Logistics

- Attendance - mark using SAFE (between 3:30 - 3:40)
  - For any issues: write to safe@cse.iitb.ac.in
- Accessing documents (iitb login)
- Extra lab sessions (every Saturday 2-4 pm).
- Not having laptops, Additional slots? — talk to me at the end.
- 15 Aug lab

Lecture 3: Variables, data types, expressions

What we have seen so far

- Turtle graphics commands like, forward(n), left(n)
- repeat(n){} — iteration,
- cin, cout
- How all data is represented using numbers
- Numbers are stored in binary (sequences of 0/1)

How numbers are written in a computer?

- A computer has a large set of capacitors
  - 8 GB RAM means $2^{36} \approx 6 \times 10^{10}$ capacitors
  - Each capacitor represents a bit (0 or 1)
    - Low charge represents 0
    - High charge represents 1
- Once you represent 0 and 1, you can represent anything.
- Binary number system
- Electrical circuits are designed to perform basic operations like addition, multiplication, copying.

Binary number system

- A sequence $a_n\, a_{n-1} \ldots \; a_1\, a_0\, a_{-1} \ldots a_{-k}$ of 0s and 1s
- Will represent the number
  $a_n\, 2^n + a_{n-1}\, 2^{n-1} + \ldots + a_1\, 2^1 + a_0\, 2^0 + a_{-1}\, 2^{-1} + \ldots + a_{-k}\, 2^{-k}$
- Converting decimal integer v to binary
  - Divide v by 2, remainder gives a_0
  - Repeat previous step with the quotient to get a_1, a_2, …
- Converting fraction f to binary
  - If $f > 0.5$, $a_{-1} = 1$ and 0 otherwise.
  - Similarly other bits…

Positive/negative numbers

- Typically number of capacitors used for an integer
  - 8, 16, 32, 64
- One of the bits (leftmost) indicates sign
- Numbers stored from -2^{31} from $2^{31}-1$
- Two's complement
  - For $0 \leq x \leq 2^{31}-1$, x represents x
  - For $2^{31} \leq x \leq 2^{32}-1$, x represents $2^{32} - x$
  - See next slide for examples.

Two's complement for three bits

Bits	Unsigned value	Signed value
000	0	0
001	1	1
010	2	2
011	3	3
100	4	-4
101	5	-3
110	6	-2
111	7	-1

int nSides

```
int nSides;
```

- C++ will typically designate some 32 capacitors in your computer for the variable nsides.
- The bit pattern stored in it will be interpreted as having a sign and a magnitude.
- 10000000000000000000000000011001 will mean $-2^{31}+25$

```
unsigned int nSides;
```

- 32 capacitors will be given, but the bit pattern in it will be interpreted as 32 bit binary number.
- 10000000000000000000000000011001 will mean $2^{31}+25$

Bit, Byte, word

- Bit = 1 binary “digit”, (one number = 0 or 1)
- byte = 8 bits
- half-word = 16 bits
- word = 32 bits
- double word = 64 bits

- “one byte of memory” = memory capable of storing 8 bits = 8 capacitors.

Representing real numbers

- Use analogue of scientific notation: $significand \times 10^{exponent}$
  - e.g. 6.022×10^{23}
- Same idea, but significand, exponent are in binary, thus number is:
  - $significand \times 2^{exponent}$
- "Single precision":
  - store significand in 24 bits (7-8 decimal digits),
  - exponent in 8 bits.
  - Fits in one word!
- "Double precision": store significand in 53 bits, exponent in 11 bits.
  - Fits in a double word!
  - 53 bits of significand = 16-17 decimal digits
- More implementation details later

Exercises

- Suppose I want to represent 8 digit telephone numbers.
  - I should use 8/16/32/64 bit signed/unsigned representation.
- What is roughly the largest number that can be represented using 64 bits?
- If we represent real numbers with 24 bits significand and 8 bit exponent
  - What is the largest possible number ?
  - What is the smallest possible positive number ?

Summary

- Numbers are represented by sequence of 0s and 1s
- The same sequence may mean one number as an unsigned integer, a signed integer, or a floating point number, or a character
- The capacitors only store high or low charge, they are not “aware” that the charge represents numbers.
- So long as we remember what type of number we are storing, there will be no problem.
- As a user, you don’t need type/read binary numbers.
- C++ will convert binary numbers to decimal system while printing
- C++ will accept numbers typed in decimal by you and itself convert it to binary for use on the computer.
- But you should know (roughly) what range of numbers can be stored in k bit unsigned/signed/floating formats

Reserving memory for storing numbers

- Before you store numbers, you must explicitly reserve space for storing them.
  - “space” : region of memory
- This is done by a “variable definition” statement.
- variable: name given to the space you reserved.
- “Value of a variable”: value stored in the variable
- You must also state what kind of values will be stored in the variable: “data type” of the variable.

Variable creation / definition

- Statement form:
  data-type-name variable-name;
- Example from chapter 1:
  `int nSides;`
- `int` : data type name. Short for “integer”.
- Reserve space for storing integer values, positive or negative, of a “standard” size.
- Standard size = 32 bits on most computers.
- Two’s complement representation will typically be used.
- `nSides` : name given to reserved space, or the created variable.

Variable names/identifier

- Sequence of 1 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
  - Space not allowed inside variable name
- Examples: `nsides, telephone_number, x, x123, third_cousin`
- Not valid: `#sides, 3rd_cousin, 3 rd cousin`
- Recommendation: use meaningful names, describing the purpose for which the variable will be used.

Variable names/identifier

- `unsigned int` : Used for storing integers which will always be positive.
    - 1 word 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.

Examples

```
unsigned int telephone_number;
float mass, acceleration;
```

- You can define several variables in same statement as above.
- Keyword long : says, “I need to store bigger or more precise numbers, so give me more than usual space.”

```
long unsigned int cryptographic_password;
```

  - Likely 64 bits will be allocated.

```
long double more_precise_acceleration;
```

  - Likely 96 bits will be allocated

Variable initialization

- A value can be stored in a variable at the time of creation

```
int i=0;
```

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

- `i`, `vx`, `vy` given values as well as defined.
- `2.0e5` is how you write 2.0×10^5
- Although the computer uses binary, you write in decimal.

```
char command = 'f';
```

- 'f' is a "character constant". It represents the ASCII value of the quoted character.

Reading values into variables

```
cin >> nSides;
```

- Can read into several variables one after another

```
cin >> vx >> vy;
```

- User expected to type in values consistent with the type of the variable into which it is to be read.
- “Whitespace” = space characters, tabs, newlines, typed by the user are ignored.
- newline/enter key must be pressed after values are typed
- If you read into a char type variable, the ASCII code of the typed character gets stored.

```
char command;
cin >> command;
```

- If you type the character ‘f’, its ASCII value, 102, will get stored.

Printing variables on the screen

- General form: `cout << variable-name;`
- To print newline, use `endl`.
- Additional text can be printed by enclosing it in quotes.
- Many things can be printed by placing `<<` between them.

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

- Prints the text “Position: “, then values of variables `x`, `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.

```
char command = 'G', command2 = 97;
cout << command << command2;  // Ga will be printed.
```

Assignment statement: storing a value into a variable defined earlier

- Statement form: variable = expression;

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

- Expression : formula involving constants or variables, almost as in mathematics.
- Execution: The value of expression is calculated and stored into the variable which appears on the left hand side of the = symbol.
  - If expression contains variables they must have been assigned a value earlier – this value will be used to calculate the value of the expression.
  - If `u, a, t` have values 1, 2, 3 respectively, 1*3 + 0.5 * 2 * 3 * 3 = 12 will be stored in `s`.
  - The value present in `s` before the execution of the statement is lost or "overwritten".
  - The values present in other variables including `u, a, t` does not change when the above statement executes.

Rules regarding expressions

- Multiplication must be written explicitly.
- `S = u t + 0.5 a t t;` // not legal
- 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.
- Spaces can be put between operators and values as you like

```
s=u*t+0.5*a*t*t;    s = u*t + 0.5*a*t*t;  // both OK
```

Examples

```
int x=2, y=3, p=4, q=5, r, s, t, u;
r = x*y + p*q;
      // 2*3 + 4*5 = 26
s = x*(y+p)*5;
      // 2*(3+4)*5 = 70
t = x - y + p - q;
      // ((2-3)+4)-5 = -2
u = x + w;
//wrong if w not defined earlier
```

Example with division

```
int x=2, y=3, z=4, u;
u = x/y + z/y;
```

- What will this do?
- If dividend and divisor are both integers, then result is the quotient, i.e. remainder is ignored.
- u becomes: $2/3 + 4/3 = 0 + 1 = 1$
- If you divide an int variable by zero, it has undefined behaviour.