CS 101:
Computer Programming and Utilization

July-Nov 2016

Prof. Bernard L Menezes
(cs101@cse.iitb.ac.in)

Lecture 3:
Number Representations, Variables, Data Types, and Expressions

Representing Numbers

- Digital circuits can store 0's and 1's
- How to represent numbers using this capability?
- Key idea : Binary number system
- Represent all numbers using only 1's and 0's

Number Systems

- Roman system
  - new symbols for larger numbers
  - could not represent larger numbers

Roman Numeral Table

1	I	14	XIV	27	XXVII	150	CL
2	II	15	XV	28	XXVIII	200	CC
3	III	16	XVI	29	XXIX	300	CCC
4	IV	17	XVII	30	XXX	400	CD
5	V	18	XVIII	31	XXXI	500	D
6	VI	19	XIX	40	XL	600	DC
7	VII	20	XX	50	L	700	DCC
8	VIII	21	XXI	60	LX	800	DCCC
9	IX	22	XXII	70	LXX	900	CM
10	X	23	XXIII	80	LXXX	1000	M
11	XI	24	XXIV	90	XC	1600	MDC
12	XII	25	XXV	100	C	1700	MDCC
13	XIII	26	XXVI	101	CI	1900	MCM

MathATube.com

- Radix based number systems (e.g. Decimal)
- Revolutionary concept in number representation!

Radix-Based Number Systems

- Key idea: position of a symbol determines it's value! PLACE VALUE
  - How do we determine it's relative position in list of symbols?
  - A Zero symbol needed to shift the position of a symbol
- Number systems with radix *r* should have *r* symbols
  - The value of a symbol is multiplied by *r* for each left shift.
  - Multiply from right to left by: 1, r, r^2, r^3, ... and then add

Decimal Number System

- RADIX is 10. Place-Values: 1, 10,100,1000...
- In the decimal system: 346
  - Value of "6" = 6
  - Value of "4" = 4 x 10
  - Value of "3" = 3 x 10 x 10

Quadral Number System

- RADIX is 4. Place values: 1, 4, 16, 64, 256,...
- Only 4 symbols (digits) needed 0,1,2,3
- 23 in quadral:
  - Value of 3 =3
  - Value of 2 = 2 x 4
  - Value of 23 in quadral = 11 in decimal
- 22130 in quadral=
  - 0 + (3 x 4) + (1 x 4 x 4) + (2 x 4 x 4 x 4) + (2 x 4 x 4 x 4 x 4)
    = 668 in decimal

Octal Number Systems

- RADIX is 8. Place Value: 1, 8, 64, 512,....
- 8 digits needed : 0,1,2,3,4,5,6,7
- 23 in octal
  - Value of 3 = 3
  - Value of 2 =   2 x 8
  - Value of 23 in octal = 19 in decimal
- 45171 in octal =
  - 1+8*7+8*8*1+8*8*8*5+8*8*8*8*4
    = 19065 in decimal

Binary System

- Radix= 2
- Needs ONLY TWO digits : 0 and 1
- Place-value: powers of two:

128	**64**	**32**	**16**	**8**	**4**	**2**	**1**

- 11 in binary:
  - Value of rightmost 1 = 1
  - Value of next 1 = 1 x2
  - 11 in binary = 3 in decimal
- 110011

128	**64**	**32**	**16**	**8**	**4**	**2**	**1**
		1	1	0	0	1	1

= 1x1 + 1 x2 + 0 x 4 + 0 x 8 + 1 x 16 + 1 x 32

= 1 + 2 + 16 + 32= 51 (in decimal)

Binary System: Representing Numbers

- Decimal to binary conversion
  - Express it as a sum of powers of two
- Example: the number 154 in binary:
  - 154 = 128 + 16 + 8 + 2
  - $154 = 1 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$

128	**64**	**32**	**16**	**8**	**4**	**2**	**1**
1	0	0	1	1	0	1	0

  - Thus 154 in binary is 10011010

Fractions In Binary

- Powers on the right side of the point are negative:

8	4	2	1	1/2	1/4	1/8	1/16

- Binary 0.1 = 0 + 1 x 2^{-1} = 0.5 in decimal

- In Binary 0.11 = 0x 1 + 1 x 2^{-1} + 1 x 2^{-2}
  = 0.5 + 0.25 = 0.75 in decimal

Representing Non-Negative Numbers

- The number of bits (capacitors/wires) used cannot be chosen arbitrarily
- Choices allowed: 8, 16, 32, 64
- Example: To store 25 using 32 bits:
  - 25 Decimal = 00000000000000000000000000011001
  - So store the following charge pattern (H=High, L=Low)
  - LLLLLLLLLLLLLLLLLLLLLLLLLLLHHLLH
- Range stored: 0 to $2^{32} - 1$. If your numbers are likely to be larger, then use 64 bits.
- Choose the number of bits depending upon how large you expect the number to be.

Representing Integers That Can Be Positive And Negative

- Only byte, half-word, … can be used.
- One of the bits is used to indicate sign
- Sign bit = 0 (low charge/voltage) means positive number, = 1 means negative number
- To store -25 use
  - 10000000000000000000000000011001
  - Leftmost bit = sign bit
- Range stored: $-(2^{31} – 1)$ to $2^{31} – 1$
- Actual representation used: more complex. Two's complement

Representing Real numbers

- Use an analogue of scientific notation:

  significand * $10^{exponent}$, e.g. 6.022 * 10^{22}
- For us the significand and exponent are in binary

  significand * $2^{exponent}$
- Single precision: store significand in 24 bits, exponent in 8 bits. Fits in one word!
- Double precision: store significand in 53 bits, exponent in 11 bits. Fits in a double word!
- Actual representation: more complex. “IEEE Floating Point Standard”

Example

- Let us represent the number 3450 = 3.45 x 10^3
- First: Convert to binary:
- 3450 = $2^{11}+ 2^{10}+ 2^8 + 2^6+ 2^5+2^4 +2^3 + 2^1$

11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	0	1	1	1	1	0	1	0

- Thus 3450 in binary = 110101111010
- 3450 in significand-exponent notation: how?
- 1.10101111010 x 10^{1011}
  - 10 in binary is 2 in decimal
  - 1011 in binary is 11 in decimal, we have to move the "binary point" 11 places to the right

Example Continued

For computer representation:

- Use 23 bits for magnitude of significand, 1 bit for sign
- Use 7 bits for magnitude of exponent, 1 bit for sign

  0110101111010000000000000001011
- Decimal point is assumed after 2nd bit.

Concluding Remarks

- Key idea 1: use numerical codes to represent non numerical entities
  - letters and other symbols: ASCII code
  - operations to perform on the computer: Operation codes
- Key idea 2: Current/charge/voltage values in the computer circuits represent bits (0 or 1).
- Key idea 3: Larger numbers can be represented using sequence of bits.
  - In a fixed number of bits you can represent numbers in a fixed range.
  - If you dedicate a bit to representing the sign, the range of representable numbers changes.
  - Real numbers are represented approximately. If you want more precision or greater range, you need to use larger number of bits.

Outline

- How to store numbers in the memory of a computer
- How to perform arithmetic
- How to read numbers into the memory from the keyboard
- How to print numbers on the screen
- Many programs based on all this

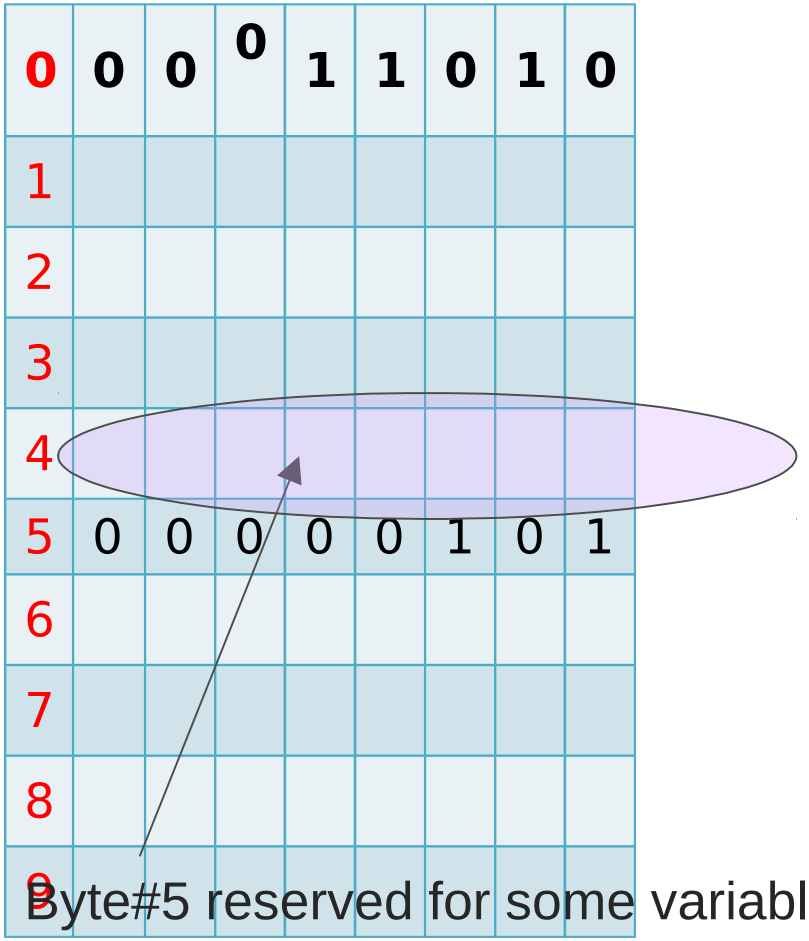
Reserving Memory For Storing Numbers

Before you store numbers in the computer's memory, you must explicitly reserve space for storing them in the memory

This is done by a variable declaration statement.

variable: name given to the space you reserved.

You must also state what kind of values will be stored in the variable: data type of the variable.



Byte#5 reserved for some variable named, "c", say.

Variable Declaration

A general statement of the form:

data_type_name variable_name;

Creates and declares variables

Earlier example

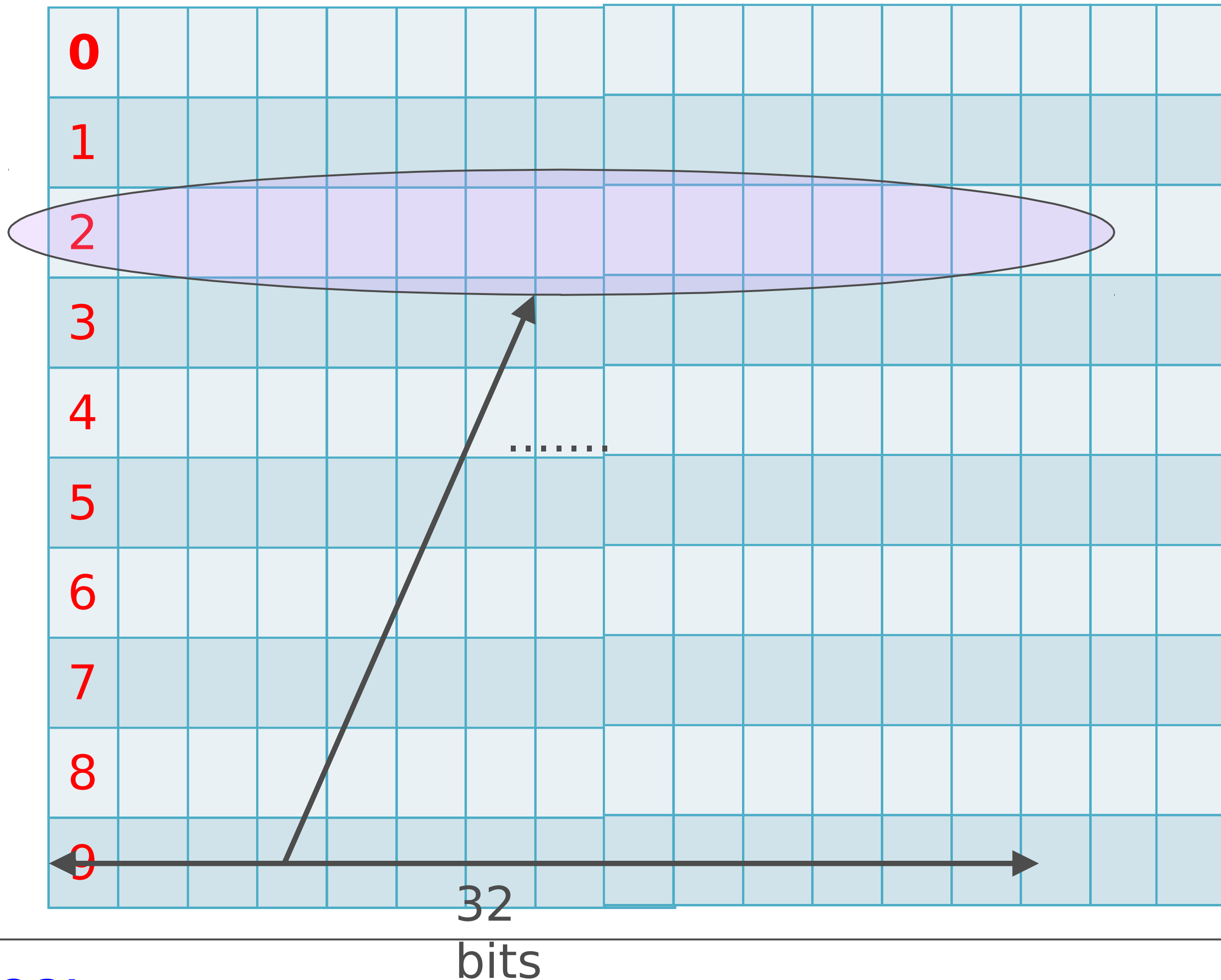
int noofsides;

int : name of the data type. Short form for integer. Says reserve space for storing integer values, positive or negative, of a standard size

Standard size = 32 bits on most computers

noofsides : name given to the reserved space, or the variable created

Variable Declaration



int noofsides;

Results in a memory location of size 32 bits being reserved for this variable. The program will refer to it by the name noofsides

Variable Names: *Identifiers*

Sequence of one or more letters, digits and the underscore "_" character

- Should not begin with a digit
- Some words such as int cannot be used as variable names. Reserved by C++ for its own use
- Case matters. ABC and abc are distinct identifiers

Examples:

- Valid indentifiers: noofsides, telephone_number, x, x123, third_cousin
- Invalid identifiers: #sides, 3rd_cousin, third cousin

Recommendation: use meaningful names, describing the purpose for which the variable will be used

Some Other Data Types Of C++

- unsigned int : Used for storing integers which will always be positive
  - 1 word (32 bits) will be allocated
  - Ordinary binary representation will be used
- char : Used for storing characters or small integers
  - 1 byte will be allocated
  - ASCII code of characters is stored
- float : Used for storing real numbers
  - 1 word will be allocated
  - IEEE FP representation, 8 bits exponent, 24 bits significand
- double : Used for storing real numbers
  - 2 words will be allocated
  - IEEE FP representation, 11 bits exponent, 53 bits significand

Variable Declarations

- Okay to define several variables in same statement
- The keyword long : says, I need to store bigger or more precise numbers, so give me more than usual space.
- long unsigned int: Likely 64 bits will be allocated
- long double: likely 96 bits will be allocated

```
unsigned int
   telephone_number;

float velocity;

float mass, acceleration;

long unsigned int
   crypto_password;

long double
   more_precise_vaule;
```

Variable Initialization

- Initialization - an INITIAL value is assigned to the variable

  the value stored in the variable at the time of its creation

  – Variables i, vx, vy are declared and are initialized
  – 2.0e5 is how we write $2.0*10^5$
  – ‘f’ is a character constant representing the ASCII value of the quoted character
  – result and weight are declared but not initialized

```
int i=0, result;

float vx=1.0,
      vy=2.0e5,
      weight;

char value = ‘f’;
```

Const Keyword

const double pi = 3.14;

The keyword const means : value assigned once cannot be changed

Useful in readability of a program

```
area = pi * radius * radius;
```

reads better than

```
area = 3.14 * radius * radius;
```

Reading Values Into Variables (1)

- Can read into several variables one after another
- If you read into a char type variable, the ASCII code of the typed character gets stored
- If you type the character ‘f’, the ASCII value of ‘f’ will get stored

```
cin >> noofsides;

cin >> vx >> vy;

char command;

cin >> command;
```

Reading Values Into Variables (2)

Some rules:

- User expected to type in values consistent with the type of the variable into which it is to be read
- Whitespaces (i.e. space characters, tabs, newlines) typed by the user are ignored.
- newline/enter key must be pressed after values are typed

Printing Variables On The Screen

- General form: *cout << variable*;
- Many values can be printed one after another
- To print newline, use endl
- Additional text can be printed by enclosing it in quotes
- This one prints the text Position: , then x and y with a comma between them and a newline after them
- If you print a char variable, then the content is interpreted as an ASCII code, and the corresponding character is printed.

  G will be printed.

```
cout << x;

cout << x << y;

cout <<“Position:" <<
x << “,  “ << y  <<
endl;

char var = ‘G’;
cout << var;
```

An Assignment Statement

Used to store results of computation into a variable. Form:
variable_name = expression;

Example:

s = u*t + 0.5 * a * t * t;

Expression : can specify a formula involving constants or variables, almost as in mathematics

- If variables are specified, their values are used.
- operators must be written explicitly
- multiplication, division have higher precedence than addition, subtraction
- multiplication, division have same precedence
- addition, subtraction have same precedence
- operators of same precedence will be evaluated left to right.
- Parentheses can be used with usual meaning

Examples

```
int x=2, y=3, p=4, q=5, r, s, t;
x = r*s;                 // disaster. r, s undefined
r = x*y + p*q;           // r becomes 2*3 + 4*5 = 26
s = x*(y+p)*q;           // s becomes 2*(3+4)*5 = 70
t = x – y + p – q;           // equal precedence,
              // so evaluated left to right,
              // t becomes (((2-3)+4)-5 = -2
```

Arithmetic Between Different Types Allowed

```
int x=2, y=3, z, w;
float q=3., r, s;
r = x;      // representation changed
        // 2 stored as a float in r  "2.0"
z = q;      // store with truncation
        //  z takes integer value 3
s = x*q; // convert to same type,
            // then multiply
            // Which type?
```

Evaluating varA op varB
e.g. x*q

- if varA, varB have the same data type: the result will have same data type
- if varA, varB have different data types: the result will have more expressive data type
- int/short/unsigned int are less expressive than float/double
- shorter types are less expressive than longer types