Dominick Girard

CSCI 2503 – Survey of Programming Language

Operator Overloading

Operator overloading is a very good feature, that not only allows developers to create more natural and expressive code but, makes the code more readable. It also becomes useful when we get to upkeep, as operators can be redefined to align closely with the specific problems you have. This can improve the clarity and intention of the code. Furthermore, when operators are overloaded in a way that is consistent, it can make the code more intuitive to those familiar with the language.

Method Parameter Passing Methods

Part A

All the same because passing the value does not change the variables value.

1) value = 2, list = { 1, 3, 5, 7, 9 }
2) value = 2, list = { 1, 3, 5, 7, 9 }
3) value = 2, list = { 1, 2, 5, 7, 9 }

Part B

The swap method does work and swaps the variables references using ref.

1) value = 1, list = { 2, 3, 5, 7, 9 }
2) value = 1, list = { 3, 2, 5, 7, 9 }
3) value = 5, list = { 3, 2, 2, 7, 9 }

Part C

Same as part B but using ref and out.

1) value = 1, list = { 2, 3, 5, 7, 9 }
2) value = 1, list = { 3, 2, 5, 7, 9 }
3) value = 5, list = { 3, 2, 5, 7, 9 }

Ada subprograms

Similarities

- Both Ada and C-based languages allow you to define functions and procedures.
- They both support subprograms that can return values (functions).
- Both languages provide options for passing parameters by value or reference.
- Both languages allow the definition of local variables within subprograms.

Differences

- Ada supports function overloading, allowing you to define multiple functions with the same name but different parameter lists, different return types, or both.
- Ada provides explicit parameter modes (in, out, in out) to specify whether parameters are read-only, write-only, or both. This ensures better parameter control and safety.
- C languages primarily use pass-by-value or pass-by-reference (through pointers), and parameter modes are not as explicitly defined.

Default parameter values

1) I see this being useful as it provides a convenient way for the caller to use the function without having to specify values for every parameter, especially when they want to stick with the default behavior.

2) Another interesting useful behavior I found with this is when you introduce new parameters to a function, you can set them as optional with default values to maintain compatibility with existing code that doesn't provide these new arguments.

Function pointers

1) This can be useful in a variety of ways, but I thought it to be helpful when employed to customize algorithms or data processing methods, allowing users to provide their own functions to tailor the behavior.

2) Another use of Function pointers is commonly used in event-driven systems where you want to allow users or components to specify custom actions or behavior without changing the underlying code.