Lecture 13: C/C++ Data Types

Dr. Mohammed Hawa
Electrical Engineering Department
University of Jordan

C/C++ fundamental data types

```
// C/C++ fundamental data types

char     c = 'A';    // 1 byte (8 bits)
short    s = 20;     // 2 bytes
int      i = 110;    // 2 bytes (16-bit CPU) or 4 bytes (32/64-bit CPU)
long     l = 200;    // 4 bytes
long long g = 330;   // 8 bytes
bool     b =  true; // 1 byte (true/false)
float    f = 3.1416;// single precision
double   d = 3.755; // double precision

signed int    si; // signed is the default (so can be removed)
unsigned int  ui;
```

Sizes and Ranges

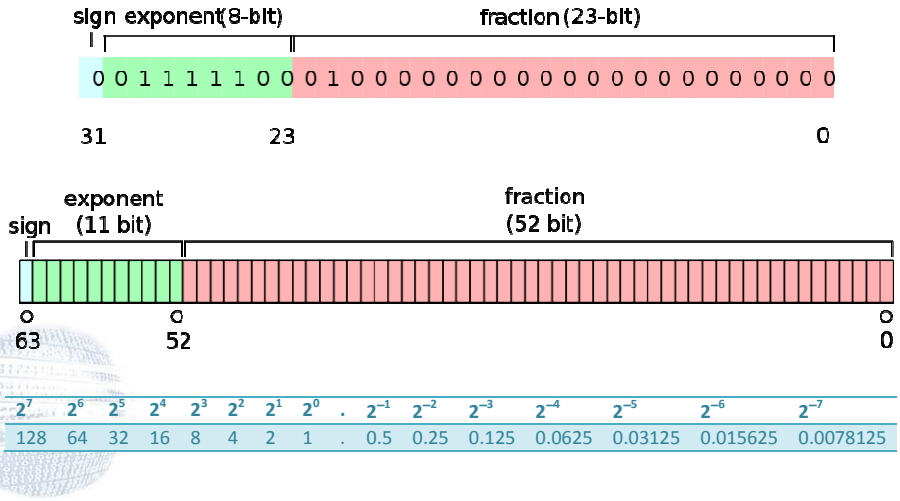
Type Name	Bytes	Other Names	Range of Values
char	1	none	–128 to 127
signed char	1	none	–128 to 127
unsigned char	1	none	0 to 255
short	2	short int, signed short int	–32,768 to 32,767
unsigned short	2	unsigned short int	0 to 65,535
int	4	signed	–2,147,483,648 to 2,147,483,647
unsigned int	4	unsigned	0 to 4,294,967,295
bool	1	none	false or true

Type Name	Bytes	Other Names	Range of Values
long	4	long int, signed long int	–2,147,483,648 to 2,147,483,647
unsigned long	4	unsigned long int	0 to 4,294,967,295
long long	8	none	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
unsigned long long	8	none	0 to 18,446,744,073,709,551,615
float	4	none	IEEE-754 (7 digits) min: ±1.175,494,3 x 10^-38 max: ±3.402,823,4 x 10^38
double	8	none	IEEE-754 (15 digits) min: ±2.225,073,858,507,201,4 x 10^-308 max: ±1.797,693,134,862,315,7 x 10^308
long double	8	none	Same as double (except in certain systems where it is 80-bit extended precision)
enum	varies	none	

Reminder

- We use two's complement (not one's complement)
- A binary integer 1111 1110 is negative?
- Converting 0000 0010 = -2
- A binary integer 1000 0000 is negative?
- Converting 1000 0000 = -128
- A binary integer 0001 0001 is positive?
- No conversion = +17

Float and Double



Example

- Assume this is a float stored in memory. What is its value?

1A12345Dh	
1A12345Ch	
1A12345Bh	1011 1111
1A12345Ah	1110 0000
1A123459h	0000 0000
1A123458h	0000 0000
1A123457h	
1A123456h	

f

Solution: Float Details

- Bit 31: Sign Bit (0: +, 1: –) e.g., 1
- Bits 30 – 23: Exponent Field (bit sequence *unsigned* – 127) e.g., 01111111
- Bits 22 – 0: Significand (1.yyy) is assumed e.g., 110 0000 0000 0000 0000 0000
- Result:
- $(-1)^S$ x 1.75 x 2 ^ (127 – 127)
- – 1.75

Homeworks

- **HW#1:** A float is stored in memory as:
- 0C410100 h = 00001100 01000001 00000001 00000000
- What is the value of this float?
- Solution: 1.486848E-31
- **HW#2:** Write the number 2.75 as IEEE-754 4-byte binary sequence stored in a float data type.

Endianness

- How bytes of a data word are ordered in memory
- **Big-endian**: most significant byte of the word is stored in the smallest address.
- **Little-endian**: most significant byte is stored in the largest address.
- Intel x86 processor (and Microsoft Windows) uses little-endian
- IBM z/Architecture mainframes and Linux use big-endian
- Big-endian is the most common convention in data networking (including IPv6), also known as *network byte order*.

Example

- `long l = 3721182122; // DDCCBBAAh`
- `// Notice the pointer (and address size)`

	Little	Big
1A12345Dh		
1A12345Ch		
1A12345Bh	DDh	AAh
1A12345Ah	CCh	BBh
1A123459h	BBh	CCh
1A123458h	AAh	DDh
1A123457h		
1A123456h		

Example

- Show memory contents for the following data types
- Identify the function call?
- What will you see on the screen?

```
char ch = 'A'; // ASCII for 'A' is 65 = 0100 0001b
short s = ch;

printf("%d\n", s); // will print 65
printf("%c\n", s); // will print A
```

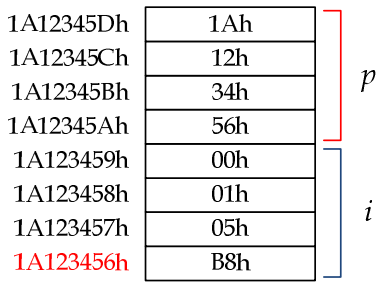
1A12345Dh		
1A12345Ch		
1A12345Bh		
1A12345Ah	0000 0000	*s*
1A123459h	0100 0001	
1A123458h	0100 0001	*ch*
1A123457h		
1A123456h		

Homework

```
short s = 300; // does not fit inside 8-bits
char  ch = s;  // truncate

printf("%d\n", ch); // will print 44


short s = -20;
long  l = s; // extend negative (how?)

printf("%d\n", s); // will print -20
```


Pointers

```
int    i = 0x105B8;// 4 bytes on 32-bit CPU
int*   p = &i;     // & means 'address of'

printf("%d\n", i); // print contents of i = 67000
printf("%d\n", p); // print address of i
printf("%d\n", *p); // print contents at this address (space is int)
```



Integer to Float: CPU is careful!

```
int   i = 5; // 4 bytes
float f = i; // CPU does a conversion (not a copy)

printf("%f\n", f); // %f means floating point
```

- A similar thing happens when converting float/double to integer

Solution

- The value of 5 is converted to 1.25 x 2^2:
- + 1.25 x 2 ^ (129 – 127)
- Bit 31: Sign Bit (0: +, 1: –) e.g., 0
- Bits 30 – 23: Exponent Field (bit sequence *unsigned* – 127) e.g., 10000001
- Bits 22 – 0: Significand (1.yyy) is assumed e.g., 010 0000 0000 0000 0000 0000
- The stored sequence: 40A00000h

C/C++ Allows You Full Control

- You can (if you want) copy the bit sequence *as is* from the integer variable to the float variable!

```
int   i = 5; // 4 bytes
float f = * (float*) &i; // a simple copy :-)

printf("%g\n", f); // 00000005h = 7.00649e-45
```


Homework

```
float f = 7.0; // 4 bytes
short s = * (short*) &f; // a copy and what?

printf("%d\n", s); // prints a 0
```

- Why do we get a 0 as a print out?
- Always solve homework twice:
  - Once using paper and pencil
  - Also using Microsoft Visual Studio C++

Enumeration

```
enum Suit { Diamonds, Hearts, Clubs, Spades };

void PlayCard(Suit suit)
{
     // Enumerator visible without qualification
       if (suit == Clubs)
    {
             /*...*/
    }
}
```


C++ classes

- C++ adds classes
- The standard (std::) library has classes to represent new types:
- `string`, `vector`, `list`, `map`, etc

```
std::string          str = "hello!";
std::vector<int>     v;
std::list<double>    ld;
```