# Programming (general)

## Computer program basics

* A computer program consists of instructions executing one at a time. Basic instruction types are:
  + **Input**: A program gets data, perhaps from a file, keyboard, touchscreen, network, etc.
  + **Process**: A program performs computations on that data, such as adding two values like x + y.
  + **Output**: A program puts that data somewhere, such as to a file, screen, network, etc.
* Programs use **variables** to refer to data
  + The name is due to a variable's value varying as a program assigns a variable like x with new values.

## Computational thinking

* Mathematical thinking became increasingly important throughout the industrial age to enable people to successfully live and work.
  + In the information age, many people believe **computational thinking**, or creating a sequence of instructions to solve a problem, will become increasingly important for work and everyday life.
* A sequence of instructions that solves a problem is called an **algorithm**

# Programming basics

## A first program

* A program starts in **main()**, executing the statements within main's braces { }, one at a time.
* Each statement typically appears alone on a line and ends with a **semicolon**, as English sentences end with a period.
* The **int** **wage** statement creates an integer variable named **wage**. The wage = 20 statement assigns wage with 20.
* The **cout** statements output various values.
* The **return 0** statement ends the program (the 0 tells the operating system the program ended without error).
* **Code** is the textual representation of a program

### Example

| #include <iostream>  using namespace std;  int main() {  int wage;  wage = 20;  cout << "Salary is ";  cout << wage \* 40 \* 52;  cout << endl;  return 0;  } |
| --- |

## Basic input

* The following statement gets an input value and puts that value into variable x: **cin >>** **x;** cin is short for characters in.

### Example of cin

| #include <iostream>  using namespace std;    int main() {  int wage;    cin >> wage; //the users input to the variable wage    cout << "Salary is ";  cout << wage \* 40 \* 52;  cout << endl;    return 0;  } |
| --- |

## Basic output: Text

* The **cout** construct supports output; **cout** is short for characters out. Outputting text is achieved via: **cout << "desired text";**. Text in double quotes " " is known as a **string literal**.
  + Multiple cout statements continue printing on the same output line.
* The statement **cout << endl;** starts a new output line, called a **newline**.
  + Note endl is short for "end line". A common error is to type the number "1" or a capital I as in "in", instead of a lower case l as in "end line".

## Outputting a variable's value

* Outputting a variable's value is achieved via: **cout << x;**
* Programmers commonly use a single output statement for each line of output by combining the outputting of text, variable values, and a new line.

### Outputting multiple items with one statement

* + The programmer simply separates the items with **<<** symbols. Such combining can improve program readability because the program's code corresponds more closely to the program's output.

### Example

| #include <iostream>  using namespace std;  int main() {  int wage;  wage = 20;  cout << "Wage is: " << wage << endl;  cout << "Goodbye." << endl;  return 0;  } |
| --- |

* A new output line can also be produced by inserting **\n**, known as a newline character, within a string literal.

# Comments and whitespace

## Comments

* A **comment** is text a programmer adds to code, to be read by humans to better understand the code but ignored by the compiler. Two common kinds of comments exist:
  + A **single-line comment** starts with **//** and includes all the following text on that line. Single-line comments commonly appear after a statement on the same line.
  + A **multi-line comment** starts with **/\*** and ends with **\*/**, where all text between **/\* and \*/** is part of the comment.
    - A multi-line comment is also known as a **block comment**.

## Whitespace

* **Whitespace** refers to blank spaces (space and tab characters) between items within a statement and blank lines between statements (called newlines).
  + A compiler ignores most whitespace.

### Example of good Whitespace

| #include <iostream>  using namespace std;  int main() {  int myFirstVar; // Aligned comments yield less  int yetAnotherVar; // visual clutter  int thirdVar;    // Above blank line separates variable declarations from the rest  cout << "Enter a number: ";  cin >> myFirstVar;    // Above blank line separates user input statements from the rest  yetAnotherVar = myFirstVar; // Aligned = operators  thirdVar = yetAnotherVar + 1;  // Also notice the single-space on left and right of + and =  // (except when aligning the second = with the first =)  cout << "Final value is " << thirdVar << endl; // Single-space on each side of <<    return 0; // The above blank line separates the return from the rest  } |
| --- |

# Errors and Warnings

## Syntax Errors

* **Syntax error** - is to violate a programming language's rules on how symbols can be combined to create a program.

### Example

| main.cpp:6:27: error: expected ';' after expression  cout << "Traffic today"  ^  ; |
| --- |

## Unclear error messages

* Sometimes errors can be wrong, like it might say there is a missing “;” but really there's a missing “<<” just as an example. Look through the code to verify. Look at the code before it says where the error was.
  + Focus on the FIRST error message, ignoring the rest.
  + Look at the reported line of the first error message. If an error is found, fix it. Else, look at the previous few lines.
  + Compile, repeat.

## Logic errors

* A syntax error is known as a type of **compile-time error.**
* A **logic error**, also called a **bug**, is an error that occurs while a program runs.
  + The program would compile but would not run as intended.
    - Writing many lines of code without compiling and running is bad practice.
    - New programmers should compile and run programs after every few lines. Even experienced programmers compile and run frequently.

## Compiler warnings

* A compiler will sometimes report a **warning**, which doesn't stop the compiler from creating an executable program but indicates a possible logic error.

# Computers and programs (general)

## Switches

* A **switch** controls whether or not electricity flows through a wire.
  + In an electronically controlled switch, a positive voltage at the control input allows electricity to flow, while a zero voltage prevents the flow.
  + Engineers soon realized they could use electronically controlled switches to perform simple calculations.
  + The engineers treated a positive voltage as a "1" and a zero voltage as a "0".
    - 0s and 1s are known as **bits (binary digits)**.
  + They built connections of switches, known as **circuits**, to perform calculations such as multiplying two numbers.

## Processors and memory

* To support different calculations, circuits called **processors** were created to process (aka execute) a list of desired calculations, with each calculation called an **instruction**.
* A **memory** is a circuit that can store 0s and 1s in each of a series of thousands of addressed locations, like a series of addressed mailboxes that each can store an envelope (the 0s and 1s).

## Instructions

* Below are some sample types of instructions that a processor might be able to execute, where X, Y, Z, and num are each an integer.

### Example

| **Add X, #num, Y** | Adds data in memory location *X* to the number *num*, storing the result in location *Y*. |
| --- | --- |
| **Sub X, #num, Y** | Subtracts *num* from data in location *X*, storing the result in location *Y*. |
| **Mul X, #num, Y** | Multiplies data in location *X* by *num*, storing result in location *Y*. |
| **Div X, #num, Y** | Divides data in location *X* by *num*, storing the result in location *Y*. |
| **Jmp Z** | Tells the processor that the next instruction to execute is in memory location *Z*. |

* Memory stores instructions and data as 0s and 1s.
* The material will commonly draw the memory with the corresponding instructions and data to improve readability.
* The programmer-created sequence of instructions is called a **program**, **application**, or just **app**.

## Writing Computer Programs

* Instructions represented as 0s and 1s are known as machine instructions, and a sequence of **machine instructions,** together form an **executable program** (sometimes just called an **executable**).
* **Assemblers** to automatically translate human readable instructions, such as "Mul 97, #9, 98", known as **assembly** language instructions, into machine instructions.
  + The assembler program thus helped programmers write more complex programs.
* To support high-level languages, programmers **created compilers**, which are programs that automatically translate high-level language programs into executable programs.