine to cast a low-precision (or low-range) variable to a high-precision (or high-range, resp.) type.
- The opposite is however discouraged.
    - It results in loss of information and even wrong number!
    - Do it only if that is really what you want.

Outline

1 Conversion Specifiers

2 More on Types

3 Arithmetics

4 Math Library

5 Cast Operator

6 Symbolic Constant

Symbolic Constant

- Symbolic constants are defined using the pre-processor directive `define`, for example,
  - `#define NBR_OF_STUDENTS 455`
    which instructs the compiler to replace every occurrence of string `NBR_OF_STUDENTS` with `455` in the code before compilation.
  - `#define PI 3.141593`
    which instructs the compiler to replace every occurrence of string `PI` with `3.141593` in the code before compilation.
- Recall that pre-processor directives do not require a ";" at the end.

Highlight

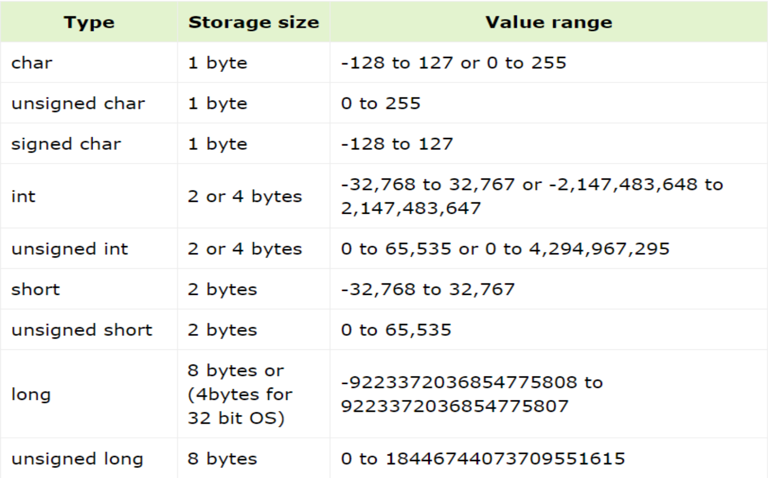
The use of symbolic constants can eliminate "magic numbers" and makes the program more readable and easier to maintain.

Coding Demonstration

https://github.com/hjleed/GNG1106_Archive/tree/main/week3_codes

Integer Types

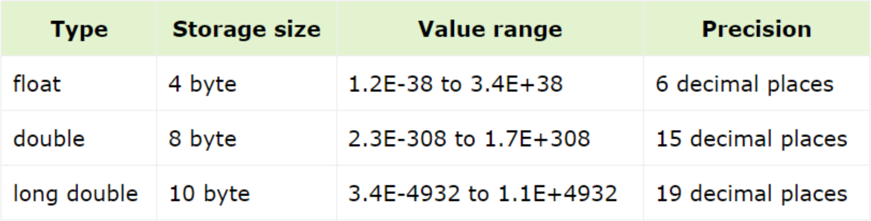
The following table provides the details of standard integer types with their storage sizes and value ranges.



Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	8 bytes or (4bytes for 32 bit OS)	-9223372036854775808 to 9223372036854775807
unsigned long	8 bytes	0 to 18446744073709551615

Floating-Point Types

The following table provide the details of standard floating-point types with storage sizes and value ranges and their precision



Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places