上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

Lecture 02

C FUNDAMENTALS

C Fundamentals

Keywords

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Variable Type

- C has the following simple data types:

Variable Type	Keyword	Bytes Required	Range
Character	char	1	-128 to 127
Unsigned character	unsigned char	1	0 to 255
Integer	int	2	-32768 to 32767
Short Integer	short int	2	-32768 to 32767
Long Integer	long int	4	-2,147,483,648 to 2,147,438,647
Unsigned Integer	unsigned int	2	0 to 65535
Unsigned Short Integer	unsigned short int	2	0 to 65535
Unsigned Long Integer	unsigned long int	4	0 to 4,294,967,295
Float	float	4	1.2E-38 to
Double	double	8	2.2E-308 to
Long Double	long double	10	3.4E-4932 to 1.1E+4932

Data Types

- char, int, float, double
- long int (long), short int (short), long double
- signed char, signed int
- unsigned char, unsigned int
- 1234L is long integer
- 1234 is integer
- 12.34 is float
- 12.34L is long float

Floating Types

float single-precision floating-point
double double-precision floating-point
long double extended-precision floating-point

Type	Smallest Positive Value	Largest Value	Precision
float	$1.17*10^{-38}$	$3.40*10^{38}$	6 digits
double	$2.22*10^{-308}$	$1.79*10^{308}$	15 digits

```
double x;
scanf("%lf", &x);
printf("%lf", x);
```

```
long double x;
scanf("%Lf", &x);
printf("%Lf", x);
```

Character Types

```
char ch;
int i;
i = ‘a’;                /* i is now 97 */
ch = 65;                /* ch is now ‘A’ */
ch = ch + 1;            /* ch is now ‘B’ */
ch++;                   /* ch is now ‘C’ */

if(‘a’ <= ch && ch <= ‘z’)

for(ch = ‘A’; ch <= ‘Z’; ch++)
```

Constants

- Declaration:
  - `#define MAXLINE 1000`

    `char line[MAXLINE+1]`
- ‘a‘, ‘\t’, ‘\n’, ‘\0’, etc. are character constants
- strings: character arrays
  - (see <string.h> for string functions)
  - "I am a string“
  - always null (‘\0’) terminated.

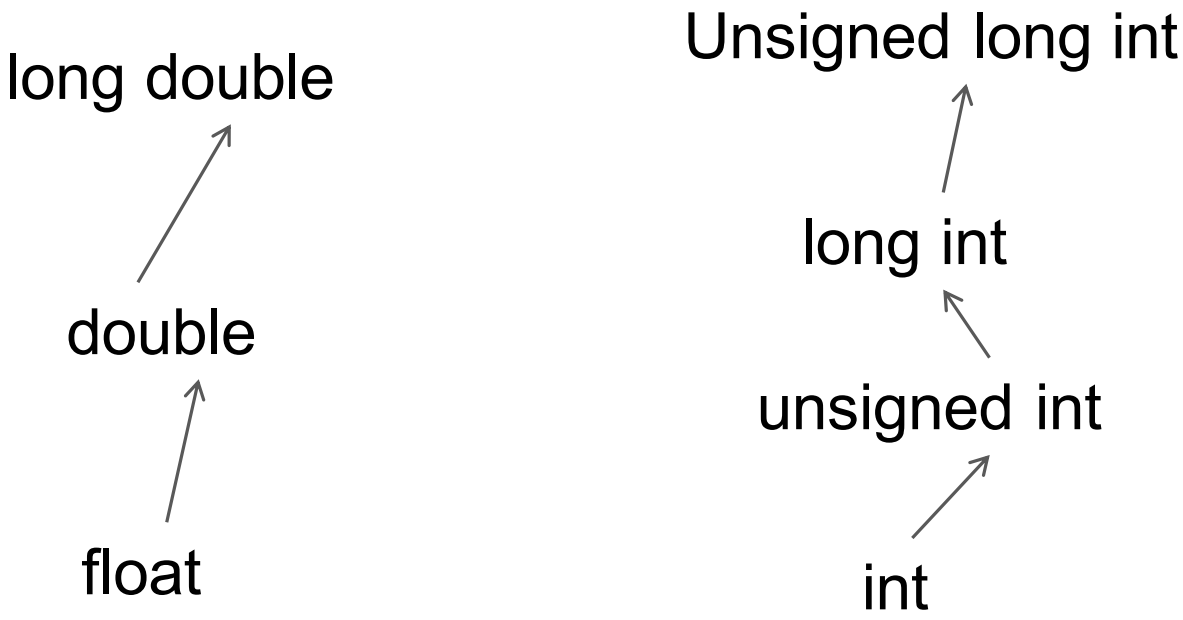
Type Conversion

- narrower types are converted into wider types
  - f + i int i converted to float
- characters <---> integers
- <ctype.h> library contains
- conversion functions, e.g:
  - tolower(c) isdigit(c) etc.
- Boolean values:
  - true : >= 1 false: 0

```
int atoi(char s[]) {
   int i, n=0;
   for (i=0; s[i] >= '0'
&& s[i] <= '9'; i++)
     n = 10*n + (s[i]-'0');

   return n;
}
```

Type Conversion



Type Conversion

```
char c;
short int s;
int i;
unsigned int u;
long int l;
unsigned long int ul;
float f;
double d;
long double ld;
```

```
i = i + c; /*c change to int*/
i = i + s;
u = u +i;
l = l + u
ul =ul + l;
f = f + ul;
d = d + f;
ld = ld + d;
```

Formatted Input/Output

printf function

printf(string, expr1, expr2, ……..)

string: ordinary characters and conversion specifications (%)

%d --- int %s --- string %f --- float

printf("i=%d, j=%d. x=%f\n", i, j, x);

Formatted Input/Output

Escape Sequence

Enable strings to contain characters that would otherwise cause problems for the compiler

alert	\a	new line	\n	\”	\
backspace	\b	horizontal tab	\t	\\	

Most used type

Type	Format	Declaration
char	%c	char x=’a’;
int	%d or %i	int x=1;
(base 8)	%o	int x=03;
(base 16)	%x	int x=0x123;
unsigned int	%u	unsigned int x=2;
float	%f	float x=1.25
double	%lf	double x=1.25

Reading and Writing Integers

```
unsigned int u;
scanf("%u", &u);      /* reads u in base 10 */
printf("%u", u);        /* writes u in base 10 */
scanf("%o", &u);      /* reads u in base 8 */
printf("%o", u);        /* writes u in base 8 */
printf("%d", u);
scanf("%x", &u);      /* reads u in base 16 */
printf("%x", u);        /* writes u in base 16*/
printf("%d", u);
```

```
short int x;
scanf("%hd", &x);
printf("%hd", x);
```

```
long int x;
scanf("%ld", &x);
printf("%ld", x);
```

Formatted Input/Output

- **scanf("%c %f", &type, &temp)**
- **scanf** is controlled by the **conversion specification** in the format string (in "…") starting from left to right.
- When called, it tries to locate an item of the appropriate type (eg. %d, %f) in the input data, skipping ***white-space characters***
  - the space,
  - horizontal and vertical tab,
  - form-feed,
  - and new-line character
- Pay very attention to "%c"

Ordinary Characters in Format String

- ***White-space characters***:
  - one white-space character in the format string will match any number of white-space character in the input.
- ***Other characters***:
  - when it encounters a non-white-space character in a format string, scanf compares it with the next input character.
    - If the two characters match, scanf discards the input character and continues processing the format string.
    - Else, scanf puts the offending character back into the input, then aborts without further processing.
  - eg:
    - %d/%d will match _5/_96, but not _5_/_96
    - %d_/%d will match _5_/_96

Standard Input/Output Library Functions

- Functions in **<stdio.h>**
  - Used to manipulate character and string data

Function prototype	Function description
`int getchar( void );`	Inputs the next character from the standard input and returns it as an integer.
`char *gets( char *s );`	Inputs characters from the standard input into the array **s** until a newline or end-of-file character is encountered. A terminating null character is appended to the array.
`int putchar( int c );`	Prints the character stored in **c**.
`int puts( const char *s );`	Prints the string **s** followed by a newline character.
`int sprintf( char *s, const char *format, ... );`	Equivalent to **printf**, except the output is stored in the array **s** instead of printing it on the screen.
`int sscanf( char *s, const char *format, ... );`	Equivalent to **scanf**, except the input is read from the array **s** instead of reading it from the keyboard.

Expressions

Arithmetic operator: +, -, *, /, %, ++, --.....

- Arithmetic operator:
  - +,-,*./,%
- Relation operators:
  - <,>,<=,>=,==,!=
- Logical operator:
  - && (and), | | (or)
- Increment and decrement operators:
  - ++,--,++,--
- Assignment operators: +=,-=,/=…

Relation Operators

- Relation operators has lower precedence than arithmetic operators:
  - >, >=, <, <=
  - ==, !=
- Don't confuse = and == ! The compiler will warn "suggest parens".

```
int x=5;
if (x==6)    /* false */
{
  /* ... */
}
/* x is still 5 */
```

```
int x=5;
if (x=6)    /* always true */
{
  /* x is now 6 */
}
/* ... */
```

Increment and Decrement Operators

`x++`	`post-increment x`	`++x`	`pre-increment x`
`x--`	`post-decrement x`	`--x`	`pre-decrement x`

Note the difference between ++x and x++:

```
int x=5;
int y;
y = ++x;
/* x == 6, y == 6
*/
```

```
int x=5;
int y;
y = x++;
/* x == 6, y == 5
*/
```

recommendation

Bitwise Operations

- Applied to char, int, short, long
  - And &
  - Or |
  - Exclusive Or ^
  - Left-shift <<
  - Right-shift >>
  - one's complement ~

Assignment Operators

- Most binary operators have a corresponding assignment operator *op*
  - *+, -, *, /, %, <<, >>, &, ^, |*
  - *expr1 op= expr2 <=> expr1 = (expr1) op (expr2)*
  - *x *= y+1 <=> x = x* (y+1)*
- Assignment statement has a value

```
While ((c = getchar()) != EOF)
```

Operator Precedence and Associativity

highest: + - (unary)
* / %
lowest: + - (binary)

-i * -j = (-i) * (-j)
+i + j / k = (+i) + (j / k)

left/right associative: it groups from left/right to right/left

The binary arithmetic operators (*, /, %, + and -) are all left associative
i – j – k = (i – j) – k i * j / k = (i * j) / k

The unary arithmetic operators(+ and -) are both right associative
- + i = - (+i)

Expression Evaluation

Precedence	Name	Symbol(s)	Associativity
1	X++/X--		left
2	++X/--X unary +/-		right
3	multiplicative	*, /, %	left
4	additive	+, -	left
5	assignment	=, *=, /=, +=, -=	right

Expression Evaluation

a = b += c++ - d + --e / -f

a = b += (c++) - d + --e / -f

a = b += (c++) - d + (--e) / -f

a = b += (c++) - d + (--e) / (-f)

a = b += (c++) - d + ((--e) / (-f))

a = b += ((c++) – d) + ((--e) / (-f))

a = b += (((c++) – d) + ((--e) / (-f)))

a = (b += (((c++) – d) + ((--e) / (-f))))

(a = (b += (((c++) – d) + ((--e) / (-f)))))

Example: Bit Count

```
/*
   count the 1 bits in a number
   e.g. bitcount(0x45) (01000101 binary) returns 3
*/

int bitcount (unsigned int x) {
   int b;

   for (b=0; x != 0; x = x >> 1)
      if (x & 01)      /* octal 1 = 000000001 */
          b++;

   return b;
}
```

Conditional Expressions

- Conditional expressions
- expr1? expr2:expr3;
- if expr1 is true then expr2 else expr3