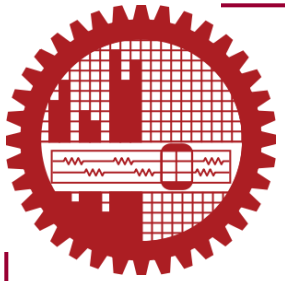

ICT 5101

Lecture 2

Dr. Hossen A Mustafa

Binary

- Computers understand only binary: 0 and 1
- We are used to decimal system: 0 to 9
- We can represent a decimal number 6183 in polynomial form
  - $(6\times10^3) + (1\times10^2) + (8\times10^1) + (3\times10^0) = 6183$
- Binary to Decimal
  - We have binary number 1101
  - $(1\times2^3) + (1\times2^2) + (0\times2^1) + (1\times2^0) = 13$

Byte

- 1 byte or 1B = 8 bit
- 2 byte or 2B = 16 bit
- 4 byte or 4B = 32 bit
- 1 kilobyte or 1KB = 2^10 * 1B = 1024 * 8b
- 1 megabyte or 1MB = 2^20 * 1B = 1048576* 8b
- 1 gigabyte or 1GB = 2^30 * 1B = 1073741824* 8b

Keywords

- auto break case char const continue default do double else enum extern float for goto if int long register return short signed sizeof static struct switch typedef union unsigned void volatile while
- The keywords cannot be used as variable or function name
- Compiler will give error if used
- Example:
  - char is invalid
  - char1 is valid

Datatypes

Type	Size	Example	Values
short int	2 Byte	int x;	-65536 to 65535
int	4 Byte	int x;	-2^31 to 2^31-1
unsigned int	4 Byte	unsigned int x;	0 to 2^32-1
char	1 Byte	char c;	-128 to 127 'A', 'a', '<', '1', etc.
float	4 Byte	float n;	1.17549*10-38 to 3.40282*1038
double	8 Byte	double n;	2.22507*10^-308 to 1.79769*10^308

Expressions

- Traditional mathematical expressions

```
y = a*x*x + b*x + c;
```

- Very rich set of expressions
- Able to deal with arithmetic and bit manipulation

C Expression Classes

- arithmetic: + – * / %
- comparison: == != < <= > >=
- bitwise logical: & | ^ ~
- shifting: << >>
- lazy logical: && || !
- conditional: ? :
- assignment: = += -=
- increment/decrement: ++ --
- sequencing: ,
- pointer: * -> & []

Bitwise operators

- Useful for bit-field manipulations

Operator	Example
and: &	0 & [0 or 1] = 0, 1 & 1 = 1
or: |	1 | [0 or 1] = 1 0 & 0 = 0
xor: ^	0 ^ 0 = 0, 0 ^ 1 = 1, 1 ^ 0 = 1, 1 ^ 1 = 1
not: ~	~0 = 1, ~1 = 0
right shift: >>	10010011 >> 1 = 01001001
left shift: <<	10010011 << 1 = 00100110

Lazy Logical Operators

- "Short circuit" tests save time
- (a == 3 && b == 4 && c == 5)
  - Every expression must be true for the whole expression to be true
- (a == 3 || b == 4 || c == 5)
  - At least one expression must be true for the whole expression to be true
- Evaluation order: left before right

Conditional Operator

- c = a < b ? a + 1 : b – 1;
- Evaluate first expression (a<b).
- If true, use second (a+1), otherwise use third (b-1).
- Puts almost statement-like behavior in expressions.
- Example:
  - what is the values of c is a = 5, b =3?
  - what is the values of c is a = 5, b =13?

Side-effects in expressions

- Evaluating an expression often has side-effects
- a++ increment a afterwards
- a = 5 changes the value of a
- a = foo() function foo may have side-effects

Operator Precedence

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