Introduction to Programming (CS 101)

Spring 2024

Lecture 2:

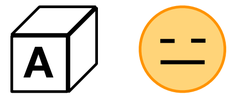
Variables, data types, operators
cout/cin

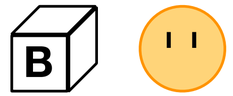
Instructor: Preethi Jyothi

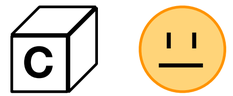
Based on material developed by Prof. Abhiram Ranade

Recap

Pick the closest emoji that the following piece of code will draw.








```
#include <simplecpp>
main_program{
  turtleSim();
  int nsides = 40;
  repeat(nsides){
    forward(400.0/nsides);
    right(360.0/nsides);
  }

  forward(5); right(90);
  penUp();
  forward(50); right(90); forward(10);
  penDown(); forward(20); right(180);
  penUp(); forward(30); forward(10);
  penDown(); forward(20);

  hide();
  getClick();
}
```

Variables, data types and operators

- Recall:

```
#include <simplecpp>
main_program {
    turtleSim();
    int nsides = 4;
    repeat(nsides) {
        forward(100);
        right(90);
    }
    getClick();
}
```

A ***variable*** called "`nsides`" whose value can be set and modified later

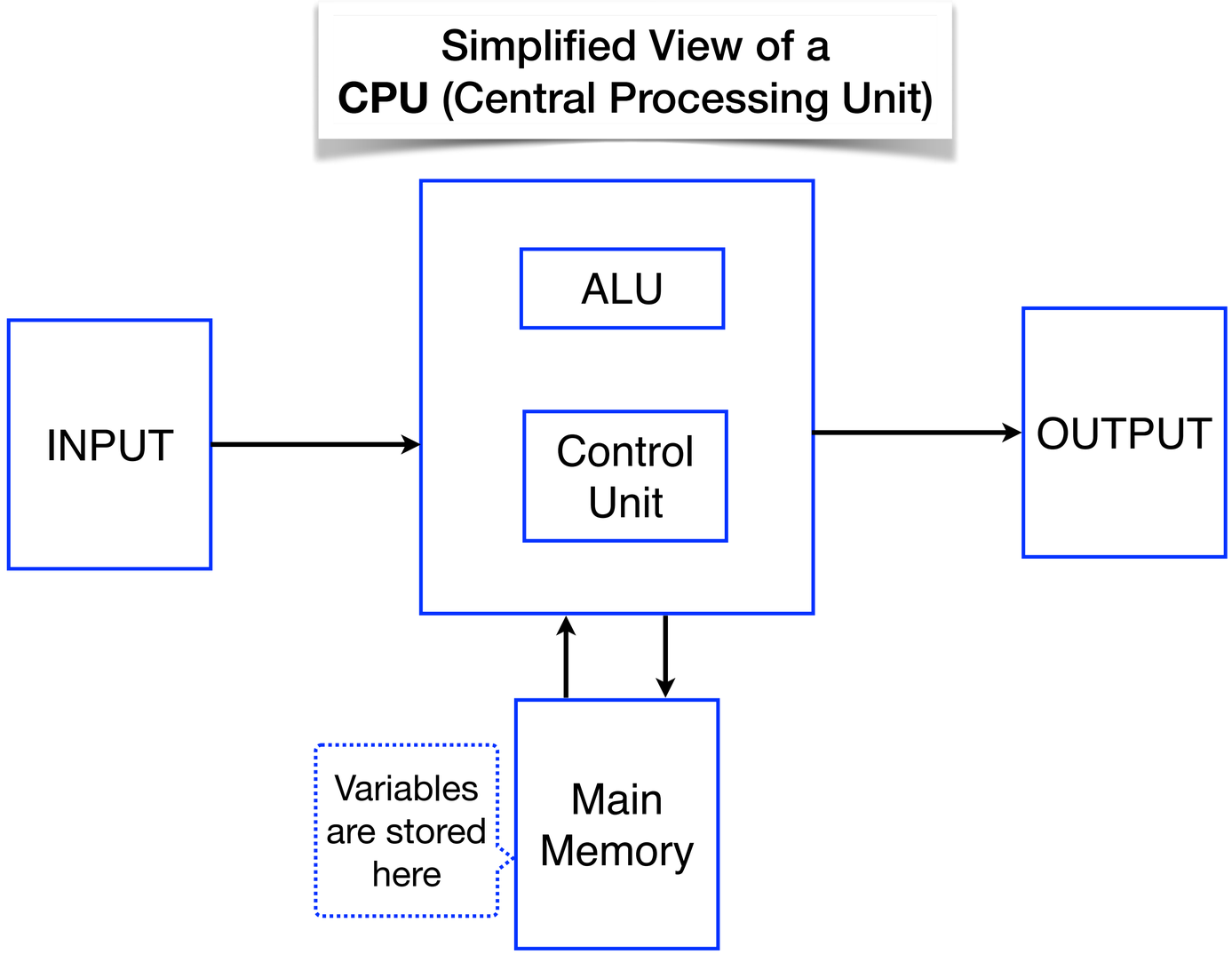
= is the assignment operator

The ***type*** of the variable is ***int***: short for integer

- We will cover four data types today (and more later): `char`, `int`, `float` and `double`

Variables

- A ***variable*** occupies a region of memory (in a computer) into which you can store a value



Variables and data types

- A ***variable*** occupies a region of memory (in a computer) into which you can store a value
- Create ***variables*** of a particular ***data type*** by giving each variable a name, to refer to in the rest of the program
- A variable declaration is as follows:

```
data-type variable-name;
```

- Several variables of the same data type can be defined in a single statement:

```
data-type variable-name-1, variable-name-2, variable-name-3;
```

Variables and data types

- A ***variable*** occupies a region of memory (in a computer) into which you can store a value
- Create ***variables*** of a particular ***data type*** by giving each variable a name, to refer to in the rest of the program
- A variable declaration is as follows:

```
data-type variable-name;
```

where `data-type` is one of:

Data type	Possible values	Used for
`char`	`-128 to 127`	Characters
`int`	`-2147483648 to 2147483647`	Standard size Integers
`float`	`About 7 digits of precision`	Real numbers
`double`	`About 15 digits of precision`	Real numbers

More data types such as `bool` will be covered in later lectures

Variable names

- Variable names or ***identifiers*** can consist of letters, digits, and the underscore character "_"
- Reserved words in C++ (e.g., `int`, `char`, etc.) cannot be used as variable names
- Not recommended to start a name for an ordinary variable with "_"
- A variable name cannot start with a digit. E.g., `2ndname`
- *Case (lowercase vs. uppercase) is important* and distinguishes variable names from one another. E.g., `coursetotal` is different from `CourseTotal`

`char` (short for character)

- `char` data type is most commonly used to store a single character
- Use single quotes around the character. Example of usage: `char grade = 'A';`
- An integer value is stored in char variables, referred to as ASCII values (ranging from 0 to 127). ASCII integer value to character mapping is at [1]. E.g., ASCII value of 'A' is 65.
- `char j = 'A';` and `char j = 65;` are equivalent
- `a–z`, `A–Z`, `0–9` → all have consecutive ASCII codes

What does this code segment do?

```
char in_char = ‘c’, out_char;
out_char = in_char + 'A' – 'a'
```

Assume only lower case letters in `in_char`

Ans: Prints out the upper case letter corresponding to the one in `in_char`

[1] https://en.cppreference.com/w/cpp/language/ascii

`char` (short for character)

- `char` data type is most commonly used to store a single character
- Use single quotes around the character. Example of usage: `char grade = 'A';`
- An integer value is stored in char variables, referred to as ASCII values (ranging from 0 to 127). ASCII integer value to character mapping is at [1]. E.g., ASCII value of 'A' is 65.
- `char j = 'A';` and `char j = 65;` are equivalent
- `a-z`, `A-Z`, `0-9` → all have consecutive ASCII codes
- Escape sequences: Special characters such as ', ", \, etc. can be written as they are with an escape sequence.

  `char j = ''';` throws an error but `char j = '\'';` does not

[1] https://en.cppreference.com/w/cpp/language/ascii

`int` (short for integer) and `float/double` (short for floating-point)

`int`

- `int` represents integers in the range -2147483648 to 2147483647
- If you know you are only dealing with non-negative integers, use the data type `unsigned int` that represents numbers in the range 0 to 4294967295
- Other *flavours* of `int` that we will learn later (`short int`, etc.)

`float/double`

- `float/double` are both used for decimals with 7 and 15 decimal digits precision, respectively. Magnitude ranges are 1.17549×10^{-38} to 3.4028×10^{38} and 2.22507×10^{-308} to 1.7977×10^{308}
- Typically use `double` unless you know you are dealing with small floating-point numbers
- Scientific notation: `float i = 2E-3f` that stands for `2 x (10^-3)`

Note the suffix '`f`' that denotes the value is float

`int` (short for integer) and `float/double` (short for floating-point)

`int`

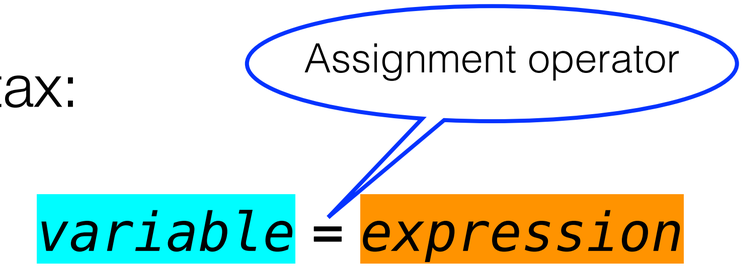
- `int` represents integers in the range -2147483648 to 2147483648
- If you know you are only dealing with non-negative integers, use the data type `unsigned int` that represents numbers in the range 0 to 4294967295
- Other *flavours* of `int` that we will learn later (`short int`, etc.)

`float/double`

- `float/double` are both used for decimals with 7 and 15 decimal digits precision, respectively. Magnitude ranges are 1.17549×10^{-38} to 3.4028×10^{38} and 2.22507×10^{-308} to 1.7977×10^{308}
- Typically use `double` unless you know you are dealing with small floating-point numbers
- Scientific notation: `float i = 2E-3f` and `double i = 2E-3`
- Floating point arithmetic has many subtleties that we will cover in a later class

Assignment operation

- Syntax:



- Semantics (Meaning): Evaluate *expression* and then store in the result in *variable*

- Examples:

```
int nsides = 4;
```

Here, *expression* is a constant (i.e., 4)

```
int i = 1;
i = i + 1;
```

Here, *expression* is a simple arithmetic operation on `i` (i.e., `i + 1`)

```
int i = 1;
i = (i * i) + 2;
```

Here, *expression* is another arithmetic operation on `i` with two operators '`*`' and '`+`'

Illustrating assignment

- What does this program do?

```
#include <simplecpp>
main_program {
    turtleSim();
    int x = 5;
    repeat(100) {
        forward(x); right(90);
        x = x + 5;
    }
    hide();
    getClick();
}
```

- `int x = 5`: x is assigned an initial value of 5 or x is initialized with the value 5
- `x = x + 5`: Add 5 to x and then store this value back into x

Arithmetic operations

- Variables can be manipulated using arithmetic operations, much like in algebra
- Common arithmetic operators: `+`, `-` , `*`, `/` E.g.: `x = (a / d) + (b * c)`
- Preference order of operators:
  - Multiplication and division have the same precedence, which is higher than ...
  - ... addition and subtraction that have the same precedence
  - Among operators of the same precedence, we go left-to-right

```
int x = 6 * 2 + 3;
```

Here, x will evaluate to ***15***

```
int x = 6 / 3 * 2;
```

Here, x will evaluate to ***4***

```
int x = 6 - 3 / 3 + 2;
```

Here, x will evaluate to ***7***

Order of assignment vs. arithmetic operators

- With arithmetic operators, evaluation order is left-to-right
- With multiple assignment operators, the rightmost assignment is evaluated first i.e. the order is right-to-left

```
int x = 1, y = 2, z = 3;
x = y = z = 4;
```

What are the values of `x, y, z`?

Ans: `x = 4, y = 4, z = 4`

```
int x = 1, y = 2, z = 3;
x = y = z = x + z;
```

What are the values of `x, y, z`?

Ans: `x = 4, y = 4, z = 4`

```
int x = 1, y = 2, z = 3;
x = y = z = x - z + y;
```

What are the values of `x, y, z`?

Ans: `x = 0, y = 0, z = 0`

Relational operators

- Relational operators are used to compare two variables or expressions
- Returns either false (zero) or true (non-zero) value
- Operators include:
  - Greater than (>): `sum > 10`
  - Greater than or equal to (>=): `i >= 5`
  - Less than (<): `sum < 10`
  - Less than or equal to (<=): `i <= 5`
  - Equal to (==): `a == 1`
  - Not equal to (!=): `1 != 2`

Note this is different from the assignment = operator

Compound assignment operators

- A compound assignment is as follows:

```
variable += expression
```

- This is short-hand for:

```
variable = variable + expression
```

- Also exists for other arithmetic operators: `-=`, `*=`, `/=`

Input / Output

CS 101, 2025

Recall code to draw a square

- Recall:

```
#include <simplecpp>
main_program {
    turtleSim();
    int nsides = 4;
    repeat(nsides) {
        forward(100);
        right(90);
    }
    getClick();
}
```

- What if I want to
  - ask the user for their choice of nsides?
  - print that number to the screen?

Input/output: `cin`/`cout` operators

- Recall:

```
#include <simplecpp>
main_program {
    turtleSim();
    cout << "How many sides?";
    int nsides;
    cin >> nsides;
    repeat(nsides) {
        forward(100);
        right(90);
    }
    getClick();
}
```

Print "How many sides?" to the screen

Read from keyboard into the variables `nsides`

Input/output: `cin`/`cout` operators

- Standard input (`cin`) and output (`cout`) streams: Get input (from a keyboard) and produce output (on screen)

- `cout` used with insertion operators `<<`

- Multiple insertion operations (`<<`) may be chained in a single statement...

```
cout << "How many sides " << "in the polygon?";
```

... also with variables to print out their values:

```
cout << "There are " << nsides << " in the polygon" << endl;
```

- Note: cout typically ends with an `endl` so that the statement appears in the order of execution

Input/output: `cin`/`cout` operators

- `cin` used with extraction operators `>>`
- Syntax:

```
int x;
cin >> x;
```

- Multiple extraction operations (`>>`) may be chained in a single statement...

```
cin >> a >> b;
```

- Any kind of space (space, tab, newline) can separate consecutive input operations

Coding Hygiene

CS 101, 2025

Comments

- Comments are very useful in helping the reader understand the code
- Single-line comments using `//`:

```
int n = 4;  // n refers to the number of sides in a polygon
```

- Multi-line or block comments:

```
/* The code below computes the sum of n numbers
   Input: Reads n numbers - i_1,i_2,...,i_n
   Output: Prints out i_1 + i_2
*/
repeat(n) {
```

- Text between `//` and end of line, or between `/*` and `*/` is ignored by the compiler

Image from: https://xkcd.com/1513/

A few good coding styles/practices (e.g., [1])

- Common to use spaces and not tabs for indentation
- Use blank lines to separate sections within a file
- Choose one of the following styling of braces, and use consistently:

```
repeat(n) {
    body
}
```

OR

```
repeat(n)
{
    body
}
```

- Variable names must be meaningful (as far as possible); multiple word variables typically separated by "`_`" (e.g., `light_on`)
- Typically initialize (if at all) variables as part of definition, and not as a separate statement. That is, `int i = 5;` preferred over `int i; i = 5;`
- Or, ignore style guides and write obfuscated C code! [2]

---

[1] https://google.github.io/styleguide/cppguide.html
[2] https://www.ioccc.org/

Going from program statement to code

CS 101, 2025

Problem statements

Q. Write C++ code to calculate $\sin(x)$ using the Taylor series expansion (where x is in radians):

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

Ask for x (in degrees) from the user and use a fixed number of terms. You can use `PI` (offered by `simplecpp`) to access the value of π.

[Easier Q]. Write C++ code to calculate the following series: $1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \ldots$

Next class: Variables, Operators, Data types

CS 101, 2025