Software Engineering

Good software

❑ Good software has:

1. A good design document.
2. Well designed components.
3. Safe, clean and well structured code.
4. Fitting data structures and good algorithms.
5. Good test matrix.

Coding Guidelines

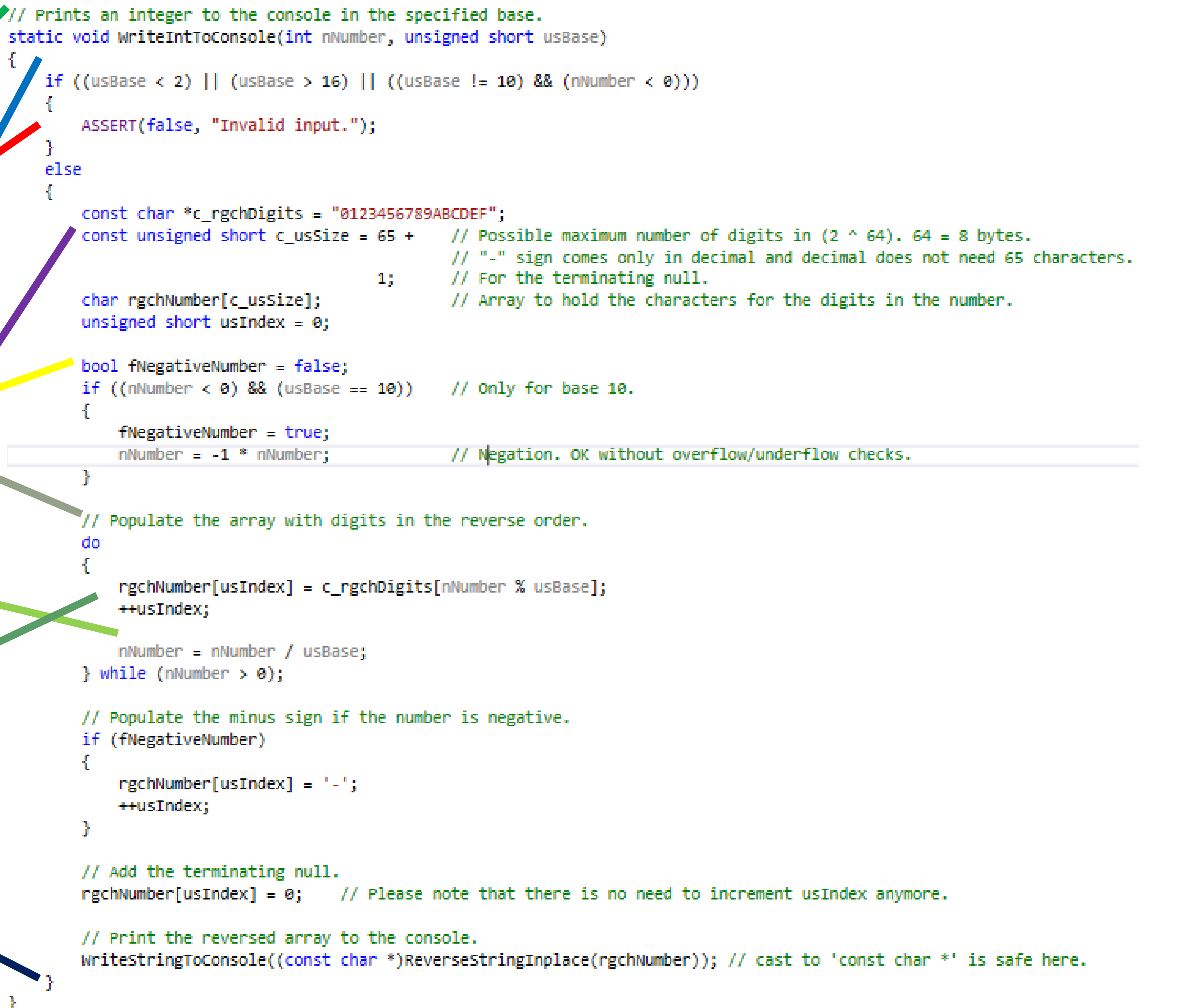
- Programming is the art of writing essays in crystal clear prose and making them executable

  Dr. Per Brinch Hansen

- Coding standards
  - Structuring
    - Function signatures.
    - Function length.
  - Formatting
    - Consistency.
    - Indentation, whitespace.
    - Readability and maintainability.
  - Commenting
    - Public interfaces and difficult-to-understand steps should be commented.
    - Don’t be too verbose. Don’t comment the obvious.

Sample code

- Comment the function. (Xml style, more detailed, comment parameters/returns).
- Parameter validation.
- Comment where required.
- Do **not** comment the obvious.
- Empty lines to reduce clutter.
- Function signatures. HRESULTs versus Booleans versus voids.
- Language features: const and the like.
- Conventions: Hungarian notations and the like.
- Fits in one page / monitor screen.



```
// Prints an integer to the console in the specified base.
static void WriteIntToConsole(int nNumber, unsigned short usBase)
{
    if ((usBase < 2) || (usBase > 16) || ((usBase != 10) && (nNumber < 0)))
    {
        ASSERT(false, "Invalid input.");
    }
    else
    {
        const char *c_rgchDigits = "0123456789ABCDEF";
        const unsigned short c_usSize = 65 +    // Possible maximum number of digits in (2 ^ 64). 64 = 8 bytes.
                                                // "-" sign comes only in decimal and decimal does not need 65 characters.
                                        1;      // For the terminating null.
        char rgchNumber[c_usSize];              // Array to hold the characters for the digits in the number.
        unsigned short usIndex = 0;

        bool fNegativeNumber = false;
        if ((nNumber < 0) && (usBase == 10))    // Only for base 10.
        {
            fNegativeNumber = true;
            nNumber = -1 * nNumber;             // Negation. OK without overflow/underflow checks.
        }

        // Populate the array with digits in the reverse order.
        do
        {
            rgchNumber[usIndex] = c_rgchDigits[nNumber % usBase];
            ++usIndex;

            nNumber = nNumber / usBase;
        } while (nNumber > 0);

        // Populate the minus sign if the number is negative.
        if (fNegativeNumber)
        {
            rgchNumber[usIndex] = '-';
            ++usIndex;
        }

        // Add the terminating null.
        rgchNumber[usIndex] = 0;    // Please note that there is no need to increment usIndex anymore.

        // Print the reversed array to the console.
        WriteStringToConsole((const char *)ReverseStringInplace(rgchNumber)); // cast to 'const char *' is safe here.
    }
}
```

Coding Guidelines Revisited

- Coding standards
  - Structuring
    - Function signatures.
    - Function length.
  - Formatting
    - Consistency.
    - Indentation, whitespace.
    - Readability and maintainability.
  - Commenting
    - Public interfaces and difficult-to-understand steps should be commented.
    - Don’t be too verbose. Don’t comment the obvious.

Specs and design docs

- First phase is a specification document (spec), followed by a design document
- Writing is a rigorous test of simplicity: It is just not possible to write convincingly about ideas that cannot be understood

Dr. Per Brinch Hansen

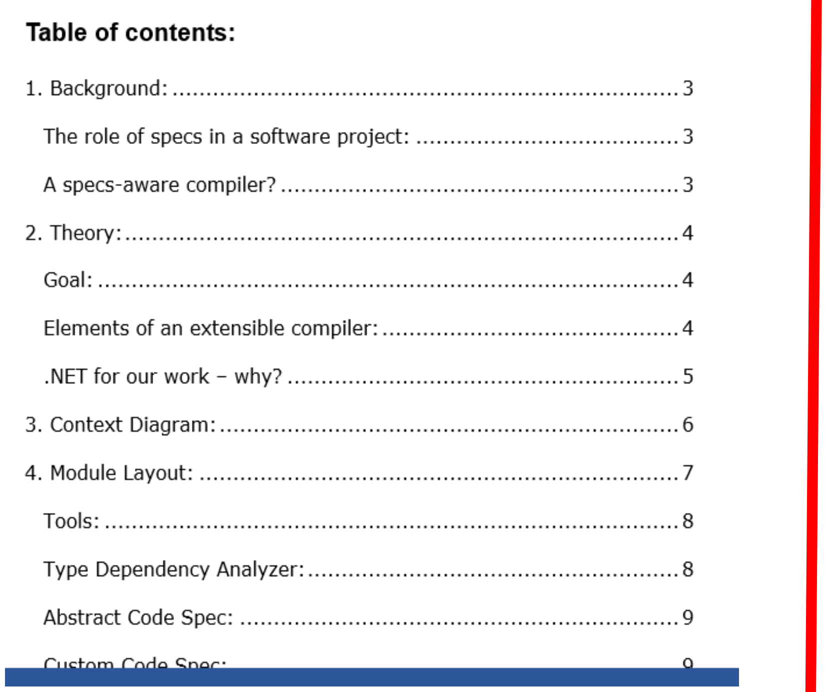
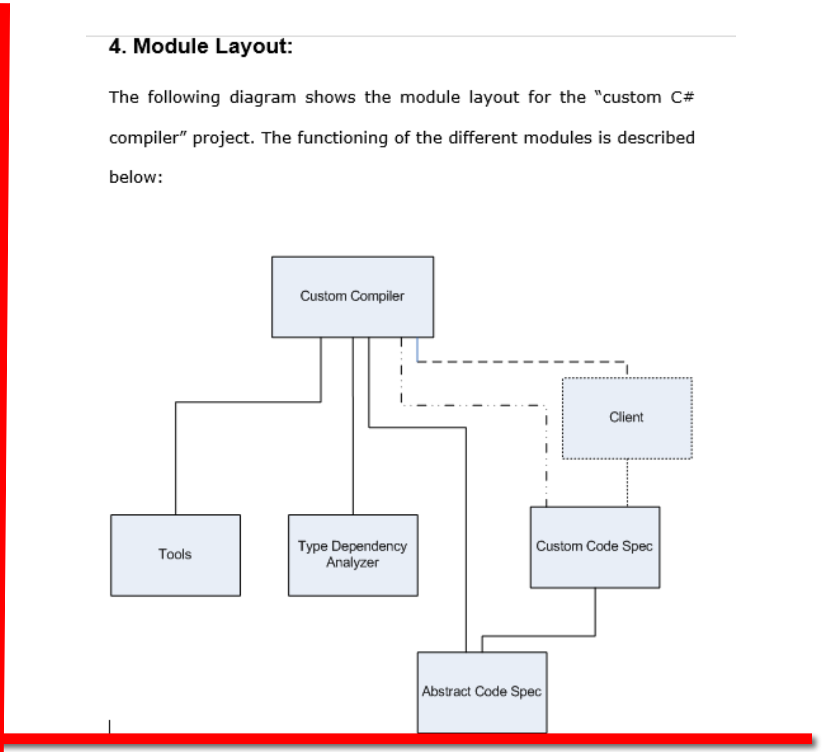
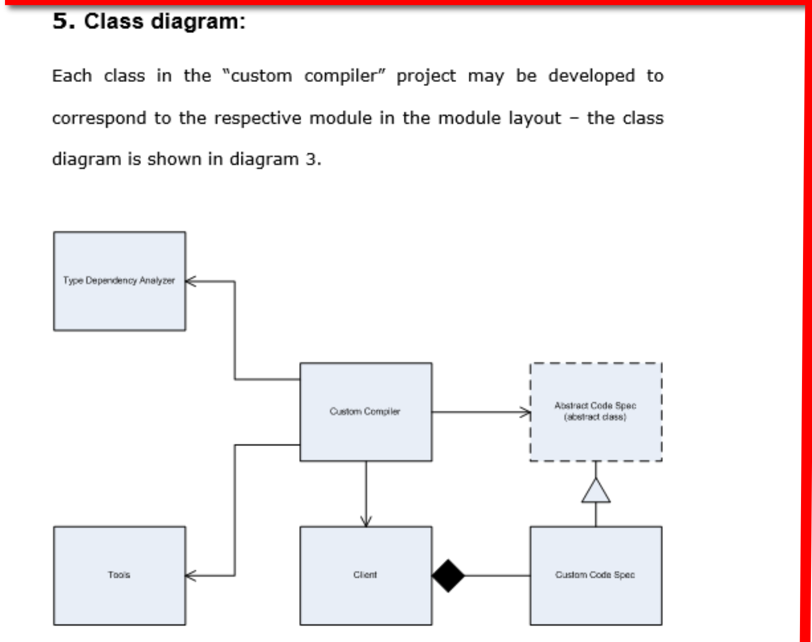


Table of contents:



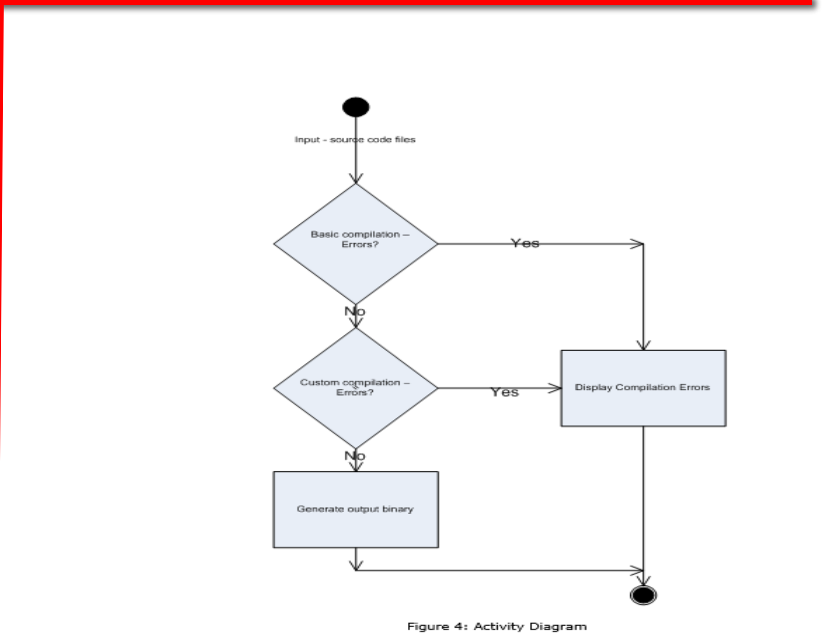
4. Module Layout:

The following diagram shows the module layout for the "custom C# compiler" project. The functioning of the different modules is described below:



5. Class diagram:

Each class in the "custom compiler" project may be developed to correspond to the respective module in the module layout – the class diagram is shown in diagram 3.



Input - source code files
Basic compilation – Errors?
Yes
No
Custom compilation – Errors?
Yes
Display Compilation Errors
No
Generate output binary

Figure 4: Activity Diagram

Design Guidelines

- Extensibility
- Pluggabilty
- Testability

Software Contracts

- What is a software contract?
  - Car <-> Axle <-> Wheel
  - Door <-> Hinge <-> Wall
- Why do we need a contract?
  - Componentization
- How do we specify a contract?
  - interface
- Example:
  - **User interface** <-> **Message** <-> **Communication channel**.
  - Contract is simple. ***Send, Receive***.
    - Send takes in a message and address.
    - Receive provides message dynamically.
    - Caller is agnostic to implementation. Communication channel can use sockets, named pipes, tcp, http etc. UI (caller) does not care.

UML

- A picture speaks a thousand words.

Appendix

QUESTIONS?