Keyword or Reserved Word: word that has a special meaning to the compiler

1. Must be spelled correctly

2. Must be in lowercase

3. Cannot be used as variables

4. Examples from pgm1.cpp

   using
   namespace
   int
   return

Others to see soon:

   float
double
char
long
1. **Constant or Literal**: data representing itself; data appearing literally in the program

   ```cpp
   int first_num, second_num; // Declare variables
   ... cin >> first_num;       // Input data into
   ... cin >> second_num;     // the variables
   ...
   sum = first_num + second_num; // Reference the data
                                 // using the variable names
                                 
   second_num | 5
   first_num  | 15
      
   versus
      
   sum = 15 + 5;               // 15 and 5 are constants
      
   2. **Data Types**
      **Numeric** type can be manipulated arithmetically.
      a. **Integer** type – whole numbers without a decimal point
         b. **Floating-point** type – numbers with a decimal point

      **Non-numeric** or **string** type – string of characters; cannot be manipulated arithmetically
**Integer type constant:** number without a decimal point; has no fractional digits

1. Decimal point is assumed to be to right of rightmost digit.  15

2. Fixed-point constant; point refers to decimal point

3. Legal characters: digits and optional leading sign (+ or - to left of leftmost digit)

   No commas or special characters

<table>
<thead>
<tr>
<th>Invalid</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. 25,500</td>
<td>25500</td>
</tr>
<tr>
<td>b. 7%</td>
<td>7</td>
</tr>
<tr>
<td>c. $50</td>
<td>50</td>
</tr>
<tr>
<td>d. 88-</td>
<td>-88</td>
</tr>
<tr>
<td>e. 55.</td>
<td>55</td>
</tr>
</tbody>
</table>

4. Unsigned constants are assumed positive.

5. **short** range: -32768 to 32767
   **int** range: -2147483648 to 2147483647

6. Integers are the Natural Numbers which are the counting numbers, zero, and the negatives of natural numbers.

   -5 -4 -3 -2 -1 0 1 2 3 4 5
Floating-point type constant: number with a decimal point; Can have fractional digits

1. 1234. 12.34 12340. .01234
   123.4 1.234 123400. .001234

Decimal point can float to a desired position.

2. Legal characters: digits, 1 decimal point, and optional leading sign
   No commas or special characters

<table>
<thead>
<tr>
<th>Invalid</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. 25,500.25</td>
<td>25500.25</td>
</tr>
<tr>
<td>b. 7.25%</td>
<td>7.25</td>
</tr>
<tr>
<td>c. $59.99</td>
<td>59.99</td>
</tr>
<tr>
<td>d. 88.5-</td>
<td>-88.5</td>
</tr>
<tr>
<td>e. 55</td>
<td>55.</td>
</tr>
</tbody>
</table>

3. Scientific or e-notation:

23.45 in e-notation
   a. 23.45e0 means 23.45 * 10^0 23.45 * 1
   b. 2.345e1        2.345 * 10^1 2.345 * 10
   c. .2345e2        .2345 * 10^2 .2345 * 100
   d. 234.5e-1       234.5 * 10^-1 234.5 * 1 / 10
   e. 2345.e-2       2345. * 10^-2 2345. * 1 / 100
   f. 2345e-2

Must have decimal point or e-notation or both in floating-point type.

2. 2e0 2.e0
4. Unsigned constants are assumed positive.

5. float precision: 6 or 7 decimal digits  
   double precision: 15 digits  
   long double precision: 18 or 19 digits  
   1.2345678901234567890

6. All constants are stored in main memory.  
   cout << 2.0; // double  
   cout << 2.0F; // float  
   cout << 2.0L; // long double

   The F and L are known as a suffix.
Variable: symbolic name that denotes the contents of a memory location

1. Contents of a memory location can change or vary

2. A variable can have only one value at a time, the current value

   10   a = 5;
   20   a = 25;
   30   a = 62;

3. Variables are programmer-supplied.

4. Syntax rules for forming variables:
   
a. any combination of letters, digits, and the underscore character
   
b. first character must be a letter or underscore
      (Do not start with underscore.)
   
c. can be any length
   
d. cannot be a keyword

5. Variables are case-sensitive: sum SUM Sum suM

6. Case Style: sum_of_grades sumOfGrades
   
      first_num firstNum
7. **Valid Variables**
   - first_num
   - second_num
   - sum
   - difference
   - product
   - quotient
   - sum_of_grades
   - sum_of_10_grades
   - sumof10grades
   - sumOfGrades
   - voidCheck
   - interval

   **Invalid Variables**
   - 2nd_num
   - num-1
   - rate*time
   - total.salary
   - int
   - void

8. Should use meaningful (descriptive) variables
   a. a and b
   b. tax1 and tax2
   c. state_tax and fed_tax

9. Length of variables:
   - `average_midterm_grade_for_CS_110A` 33 characters
   - `ave_grade`

   8 to 15 characters is a good length. Your eyes shouldn't need to scan a variable to read it.
1. Variables must be declared before they are used; At the top of the function (block) for now

2. Declaration Statements:

14 int first_num, second_num; // From pgm1.cpp
15 int sum, difference, product, quotient;

The statement terminator is a semicolon.

General Form: type list of comma-separated variables;

Type indicates the type of data to be stored.

20 int first_num; // An alternate form for pgm1.cpp
30 int second_num;
40 int sum;
50 int difference;
...

more examples:
60 long student_count; // Note the semicolon
70 float interest_rate;
80 double annual_salary;
90 long double national_debt;

3. Can give an initial value.

int maximum = 10000, minimum = -10000; or
int maximum(10000), minimum(-10000);

4. Defining declaration: Memory is allocated as a result of the declaration.
5. Variable declaration tells the computer how much memory to allocate for the stored data and what internal format to use.

Integer and floating-point types have a different internal binary format.

200 versus 200.0
int double

```c
int i = 200;
double d = 200.0;
```

variable i:
4 bytes [00000 00200]

variable d:
8 bytes [3 \overset{\text{Exponent and Mantissa}}{\wedge} 20000 00000 00000]

The above means .2 x 10^3
1. **Arithmetic Operators:**
   a. + addition      \( a + b \) => sum
   b. - subtraction   \( a - b \) => difference
   c. * multiplication \( a \times b \) => product
   d. / division      \( a / b \) => quotient
   e. % modulus       \( a \% b \) => modulo (integers only)

2. **Arithmetic Expression:** combination of numeric constants and/or variables joined by arithmetic operators, representing a numeric value.

   3 Forms:
   - single constant
   - single variable
   - combination of constants and/or variables

   ```
   10 int a, b, c;
   20 a = 5;       // constant
   30 b = a;       // variable
   40 c = a + b - 2; // combination
   ```

   ![Table]

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>8</td>
</tr>
<tr>
<td>b</td>
<td>5</td>
</tr>
<tr>
<td>a</td>
<td>5</td>
</tr>
</tbody>
</table>

3. **General Form:** single variable = arithmetic expression;

   **Action:**
   a. expression is evaluated
   b. value of the expression is assigned to the variable to the left of =

4. \( c = c + 1; \)

5. \( 3 = a; \)

   \( a + b = c; \)
1. **Input stream:** stream of bytes flowing from keyboard to computer

2. General form:  
   ```
   cin >> variable >> variable ... ;
   ```

   **Extraction operator:** `>>`

   Extract data from the input stream

3. ```
   int a, b;
   
   cout << "Enter two numbers: ";
   cin >> a >> b; // or cin >> a >> b;
   ```

   a. Data is buffered (Entered characters are stored temporarily in a buffer.)
      **Buffer:** a temporary storage area

   b. `<Enter>` key places a newline character in the input buffer and makes the data available to program

   c. User can backspace and make corrections before pressing `<Enter>`

   d. Multiple data items must be separated by whitespace which is any combination of spaces, newlines and tabs.

   e. Leading whitespace is ignored.

   **Buffer contents**

<table>
<thead>
<tr>
<th>Example</th>
<th>Buffer contents</th>
</tr>
</thead>
</table>
   | 1: 5^15<Enter> | 5^15
   | 2: ^^^^^5^^^^^15<Enter> | ^^^^^5^^^^^15
   | 3: 5<Enter> 15<Enter> | 5

   Better style: Code one variable for each `cin`.

   More user-friendly; No data separator required.

   ```
   cout << "Enter first number: ";
   cin >> a;
   cout << "Enter second number: ";
   cin >> b;
   ```
4. More examples:

```cpp
int a;

cin >> a;
```

<table>
<thead>
<tr>
<th>Buffer contents</th>
<th>Stored value</th>
</tr>
</thead>
<tbody>
<tr>
<td>250\n</td>
<td></td>
</tr>
<tr>
<td>~250\n</td>
<td></td>
</tr>
<tr>
<td>250A\n</td>
<td></td>
</tr>
<tr>
<td>A250\n</td>
<td></td>
</tr>
</tbody>
</table>

```cpp
double d;
```

```cpp
cin >> d;
```

<table>
<thead>
<tr>
<th>Buffer contents</th>
<th>Stored value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.5\n</td>
<td></td>
</tr>
<tr>
<td>~2.5\n</td>
<td></td>
</tr>
<tr>
<td>2.5A\n</td>
<td></td>
</tr>
<tr>
<td>A2.5\n</td>
<td></td>
</tr>
<tr>
<td>2.5^3\n</td>
<td></td>
</tr>
</tbody>
</table>
1. **Output stream:** stream of bytes flowing from program to screen

2. General form: `cout << item1 to be printed << item2 to be printed … ;`

   `item to be printed` can be:
   - variable
   - constant
   - expression

   **Insertion operator:** `<<`
   Insert the item to be printed into the output stream.

   ```
   int ten = 10;
   ```

   ```
   cout << ten << "^minus^" << 2 << "^equals^" << (ten - 2) << "\n";
   ```

   Output from above **cout:** `10^minus^2^equals^8`

3. **String constant:** any series of characters enclosed in double quotes

4. Newline character is represented by `\n`.

   ```
   cout << "Hi There\n"; // Always used in a string constant
   ```

5. **endl** (end line) is a stream manipulator

   ```
   cout << "Hi There" << endl; // Follows insertor
   ```