# CS 101: Working with numbers in C++

Abhiram Ranade

# Outline

- How to store numbers in the memory of a computer
- How to perform arithmetic on them.
- How to read them from the keyboard and how to print them.
- Some programs based on what we learn.

# Let us start with something you already know!

- A statement we saw in lecture 1:
  - int nsides;
- First half of the lecture:
  - formalization and generalization of this statement.

# Drawing a polygon

```
main_program{
  turtleSim();
  cout << “How many sides?”
  int nsides;
  cin >> nsides;
  repeat(nsides){
        forward(10); right(360/nsides);
  }
  wait(10); closeTurtleSim();
}
```

# int nsides;

- nsides: name of variable defined.
- int : Says
  - Use 1 word of memory (typically).
  - Variable will hold integers, positive or negative. So use appropriate number representation.
    - “Essentially” 1 sign bit and 31 bits of magnitude.
    - Actual representation: 2’s complement. See book.
- int : Data type

# Some definitions

- Variable: A region of memory that stores a single piece of data
  - Variables can have names
- Data type: Specifies
  - how much memory is to be used
  - what kind of data will be stored, representation used for storing values.

# Variable names

- Sequence of 1 or more letters, digits, or the underscore character
  - Should not begin with a digit.
  - Exceptions: C++ keywords, e.g. int
- Examples: nsides, nSides, a_123, A_123;
- Non-examples: #sides, 123_a
- Recommendation: Use names that describe the purpose for which the variable will be used.

# Another data type: unsigned int

- 1 word (typically).
- only non-negative integers will be stored.  Use binary representation.

Example:

```
unsigned int telephone_number;
```

int, unsigned int : built-in data types

# More built in data types

float

- 1 word (typical). Stores real numbers.  Use 24 bits for fraction, 8 bits for exponent.
- 24 bits precision = 7-8 decimal digits

double

- 2 words (typ.). Stores real numbers. 53 bits for fraction, 11 bits for exponent.
- 15 decimal digits

float velocity; double pressure;

# More built in data types

short

- Stores integers. 2s comp.  Typ.  16 bits.

long

- stores integers.  2s complement. Typ. 32 bits.

long long

- stores integers 2s complement. Typ. 64 bits.

unsigned versions also allowed

```
long long very_long_var;
unsigned short  short_var;
```

# Examples of variable definitions

```
float velocity, pressure, temperature;
```

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

- vx, vy given values as well as defined.

```
const double PI = 3.141592654;
```

- given value cannot be changed.

# Reading values into variables

```
cin >> varname;
```

```
cin >> var1 >> var2;
```

- “white space” ignored
- “enter” needed to signal end of typing.

# Assignment statement

Form: varname = expression

Expression: almost as in mathematics. *,/ have higher precedence than +,-.

Multiplication must be written explicitly as *.

() can be used.

```
double s, u, a, t;

cin >> u >> a >> t;

s = u*t + a * t * t / 2;
```

# More examples

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

# More examples

```
int x=2, y=3, z, w;
float q=3.5, r, s;
r = x;   // representation changed
z = q;  // store with truncation
s = x * q;  // convert to same type,
            // then multiply.
            // Which type?
```

# Evaluating “varA op varB” e.g. x*q

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

# Another example

```
int x=2, y=3, p=4, q=5, u;
u = x/y + p/q;
cout << p/y;
```

x/y : both are int. So truncation. Hence 0.

p/q : similarly 0.

p/y : 4/3 after truncation will be 1. prints 1.

# Yet another example

```
int nsides=100, i_angle1, i_angle2;
i_angle1 = 360/nsides;
i_angle2 = 360.0/nsides;

float f_angle1, fangle_2;
f_angle1 = 360/nsides;
f_angle2 = 360.0/nsides;
```

# Implication of limited precision

```
float w, y=1.5, avogadro = 6.022e23;
w = y + avogadro;
```

“Actual sum” : 602200000000000000000001.5

y + avogadro will have type float, i.e. about 7 digits of precision.  To 7 digits of precision avogadro is same as y+avogadro.

w will equal avogadro. no effect of addition!

# Program example

```
main_program{
  double centigrade, fahrenheit;
  cout << “Give temperature in Centigrade: ”;
  cin >> centigrade;
  fahrenheit = centigrade * 9 / 5 + 32;
  cout << “In Fahrenheit: ” << fahrenheit
          << endl;  // newline.
}
```

# Re assignment

- Same variable can be assigned again.

```
int p=3, q=4, r;
r = p + q;
cout << r << endl;
r = p * q;
cout << r << endl;
```

# An interesting assignment expression

```
int p=12;
p = p + 1;
```

Rule for evaluation: first evaluate the value on the left hand side. Then store the result into the lhs variable.

At the end p will be 13.

“p = p + 1” is nonsensical in mathematics.

“=” in C++ is different from “=” in math.

# Repeat and reassignment

- What does the following program print?

```
main_program{
   int i=1;
    repeat(10){
        cout << i << endl;
        i = i + 1;
    }
}
```

# Fundamental idiom

Sequence generation

- Variable takes consecutive values.
- Can we make i take values 1, 3, 5, 7, ...?
- Can we make i take values 1, 2, 4, 8, 16, ...?

# Repeat and reassignment

- What does the following program print?

```
main_program{
  int term, s = 0;
   repeat(10){
      cin >> term;
      s = s + term;
   }
   cout << s << endl;
}
```

# Another fundamental idiom

Accumulation

- Can we make s become the product of all values read?

# Composing the two idioms

- Write a program to calculate n! given n.

# Composing the two idioms

- Write a program to calculate n! given n.

```
main_program{
  int n, nfac=1, i=1;
  cin >> n;
  repeat(n){
     nfac = nfac * i;
     i++;                              // short for i = i + 1;
  }
  cout << nfac << endl;
}
```

# Exercises (practice)

- Compute $e^x = 1 + x/1! + x^2/2! + x^3/3! + \ldots$
- Compute ln x by integrating f(x)=1/x from 1 to x. Break the area from 1 to x into some n strips, and if x is the x-coordinate at the center of some strip, estimate the area of the strip to be width * height = (x-1)/n * (1/x)
- Draw a spiral. The spiral should intersect any radial line at equal intervals.
- Chapter 2 of book.