



Course organization

- Course introduction (Week 1)
 - Code editor: Emacs
- Part I: Introduction to C programming language (Week 1 - 12)
 - Chapter 1: Overall Introduction (Week 1-4)
 - C
 - Unix/Linux
 - **Chapter 2: Types, operators and expressions (Week 4)**
 - Chapter 3: Control flow (Week 6)
 - Chapter 4: Functions and program structure (Week 7-8)
 - Chapter 5: Pointers and arrays (Week 9)
 - Chapter 6: Structures (Week 10)
 - Chapter 7: Input and Output (Week 11)
- Part II: Skills others than programming languages (Week 12- 14)
 - Debugging tools (Week 12-13)
 - Keeping projects documented and manageable (Week 14)
 - Source code managing (Week 14)
- Part III: Reports from the battle field (student forum) (Week 15 – 16)



Variable (变量)

- `int i, a = 0;`



Assignment (赋值)

- `i = 1; i = a;`
- `i == 1; /* this is not an assignment */`



Expression (表达式)

- Contains variables and operators
- Example:
 - `i = (i + 2) * 5;`



上海交通大学
SHANGHAI JIAO TONG UNIVERSITY



Chapter 2. Types, operators and expressions

Chaochun Wei

Shanghai Jiao Tong University

Spring 2014





2.1 Variable names

- The first character must be a letter
 - “_” is a letter
 - A library function often starts with “_”
 - Upper case and lower case are distinct
 - Lower case for variable names
 - Upper case for symbolic constant
 - Keywords can NOT be used for variable names
 - if, else, int, float, ...
 - Make the variable names meaningful
-



2.1 Variable names

```
1.  /* Here is a program converting Fahrenheit temperatures to their Centigrade or
    2.  Celsius equivalents. */
    3.  //=====
    4.  #include <stdio.h>
    5.  /* print Fahrenheit - Celsius table for fahr = 0, 20 , ..., 300; */
    6.  main() {
    7.      float fahr, celsius;
    8.      int lower, upper, step;
    9.      lower = 0;
    10.     upper = 200;
    11.     step = 20;
    12.     fahr = lower;
    13.     while(fahr <= upper) {
    14.         celsius = (5.0/9.0)* (fahr - 32.0);
    15.         printf("%3.0f %6.1f\n", fahr, celsius);
    16.         fahr = fahr + step;
    17.     }
```



2.2 Data types and sizes

Basic data types

- char one character, a single byte
- int integer
- float single-precision floating point
- double double-precision floating point

Qualifiers of basic types:

- short
 - long
 - signed
 - unsigned
 - const
-



Course organization

- Course introduction (Week 1)
 - Code editor: Emacs
- Part I: Introduction to C programming language (Week 1 - 12)
 - Chapter 1: Overall Introduction (Week 1-4)
 - C
 - Unix/Linux
 - **Chapter 2: Types, operators and expressions (Week 4)**
 - Chapter 3: Control flow (Week 6)
 - Chapter 4: Functions and program structure (Week 7-8)
 - Chapter 5: Pointers and arrays (Week 9)
 - Chapter 6: Structures (Week 10)
 - Chapter 7: Input and Output (Week 11)
- Part II: Skills others than programming languages (Week 12- 14)
 - Debugging tools (Week 12-13)
 - Keeping projects documented and manageable (Week 14)
 - Source code managing (Week 14)
- Part III: Reports from the battle field (student forum) (Week 15 – 16)



2.2 Data types and sizes

Type	Size (bytes)
int	4
char	1
float	4
double	8
short	2
long	8
long double	16



2.3 Constant

- Integer constant
 - 1234 , 123456789L, 123456789UL
- Character constant
 - 'x', 'A', 'Z'
- Constant expression
 - 31+28+31
- String constant
 - "Hello world"
- Enumeration constant
 - enum months {JAN=1, FEB, MAR, APR, MAY, JUN}



2.4 Declarations

- All variables must be declared before use
- A declaration
 - specify a type
 - contains a list of variables of that type
 - `int lower=0, upper, step;`
 - `char c, line[1000];`



2.5 Arithmetic operators

• $+$, $-$, $*$, $/$ and $\%$

$x \% y$

• Precedence (优先级)

• $++$, $--$ $>$ $*$, $/$, $\%$ $>$ $+$, $-$



Relational operators

- $>$ $>=$ $<$ $<=$
- $==$ $!=$
- Precedence

- $== < != < > >= < <= < +, -, *, /, \%$

Logical operators

- $\&\&$ $\|\|$
 - The order of evaluation



2.7 type conversion

- ④ Without information loss: “narrower” to “wider”
 1. float a = 1.2, f;
 2. int i = 2;
 3. f = a + i; /* this is ok */
 4. i = a; /* this will lose information */



2.9 Bitwise operators



Bitwise operators

- `&` bitwise AND
- `|` bitwise inclusive OR
- `^` bitwise exclusive OR
- `<<` left shift
- `>>` right shift
- `~` one's complement (unary)



Examples

- `N=~0` \rightarrow `N= ffffffff 377777777777`



Course organization

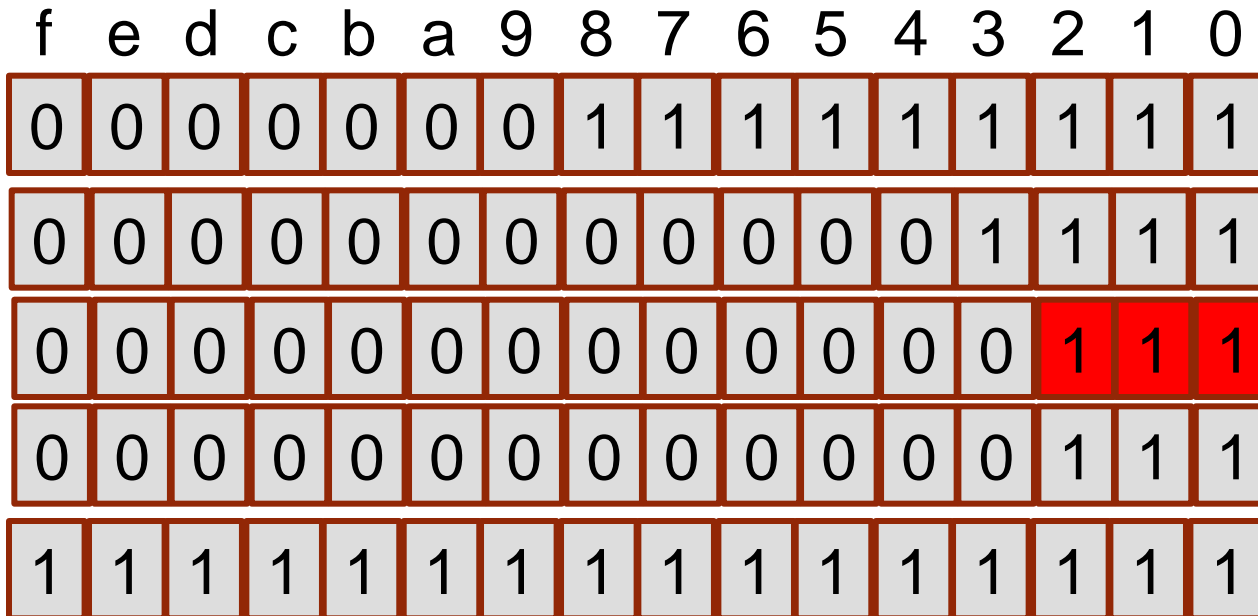
- Course introduction (Week 1)
 - Code editor: Emacs
- Part I: Introduction to C programming language (Week 1 - 12)
 - Chapter 1: Overall Introduction (Week 1-4)
 - C
 - Unix/Linux
 - **Chapter 2: Types, operators and expressions (Week 4)**
 - Chapter 3: Control flow (Week 6)
 - Chapter 4: Functions and program structure (Week 7-8)
 - Chapter 5: Pointers and arrays (Week 9)
 - Chapter 6: Structures (Week 10)
 - Chapter 7: Input and Output (Week 11)
- Part II: Skills others than programming languages (Week 12- 14)
 - Debugging tools (Week 12-13)
 - Keeping projects documented and manageable (Week 14)
 - Source code managing (Week 14)
- Part III: Reports from the battle field (student forum) (Week 15 – 16)



2.9 Bitwise operators

1. /* getbits: get n bits from position p of x*/
2. unsigned getbits(unsigned x, int p, int n)
3. {
4. return (x >> (p+1-n)) & ~(~0 << n);
5. }

getbits(511, 7, 3) → 0111



511

511>>(7+1-3)

(511>>(7+1-3)) & ~(~0<<3)

~(~0<<3)

~0



2.10 Assignment operator (op=)

⊙ op=, where op is one of
+ - * / % << >> & ^ |

Example: bitcount

1. /* bit count: count 1 bits in x */
2. int bitcount (unsigned x) {
3. int b;
4. for (b = 0; x!=0; x>>= 1) {
5. if (x & 01) b++;
6. }
7. return b;
8. }



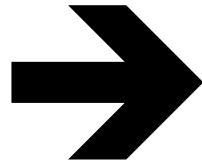
```
/* z = max (a, b) */
```

1. If (a > b)

2. z = a

3. else

4. z = b



z = (a > b) ? a : b



- Basic data types
 - Types: *char, int, float and double*
 - Qualifiers: *short, long, unsigned, signed, const*
- Constant: `0x1234, 12, "Some string"`
- Enumeration:
 - Names in different enumerations must be distinct
 - ```
enum WeekDay_t {Mon, Tue, Wed, Thur, Fri};
enum WeekendDay_t {Sat = 0, Sun = 4};
```
- Arithmetic: `+, -, *, /, %`
  - prefix `++i` or `--i`; increment/decrement before value is used
  - postfix `i++`, `i--`; increment/decrement after value is used
- Relational and logical: `<, >, <=, >=, ==, !=, &&, ||`
- Bitwise: `&, |, ^ (xor), <<, >>, ~(ones complement)`



## 2.12 Precedence and associativity of operators

| Operators                         | Associativity |
|-----------------------------------|---------------|
| () [] -> .                        | Left to right |
| ! ~ ++ -- + - * & (type) sizeof   | Right to left |
| * / %                             | Left to right |
| + -                               | Left to right |
| << >>                             | Left to right |
| < <= > >=                         | Left to right |
| == !=                             | Left to right |
| &                                 | Left to right |
| ^                                 | Left to right |
|                                   | Left to right |
| &&                                | Left to right |
|                                   | Left to right |
| ?:                                | Right to left |
| = += -= *= /= %= &= ^=  = <<= >>= | Right to left |
| ,                                 | Left to right |