

# Glossary

<https://csci-1301.github.io/about#authors>

January 17, 2023 (03:24:10 PM)

## Contents

**Keywords/Reserved Words**<sup>1</sup>: words defined by the C# language and used for one thing

**Datatypes**<sup>2</sup>: categories in C# used to define types of values, such as strings

**Variable**<sup>3</sup>: values that be changed

**Constant**<sup>4</sup>: values that can not be changed

**Identifier**<sup>5</sup>: words defined by the programmer to refer to an object or variable.

**Operations:**

**Operators**<sup>6</sup>: symbols used to perform operations

**Modulo**<sup>7</sup>: the % used to divide two numbers and return the remainder

**Escape Sequence**<sup>8</sup>: used to represent a non-printable character

**Reference Types (Objects and Strings)**<sup>9</sup>: a variable of a class object holds a reference to the address of the object on the managed heap.

**Value Types (all other reserved words)**<sup>10</sup>: a variable of a class object stores the exact data value held by the variable

**Numeric Types**<sup>11</sup>:

**Booleans**<sup>12</sup>: a binary datatype that can only be true or false

Decision Structures (**if/else/switch**)<sup>13</sup>:

Control Structures (loops)<sup>14</sup>:

**Instantiation (instance of a class)**<sup>15</sup>: the act of creating a object, an instance of a C# class

**Initialization**<sup>16</sup>: the act of both declaring a variable's datatype and identifier and assigning it value

**Declaration**<sup>17</sup>: the act of creating a variable's datatype and identifier

**Assignment**<sup>18</sup>: the act of giving a value to an identifier

**Implicit Conversion**<sup>19</sup>: the act of automatically storing the value of one identifier into another identifier that differs from its own

**Explicit Conversion (casting)**<sup>20</sup>: the act of storing the value of one identifier into another identifier that differs from its own using additional syntax

**Rules**<sup>21</sup>: are required syntactical ways to write a program for it to function Conventions<sup>22</sup>:

are not required for the program to function, but are heavily encouraged for the readability and comprehension of other programmers

**Format Specifiers (C, N, P, E)**<sup>23</sup>: added to variable calls in strings to format the numeric variable in various ways (see link for specifics)

**Constructor**<sup>24</sup>: a method used to instantiate an object and assign it's attributes

**Parameter**<sup>25</sup>: any variable declared within a method

**Argument**<sup>26</sup>: any value that must passed to a method in order for it to be called

---

<sup>13</sup><https://csci-1301.github.io/book.html#decisions-and-decision-structures>

<sup>14</sup><https://csci-1301.github.io/book.html#loops-increment-operators-and-input-validation>

<sup>22</sup><https://csci-1301.github.io/book.html#conventions-of-c-programs>

**Attribute**<sup>27</sup>: the variables declared within a class to act as the characteristics of any of its instantiated object

**Method**<sup>28</sup>: a code block that contains a series of statements

**Class Member (attributes and methods)**<sup>29</sup>: anything defined within a class that can be access within and outside of the class

**Scope**<sup>30</sup>: Time and place in program where the variable exists

**Iterator**: an object that traverses an array or list

**Sentinel Value**<sup>31</sup>: a special value in the context of an algorithm which uses its presence as a condition of termination, typically in a loop or recursive algorithm.

**Guard Condition**: boolean expressions (predicates) found at the top of a method or function that determine whether the function should continue to run.

**Accumulator**<sup>32</sup>:

**Counter**<sup>33</sup>: a variable used to count the number of times a certain condition is met

**Complex Condition**<sup>34</sup>: a condition consisting of multiple conditions

**Method Signature**<sup>35</sup>: the way a computer reads a method by its name and the datatype of its parameters

**Method Overloading**<sup>36</sup>: the act of creating multiple methods with the same signature

**Return Type**<sup>37</sup>: the datatype of any value returned from a called method

**UML Diagram**<sup>38</sup>: a written diagram used to display a class and all of its members

**Input Validation**<sup>39</sup>: whenever a program checks if the user gave a usable input and responds accordingly to avoid errors implementation