**ASSIST/UNA: A Windows-Based Emulator**

**of the ASSIST/I Assembler for the IBM/360**

Software Requirements Specification

Version 1.2.1

February 28, 2014

Travis Hunt, *Team Leader*

Michael Beaver, *Technical Writer*

Andrew Hamilton, *Software Quality Assurance*

Drew Aaron

Clay Boren

Chad Farley

University of North Alabama

CS 455: Software Engineering

Dr. Patricia L. Roden

Spring 2014

# Version History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version Number** | **Author(s)** | **Comments** |
| 02/05/2014 | Version 1.0.0 | Michael Beaver | Initial draft |
| 02/06/2014 | Version 1.0.1 | Michael Beaver | Added definitions, acronyms, and abbreviations;  Added project scope;  Added references;  Added user characteristics;  Added change management process; Added assembly instructions appendix;  Removed unnecessary sections |
| 02/07/2014 | Version 1.0.2 | Michael Beaver | Added product perspective;  Added product functions;  Added assumptions and dependencies;  Added functional requirements;  Added use cases;  Added design constraints;  Added other requirements;  Added diagrams to appendix;  Added prototype screen captures;  Updated definitions, acronyms, and abbreviations;  Updated references;  Removed unnecessary sections |
| 02/08/2014 | Version 1.0.3 | Michael Beaver | Added introductory overview;  Added general constraints;  Added external interface requirements;  Added classes/objects;  Added future features appendix;  Updated references;  Updated assembly instructions appendix |
| 02/13/2014 | Version 1.0.4 | Michael Beaver | Updated the introduction;  Updated the general description;  Updated definitions, acronyms, and abbreviations;  Updated user characteristics;  Updated specific requirements;  Updated appendices |
| 02/14/2014 | Version 1.0.5 | Michael Beaver | Updated purpose;  Updated definitions, acronyms, and abbreviations;  Updated user characteristics;  Updated specific requirements;  Updated functional requirements;  Updated non-functional requirements;  Updated deliverables;  Updated appendices;  General editing |
| 02/16/2014 | Version 1.1.0 | Michael Beaver | Updated definitions, acronyms, and abbreviations;  Updated product functions;  Updated user characteristics;  Updated assumptions and dependencies;  Updated external interface requirements;  Updated functional requirements;  Updated use cases;  Updated classes and objects;  General editing |
| 02/18/2014 | Version 1.1.1 | Michael Beaver | Add two-pass assembly appendix;  Added errors appendix;  Added .PRT appendix;  Added testing plan;  Updated references;  Removed diagrams appendix |
| 02/19/2014 | Version 1.1.2 | Michael Beaver | Added ASSIST/I assembler options appendix;  Updated definitions, acronyms, and abbreviations;  Updated general constraints;  Updated specific requirements;  Updated performance requirements;  Updated design constraints;  Updated screen captures appendix;  General edits |
| 02/21/2014 | Version 1.1.3 | Michael Beaver  Andrew Hamilton | Added more section cross-referencing;  Updated testing plan;  Updated errors appendix;  General edits |
| 02/22/2014 | Version 1.1.4 | Michael Beaver | Updated definitions, acronyms, and abbreviations;  Updated specific requirements |
| 02/27/2014 | Version 1.2.0 | Michael Beaver | Updated specific requirements;  Updated .PRT format appendix;  General edits |
| 02/28/2014 | Version 1.2.1 | Michael Beaver  Andrew Hamilton | Updated testing plan |

**Table of Contents**

Version History i

1. Introduction 1

1.1 Purpose 1

1.2 Scope 1

1.3 Definitions, Acronyms, and Abbreviations 1

1.4 References 3

1.5 Overview 3

2. General Description 4

2.1 Product Perspective 4

2.2 Product Functions 4

2.3 User Characteristics 4

2.4 General Constraints 4

2.5 Assumptions and Dependencies 5

3. Specific Requirements 5

3.1 External Interface Requirements 5

3.1.1 User Interfaces 5

3.1.2 Software Interfaces 6

3.2 Functional Requirements 7

3.2.1 Source Code Editing 7

3.2.2 Source Code Assembly 8

3.2.3 Source Code Assembly and Debugging 8

3.2.4 Source Code Assembly and Final Run 9

3.2.5 Report Viewing and Printing 10

3.3 Use Cases 10

3.3.1 Create a New Project 10

3.3.2 Save a Project 10

3.3.3 Open a Project 11

3.3.4 Assemble a Program 11

3.3.5 Assemble a Program and Debug 11

3.3.6 Assemble a Program and Perform a Final Run 12

3.3.7 View a .PRT File 12

3.3.8 Print a .PRT File 12

3.3.9 Print Program Source Code 13

3.4 Non-Functional Requirements 13

3.4.1 Performance 13

3.4.2 Reliability 13

3.4.3 Licensing 13

3.4.4 Security 13

3.4.5 Maintainability 14

3.4.6 Portability and Delivery 14

3.5 Design Constraints 14

3.6 Other Requirements 14

4. Testing Plan 15

5. Deliverables 15

6. Change Management Process 15

6.1 Email Report Guidelines 16

7. Client-Developer Contractual Agreement 17

A. Appendices 18

A.1 Assembly Instructions to Implement 18

A.2 The Two-Pass Assembly Process 20

A.3 The .PRT File Format 21

A.4 Supported Assembler Errors 21

A.5 ASSIST/I Assembler Options 23

A.6 Prototype Graphical User Interface Screen Captures 24

A.7 Features for Future Consideration 30

# 1. Introduction

This document contains all the software requirements and specifications for the ASSIST/UNA software package. The software development team and the client will both sign off and agree to the terms and guarantees listed in this document (see Section 7).

This section provides an overview of what all is contained in this SRS document.

## 1.1 Purpose

The ASSIST/UNA software is primarily designed to enable students to learn and practice the ASSIST assembly language for the IBM/360. The ASSIST/UNA software will replace the current ASSIST/I software emulated with DOSBox. Students and instructors of CS 310 and possibly CS 311 will be using this software.

## 1.2 Scope

The ASSIST/UNA software is a Windows-based emulator of the ASSIST assembly language that will enable students to write assembly programs in a graphical environment. The ASSIST/UNA software will provide students with a subset of the functionality of the ASSIST/I assembler (see Sections 2 and 3). At this time, the ASSIST/UNA software is not meant to *fully* emulate all of the features of the ASSIST/I assembler; however, these features may be implemented in the future (see Appendix A.7).

The ASSIST/UNA software will be used by students and instructors in future CS 310 and possibly CS 311 courses. This port to the Windows environment eliminates the need to use other emulating software, such as DOSBox, to write assembly programs for the IBM/360. The ASSIST/UNA software will enable students to interact with an emulated ASSIST/I assembler in an intuitive and user-friendly environment.

## 1.3 Definitions, Acronyms, and Abbreviations

This section provides definitions of all terms, acronyms, and abbreviations required to properly interpret this software requirements specification document.

**1.3.1 ASCII:** American Standard Code for Information Interchange character encoding.

**1.3.2** **Assemble:** To translate source statements into corresponding machine code in the form of an object code program. See Appendix A.2 for a description of the two-pass assembly process that ASSIST/I and ASSIST/UNA follow.

**1.3.3 Assembler:** A piece of software that assembles assembly source statements.

**1.3.4 ASSIST:** The Assembler System for Student Instruction and Systems Teaching for the IBM/360 developed by John Mashey and his students at Pennsylvania State University.

**1.3.5 ASSIST/I:** The version of ASSIST for personal computers.

**1.3.6 ASSIST/UNA:** The University of North Alabama emulator of the ASSIST/I assembler.

**1.3.7 Backend:** The ASSIST/UNA backend is comprised of the ASSIST/I simulator and supporting components (e.g., .PRT file generator).

**1.3.8** **Client:** The person that has instigated development and has decided the project requirements. The current client is Dr. Patricia L. Roden of the University of North Alabama.

**1.3.9 CS 155:** The Computer Science 155: Computer Science I course taught at the University of North Alabama.

**1.3.10 CS 245:** The Computer Science 245: Introduction to Discrete Structures course taught at the University of North Alabama.

**1.3.11 CS 310:** The Computer Science 310: Computer Organization & Assembly Language Programming course taught at the University of North Alabama.

**1.3.12 CS 311:** The Computer Science 311: Computer Architecture course taught at the University of North Alabama.

**1.3.13 Developer:** The person, or persons, who actively design, implement, and maintain the ASSIST/UNA software and the accompanying documentation.

**1.3.14 DOSBox:** A Windows program that emulates the Windows DOS environment.

**1.3.15 EBCDIC:** Extended Binary Coded Decimal Interchange Code character encoding.

**1.3.16 Emulator:** Software on a hardware system that imitates the functionality of other software on its respective hardware system.

**1.3.17 Frontend:** The ASSIST/UNA backend is comprised of the visual GUI components that represent and communicate data from the backend assembler.

**1.3.18 GitHub:** A repository service used to store remote copies of project source code and documentation.[[1]](#footnote-1)

**1.3.19 GUI:** Graphical User Interface. See Appendix A.6 for screen captures of the ASSIST/UNA prototype GUI.

**1.3.20 IBM:** International Business Machines.

**1.3.21 IBM/360:** A family of mainframes designed by IBM.

**1.3.22 IDE:** Integrated Development Environment.

**1.3.23 .NET Framework:** A programming framework developed by Microsoft. This project will be based on the .NET Framework version 4.5.

**1.3.24 .PRT File:** A program report file created by ASSIST/I and ASSIST/UNA software. See Appendix A.3 for the general .PRT file format.

**1.3.25 SRS:** Software Requirements Specification document.

**1.3.26 Team Foundation Server (TFS):** A Microsoft Visual Studio version-control system used to maintain and ensure the integrity of project source code.[[2]](#footnote-2)

**1.3.27 UNA:** The University of North Alabama in Florence, Alabama.

**1.3.28 User:** The person, or persons, who interact directly with the ASSIST/UNA software.

**1.3.29 Windows:** The Microsoft Windows operating system environment.

## 1.4 References

This SRS shall be used in conjunction with the following publications and client handouts.

Client, Assembly Instructions to Implement.[[3]](#footnote-3)

Client, Appendix D: Errors and Their Causes.[[4]](#footnote-4)

Client, Initial Project Description.[[5]](#footnote-5)

Client, More Detailed Description.[[6]](#footnote-6)

IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications.[[7]](#footnote-7)

Microsoft, .NET Framework System Requirements.[[8]](#footnote-8)

## 1.5 Overview

The remainder of this SRS contains the high-level and detailed fundamental requirements of the ASSIST/UNA software. Section 2 provides a general, high-level understanding of the ASSIST/UNA software’s purpose, function, and user characteristics. Section 3 details the fundamental ASSIST/UNA software requirements, including functional requirements, use cases, class/object relationships, and non-functional requirements.

# 2. General Description

The items contained within this section are high-level topics that influence the project and its requirements; these are not the specific requirements. The purpose of this section is to generally describe the requirements and to make the specific requirements easier to comprehend.

## 2.1 Product Perspective

The ASSIST/UNA software is a standalone product. Users of the software will be able to use it without requiring or relying on any outside software.

## 2.2 Product Functions

The main function of the ASSIST/UNA software is to emulate the ASSIST/I assembler with a subset of ASSIST’s instructions. The ASSIST/UNA software will enable users to write and assemble programs from an IDE. Users will also be able to assemble and debug their programs from the ASSIST/UNA environment. Users will also be able to view and print reports and source code directly from the IDE interface.

## 2.3 User Characteristics

The ASSIST/UNA software is to be used by the students and instructors of the CS 310 course and possibly the CS 311 course at UNA. Students should have a basic understanding of programming fundamentals and basic understanding of the ASSIST assembly language. Also, students should have knowledge of binary, hexadecimal, ASCII, and EBCDIC.[[9]](#footnote-9) Although not required, students with experience with IDEs will be benefited. Students lacking experience with IDEs will find the ASSIST/UNA IDE to be intuitive and user-friendly. Instructors should have a strong understanding of the ASSIST assembly language. The ASSIST/UNA IDE will enable instructors to easily open, examine, and test students’ programs. Instructors will also be able to write their own programs.

## 2.4 General Constraints

The ASSIST/UNA software will need to reliably emulate the ASSIST/I assembler, especially since ASSIST/UNA is a teaching tool. This means that processing by and output of ASSIST/UNA must always be accurate. ASSIST/UNA will also need to maintain a certain level of speed and storage efficiency. To facilitate this, constraints will be placed on certain memory-intensive components (see Sections 3.4.1 and 3.5).

The developers will implement the product using the Visual C# programming language and the .NET Framework. Development time constraints may prevent the developers from incorporating all of the client’s desired functionality for the ASSIST/UNA software. These features may be considered in future development and maintenance (see Appendix A.7).

## 2.5 Assumptions and Dependencies

The ASSIST/UNA software is to be produced for the Microsoft Windows operating system. Since the developers will be using Visual C# and the .NET Framework, only Windows Vista, Windows 7, and Windows 8/8.1 will be supported. The ASSIST/UNA software will be developed for 32-bit and 64-bit versions of the Windows operating system.

# 3. Specific Requirements

The purpose of this section is to give a detailed description of each requirement obtained from the client. Each of these requirements will dictate the development process and design as well as implementation and testing.

## 3.1 External Interface Requirements

This section details the user and software interfaces of the ASSIST/UNA software. See Appendix A.6 for screen captures of the prototype GUI.

### 3.1.1 User Interfaces

3.1.1.1 Source Code Editor

The Source Code Editor provides an interface in which users may write their ASSIST programs. Users may input data by typing into the Source Code Editor or by importing from another source file. The program source code in the Source Code Editor will serve as input to the ASSIST/UNA backend and may also be saved to a file or printed. The Source Code Editor will automatically highlight syntax within the program source code.[[10]](#footnote-10)

3.1.1.2 Registers Display

The Registers Display (see Figure A.6.1) provides a container in which the simulated machine’s sixteen general-purpose registers and Program Status Word register contents may be displayed to the user. The registers within the Registers Display are read-only, and the contents do not serve as input to any module. The simulator will automatically update the registers within the Register Display. The general-purpose registers will consist of eight hexadecimal characters, and the Program Status Word, a special type of register, will consist of sixteen hexadecimal characters.

3.1.1.3 Memory Display

The Memory Display (see Figure A.6.1) provides a container in which the simulated machine’s memory contents may be displayed to the user. The memory content within the Memory Display is read-only, and the memory content does not serve as input to any module. The simulator will automatically update the Memory Display. The Memory Display will have the following format: Address; Memory Contents; and, Character Representation.

3.1.1.4 Symbol Table Display

The Symbol Table Display (see Figure A.6.1) provides a container in which the assembler’s symbol table content may be displayed to the user. The symbol table content within the Symbol Table Display is read-only, and the symbol table content does not serve as input to any module. The simulator will automatically update the Symbol Table Display. The Symbol Table Display will have the following format: Symbol; and, Location.

3.1.1.5 Output Dialog Window

The Output Dialog Window (see Figure A.6.11) provides a container in which an executed program’s output and errors may be displayed to the user. The executed program’s output and errors within the Output Dialog Window are read-only, and the executed program’s output and errors are also used to generate the .PRT file. The simulator will automatically update the Output Dialog Window.

3.1.1.6 View .PRT Dialog Window

The View .PRT Dialog Window (see Figure A.6.10) provides a container in which an assembled and executed program’s .PRT file may be displayed to the user. The View .PRT Dialog Window contents are read-only and will be populated by the .PRT file generated partially by the assembler and partially by the simulator. The user will have the option to print the .PRT file from this dialog.

3.1.1.7 Options Dialog Window

The Options Dialog Window (see Figure A.6.9) enables users to modify the ASSIST/I options for the assembler. See Appendix A.5 for a list of modifiable assembler options.

### 3.1.2 Software Interfaces

3.1.2.1 Open File Dialog

The Open File Dialog enables the user to open source files for editing from within the ASSIST/UNA IDE. The user may open the Open File Dialog by selecting the corresponding menu or toolbar option. The contents of the file selected within the Open File Dialog will be imported to the Source Code Editor (see Section 3.1.1.1). The Open File Dialog functionality is enabled by the Windows operating system.

3.1.2.2 Save File Dialog

The Save File Dialog enables the user to save source files from within the ASSIST/UNA IDE. The user may open the Save File Dialog by selecting the corresponding menu or toolbar option. The contents of the Source Code Editor (see Section 3.1.1.1) will be saved to the user-specified file. The Save File Dialog functionality is enabled by the Windows operating system.

3.1.2.3 Print Dialog

The Print Dialog enables the user print program source code or .PRT files from within the ASSIST/UNA IDE. The user may open the Print Dialog by selecting the respective menu or toolbar option. The contents of the Source Code Editor (see Section 3.1.1.1) will be printed in portrait mode. The contents of the .PRT file will be printed in landscape mode. The Print Dialog functionality is enabled by the Windows operating system.

3.1.2.4 Import Dialog

The Import Dialog enables the user to import source files for editing from within the ASSIST/UNA IDE. The user may open the Import Dialog by selecting the corresponding menu or toolbar option. The contents of the file selected within the Import Dialog will be imported to the Source Code Editor (see Section 3.1.1.1). The Import Dialog functionality is enabled by the Windows operating system.

3.1.2.5 Export Dialog

The Export Dialog enables the user to export source files from within the ASSIST/UNA IDE. The user may open the Export Dialog by selecting the corresponding menu or toolbar option. The contents of the Source Code Editor (see Section 3.1.1.1) will be exported to the user-specified file. The Export Dialog functionality is enabled by the Windows operating system.

## 3.2 Functional Requirements

This section details the fundamental functional requirements of the ASSIST/UNA software.

### 3.2.1 Source Code Editing

3.2.1.1 Introduction

The ASSIST/UNA software shall enable users to write and edit their assembly program source code within the IDE. Users will also have the ability to save their work or open previous work. The users will be able to print their source code from the IDE. Users shall also have access to basic functionality, such as copy, cut, and paste. In addition, users shall be able to perform search and search and replace queries.

3.2.1.2 Inputs

Users will either type their program source code directly or open a file containing the program source code. Files will be automatically imported into the Source Code Editor (see Section 3.1.1.1).

3.2.1.3 Processing

All program source code entered will be forced to uppercase characters. The Source Code Editor (see Section 3.1.1.1) will apply syntax highlighting to the program source code. The exact syntax color scheme will be decided with the client at a later date. Users will be able to assemble their program source code. In addition, users will have the option to debug assembled programs.

3.2.1.4 Outputs

Program source code may be saved to a file or printed.

3.2.1.5 Error Handling

The user will be notified via message box if any Source Code Editor errors are encountered. These errors are not program runtime/execution errors; they are specifically related to text editing.

### 3.2.2 Simulated Source Code Assembly

3.2.2.1 Introduction

The ASSIST/UNA software shall enable users to assemble program source code from the IDE’s Source Code Editor (see Section 3.1.1.1).

3.2.2.2 Inputs

Input to the assembler shall be taken directly from the IDE’s Source Code Editor.

3.2.2.3 Processing

The assembler shall assemble the program source code by following a two-pass assembly process and shall build the object program. The assembler shall maintain a location counter, reference the machine operations table (see Appendix A.1), create a symbol table and literals table, and partially generate a .PRT file. See Appendix A.2 for a description of the two-pass assembly process. See Appendix A.3 for the .PRT file format. No simulated execution will occur. See Sections 3.2.3 and 3.2.4 for simulated execution.

3.2.2.4 Outputs

Upon successful assembly of the program source code, the assembler shall partially generate and save a .PRT file (see Appendix A.3). The .PRT filename will be the same as the source program filename.

3.2.2.5 Error Handling

Errors encountered during assembly shall be displayed to the user via the Output Dialog Window (see Section 3.1.1.5) and reported in the partial .PRT file. Errors will be reported during the assembler’s second pass. See Appendix A.4 for a list of errors that may be reported.

### 3.2.3 Source Code Assembly and Debugging

3.2.3.1 Introduction

The ASSIST/UNA software shall enable users to assemble program source code from the IDE’s Source Code Editor (see Section 3.1.1.1) and to debug the assembled program.

3.2.3.2 Inputs

Input to the assembler shall be taken directly from the IDE’s Source Code Editor.

3.2.3.3 Processing

The assembler shall assemble the program source code by following a two-pass assembly process and shall build the object program. The assembler shall maintain a location counter, reference the machine operations table (see Appendix A.1) and create a symbol table and literals table. See Appendix A.2 for a description of the two-pass assembly process. The simulator will simulate execution of the object program as it would run on an IBM/360 mainframe. After assembly, the symbol table contents will be displayed. The simulator will update registers and memory contents as program statements are executed. Users shall have the ability to execute one program statement at a time. A .PRT file will not be generated.

3.2.3.4 Outputs

The symbol table contents will be displayed to the user in the Symbol Table Display after successful assembly. The contents of registers and memory contents will be updated and displayed to the user (via the appropriate IDE components) after each program statement is executed. Any output from the program (e.g., XPRNT) shall be displayed in the Output Dialog Window (see Section 3.1.1.5). A .PRT file will not be saved.

3.2.3.5 Error Handling

Errors encountered during assembly shall be displayed to the user via the Output Dialog Window. Errors will be reported during the assembler’s second pass. See Appendix A.4 for a list of errors that may be reported.

### 3.2.4 Source Code Assembly and Final Run

3.2.4.1 Introduction

The ASSIST/UNA software shall enable users to assemble program source code from the IDE’s Source Code Editor (see Section 3.1.1.1) and to perform a final run.

3.2.4.2 Inputs

Input to the assembler shall be taken directly from the IDE’s Source Code Editor. The user shall specify an identifier for the .PRT file.

3.2.4.3 Processing

The assembler shall assemble the program source code by following a two-pass assembly process and shall build the object program. The assembler shall maintain a location counter, reference the machine operations table (see Appendix A.1), create a symbol table and literals table, and partially generate a .PRT file. See Appendix A.2 for a description of the two-pass assembly process. See Appendix A.3 for the .PRT file format. The simulator will simulate execution of the object program as it would run on an IBM/360 mainframe. . After assembly, the symbol table contents will be displayed, and the simulator will execute the program without pausing, unless user input is required. The simulator will update registers and memory contents as the program source code is executed. The simulator will complete the initial generation of the .PRT file.

3.2.4.4 Outputs

The contents of registers, memory, and the symbol table will be updated and displayed to the user (via the appropriate IDE components) during program execution. Upon successful program source code assembly and program termination, the assembler and simulator shall generate and save a .PRT file (see Appendix A.3). The .PRT filename will be the same as the source program filename. Any program output (e.g., XPRNT) shall be displayed to the user via the Output Dialog Window (see Section 3.1.1.5).

3.2.4.5 Error Handling

Errors encountered during assembly and runtime errors generated by the simulator shall be displayed to the user via the Output Dialog Window and reported in the .PRT file. Errors will be reported during the assembler’s second pass and at runtime. See Appendix A.4 for a list of errors that may be reported.

### 3.2.5 Report Viewing and Printing

3.2.5.1 Introduction

Users shall be able to view and print .PRT files (see Appendix A.3).

3.2.5.2 Inputs

Users shall select either the view or print option from the IDE interface.

3.2.5.3 Processing

The .PRT file contents, if the file exists and is not empty, shall be displayed to the user in portrait mode in a separate viewing window (see Section 3.1.1.6). The .PRT file shall be printed in landscape mode. The Windows operating system shall handle the printing process.

3.2.5.4 Outputs

The .PRT file shall be displayed in a window or printed.

3.2.5.5 Error Handling

Errors encountered during assembly shall be displayed to the user via the Output Dialog Window (see Section 3.1.1.5).

## 3.3 Use Cases

This section details the fundamental use cases of the ASSIST/UNA software.

### 3.3.1 Create a New Project

3.3.1.1 Description

The user creates a new working project to develop an assembly program.

3.3.1.2 Postconditions

The ASSIST/UNA IDE is reset to enable editing of a new, blank program.

3.3.1.3 Basic Flow

1. The user selects the option to create a new project.

2. The Source Code Editor (see Section 3.1.1.1) contents are cleared.

3. The registers (see Section 3.1.1.2) are reinitialized to the F4F4F4F4 default value.

4. The memory (see Section 3.1.1.3) content is reinitialized to the F5F5F5F5 default value.

5. The symbol table (see Section 3.1.1.4) contents are cleared.

6. The user writes and edits the new source program code.

### 3.3.2 Save a Project

3.3.2.1 Description

The user saves a working project to a source file within the Windows file system.

3.3.2.2 Postconditions

The source program code is saved to a user-specified file.

3.3.2.3 Basic Flow

1. The user selects the option to save a project.

2. The user is prompted to name the source file and the save location (see Section 3.1.2.2).

3. The contents of the Source Code Editor (see Section 3.1.1.1) are saved to the user-specified source file.

### 3.3.3 Open a Project

3.3.3.1 Description

The user opens a project to edit.

3.3.3.2 Postconditions

The ASSIST/UNA IDE is reset and the opened source file’s code is loaded into the Source Code Editor (see Section 3.1.1.1).

3.3.3.3 Basic Flow

1. The user selects the option to open a project.

2. The user specifies the source file to be opened (see Section 3.1.2.1).

3. The contents of the source file are imported to the Source Code Editor (see Section 3.1.1.1).

4. The registers (see Section 3.1.1.2) are reinitialized to the F4F4F4F4 default value.

5. The memory (see Section 3.1.1.3) content is reinitialized to the F5F5F5F5 default value.

6. The symbol table (see Section 3.1.1.4) contents are cleared.

7. The user writes and edits the program source code.

### 3.3.4 Assemble a Program

3.3.4.1 Description

The user assembles the program source code.

3.3.4.2 Postconditions

The assembled program’s object code is saved. A .PRT file (see Appendix A.3) is saved. Errors encountered (see Appendix A.4) are reported in the .PRT file and in the Output Dialog Window (see Section 3.1.1.5).

3.3.4.3 Basic Flow

1. The user selects the option to assemble the program source code.

2. The backend assembler assembles the program source code (see Section 3.2.2).

3. The assembled program’s object code is saved.

4. A .PRT file is saved.

5. Errors are reported in the .PRT file and in the Output Dialog Window.

### 3.3.5 Assemble a Program and Debug

3.3.5.1 Description

The user assembles and debugs the program source code.

3.3.5.2 Postconditions

The assembled program’s object code is saved. Errors encountered are reported in the Output Dialog Window (see Section 3.1.1.5). No .PRT file is saved.

3.3.5.3 Basic Flow

1. The user selects the option to assemble and debug the program source code.

2. The backend assembler assembles the program source code (see Section 3.2.3).

3. The assembled program’s object code is saved.

4. Errors are reported in the Output Dialog Window.

5. The user executes one program statement at a time until program termination.

6. Program output is displayed in the Output Dialog Window.

### 3.3.6 Assemble a Program and Perform a Final Run

3.3.6.1 Description

The user assembles the program source code and performs a final run (see Section 3.2.4).

3.3.6.2 Postconditions

The assembled program’s object code is saved. Errors encountered are reported in the .PRT file (see Appendix A.3) and the Output Dialog Window (see Section 3.1.1.1). A .PRT file is saved.

3.3.6.3 Basic Flow

1. The user selects the option to assemble the program source code and perform a final run.

2. The user specifies the final run identifier (usually the user’s name).

3. The backend assembler assembles the program source code.

4. The assembled program’s object code is saved.

5. Errors are reported in the Output Dialog Window.

6. The assembled program executes as it would on an IBM/360 mainframe.

7. A .PRT file, including errors, is generated and saved.

8. Program output is displayed in the Output Dialog Window.

### 3.3.7 View a .PRT File

3.3.7.1 Description

The user views a .PRT file (see Appendix A.3) for an assembled program.

3.3.7.2 Postconditions

The .PRT file is displayed in a viewing window (see Section 3.1.1.6)

3.3.7.3 Basic Flow

1. The user selects the option to view the .PRT file.

2. If the .PRT file exists, it is displayed in portrait mode in a viewing window.

3. If the .PRT file does not exist, the user is notified accordingly.

### 3.3.8 Print a .PRT File

3.3.8.1 Description

The user prints a .PRT file (see Appendix A.3) for an assembled program.

3.3.8.2 Postconditions

The printing of the .PRT file is handled by the Windows operating system.

3.3.8.3 Basic Flow

1. The user selects the option to print the .PRT file.

2. If the .PRT file exists, it is printed in landscape mode (via the Windows operating system).

3. If the .PRT file does not exist, the user is notified accordingly.

### 3.3.9 Print Program Source Code

3.3.9.1 Description

The user prints the program source code.

3.3.9.2 Postconditions

The printing of the program source code is handled by the Windows operating system.

3.3.9.3 Basic Flow

1. The user selects the option to print the program source code.

2. The program source code is printed in portrait mode (via the Windows operating system).

## 3.4 Non-Functional Requirements

This section contains the high-level descriptions of the non-functional requirements of the ASSIST/UNA software.

### 3.4.1 Performance

To facilitate run-time execution efficiency, certain memory size constraints may be instituted. For example, the maximum assembly program size is 9999 bytes (see Appendix A.5). These constraints will be documented in the user’s manual and the ASSIST/UNA source code.

### 3.4.2 Reliability

The ASSIST/UNA software has no tolerance for program crashes. However, in the event of extensive user errors, the ASSIST/UNA software will gracefully shut down. In this case, the user will be notified of the shutdown and the ASSIST/UNA software will attempt to gracefully recover from the error(s).

### 3.4.3 Licensing

The UNA Computer Science and Information Systems Department will retain full rights to the ASSIST/UNA software upon delivery. The ASSIST/UNA source code will not be freely open to all persons but will be made available to future classes as necessary.

### 3.4.4 Security

There are no security concerns for the ASSIST/UNA software at this time.

### 3.4.5 Maintainability

Upon delivery of the ASSIST/UNA software, the current development team shall be relinquished of all maintenance responsibilities. If the ASSIST/UNA software requires maintenance, the task shall be given to future classes. The client reserves the right to contact the current developers for courtesy consultations; however, the current developers are under no future obligations.

### 3.4.6 Portability and Delivery

The ASSIST/UNA software will be made as portable as possible for ease of delivery and use. The actual delivery method will be negotiated with the client after this document has been approved by both the developers and the client.

## 3.5 Design Constraints

Test cases involving source program code cannot be published publicly to GitHub. One of the developers is currently enrolled in CS 310 and cannot be allowed an unfair advantage. As such, he will be delegated to a primary role in frontend interface testing.

Many visual components, such as buttons, are available by default when using Visual Studio.

However, the team will inevitably have to create at least one custom component, namely a heavily modified text field. Also, the user shall have the ability to maximize the main dialog window, the View .PRT Dialog Window (see Section 3.1.1.6), and the Output Dialog Window (see Section 3.1.1.5).

The syntax highlighting color scheme will be negotiated with the client at a later date.

Since the chosen .NET Framework is supported only on Windows Vista, Windows 7, and Windows 8/8.1, the ASSIST/UNA software will not necessarily be portable to other operating systems.

In an effort to increase run-time execution efficiency, certain backend components will need to be limited. For example, to save on the amount of memory allocated, the symbol table may be limited to a certain number of labels. Any such limitations will be documented in the user’s manual and ASSIST/UNA source code.

## 3.6 Other Requirements

Users shall be able to modify the assembler options found in the original ASSIST/I via an options menu provided by ASSIST/UNA. These options include: Saving the output; changing the maximum number of output lines; changing the maximum number of program instructions; changing the maximum number of program pages; and, changing the maximum program size (in bytes). See Appendix A.5 for more information. Users also shall not be restricted to one working directory. That is, users shall be able to specify absolute and relative paths to data files for assembled programs.

# 4. Testing Plan

The testing plan for the ASSIST/UNA software includes test cases for complete assembly programs, assembly instructions (see Appendix A.1), errors (see Appendix A.4), GUI functionality, methods, classes, and modules.

All specifications in the SRS will be tested to ensure that each requirement is met. A spreadsheet will be used to list each requirement along with a set of test cases that check each specification. This spreadsheet will contain the following information for each requirement: the name and location of the requirement in the SRS and its associated specifications; the test cases that check the specifications; the date tested; and, a note of whether each test case passed or failed.

Test cases for the complete assembly programs will include programs that are correct and programs that produce errors. The .PRT file (see Appendix A.3) produced in ASSIST/I for the correct programs will be compared to the .PRT output from the ASSIST/UNA to check for output correctness. The memory dump from the incorrect programs in ASSIST/I will be compared to the memory and register contents of ASSIST/UNA software.

Test cases will also be created for each instruction that is to be implemented (see Appendix A.1). These test cases will check various inputs and uses of each instruction. A test case will also be created for each supported error that can occur on the ASSIST/I (see Appendix A.4) to ensure that the ASSIST/UNA is able to find each error and output the correct error message. The testing plan also includes test cases for the GUI, which include testing user interaction as well as test cases for the register, memory, and symbol table displays. Each method, class, and module used in ASSIST/UNA will also be individually tested before being integrated into the project.

The test cases and test materials will be uploaded to the team’s GitHub repository so each team member will have access to them. It is important to note, however, that correct, functioning assembly programs will not be posted to the GitHub repository (see Section 3.5). Correct, functioning assembly programs will be distributed by Software Quality Assurance to the appropriate team members. Testing materials will include a spreadsheet which will consist of the name of each test case file, the date it was tested, and whether the test was successful or not. The purpose of this spreadsheet is for team members to check where errors have been found by the testing process. The tests spreadsheet and records of each test performed will be stored on the GitHub repository.

# 5. Deliverables

The development team shall deliver project documents on April 29, 2014, in addition to providing a presentation for the client and other guests. The team shall deliver the following printed documents: The ASSIST/UNA source code; team emails; meeting minutes; project designs; the SRS; client questions and answers; the user’s manual; test files; and, test data. In addition, the team will deliver a digital copy of the user’s manual. The team will also present a method for the client to effectively and appropriately present the ASSIST/UNA software to future students.

# 6. Change Management Process

Before reporting a change or an error, developers must ensure they are consulting the most recent version of the software requirements specification document. If a change is required or an error is encountered, the SRS will be amended upon negotiation with and approval by the client. Developers shall notify the technical writer and the team leader via email. The email will detail the section(s) and line number(s) that need to be modified. In addition, the email will contain detailed descriptions of each change or error. The reporting developer shall also provide a detailed suggested correction. The technical writer will review the correction reports, approve the corrections, and update this document as necessary. The new version of this document shall be uploaded to the GitHub repository.

## 6.1 Email Report Guidelines

If a developer needs to report changes or errors in the SRS, he shall follow these guidelines:

* The email will be sent to the technical writer and the team leader.
* The email subject will read: “CS 455 – Spec Doc Changes.”
* The email will specify the version of the SRS being reviewed.
* The email will contain the following sections for each change or error:
  + Section Number
  + Line Number
  + Detailed description of the change or error
  + Detailed suggested correction
* Developers should send themselves a carbon copy for their own records.

Each section of the email should be appropriately labeled. For example, the “Section Number(s)” section should be labeled “**Section Number(s)**” (note the boldface). Also note that each reported change or error will need its own set of the aforementioned sections (with appropriate labels).

# 7. Client-Developer Contractual Agreement

The client (Dr. Patricia L. Roden) is satisfied that this SRS meets the project requirements as specified at this time. The developers (listed below) recognize the client’s right to request modifications of the project requirements. The client will ensure the modifications are necessary and will notify the developers at least one week in advance. The client hereby approves this SRS in its current form. The developers agree thusly to deliver the agreed upon deliverables on April 29, 2014.

The client agrees to fairly assess the work delivered by the developers, in accordance with the specified requirements within this document. Additionally, the client agrees to assign to each developer the appropriate grade mark for his individual and overall contributions to the project. While grading the overall work, the client agrees to consult each developer’s evaluations.

The signatures below hereby bind the client and the developers to this contractual agreement.

*Dr. Patricia L. Roden, Client* Date

*Travis Hunt, Team Leader* Date

*Michael Beaver, Technical Writer* Date

*Andrew Hamilton, Software Quality Assurance*  Date

*Drew Aaron* Date

*Clay Boren* Date

*Chad Farley* Date

# A. Appendices

Supplemental tables, figures, and miscellaneous information may be found within this appendix. The first section contains a list of ASSIST instruction types and the list of required instructions to be implemented. The second section contains initial high-level frontend and backend diagrams. The third section contains screen captures of the prototype. The final section contains features that may not be implemented due to time constraints but may be desired by the client at a later time.

## A.1 Assembly Instructions to Implement

This appendix contains the list of required ASSIST assembly instructions to be implemented in the ASSIST/UNA emulator. Table A.1.1 lists the types of instructions supported by ASSIST/UNA. Table A.1.2 lists each instruction’s mnemonic, description, instruction type, and basic form.

Table A.1.1: ASSIST instruction types supported by ASSIST/UNA.

|  |  |
| --- | --- |
| **Type** | **Type Description** |
| RR | Register-Register. Operands are taken from registers, manipulated, and the result is stored into the first operand’s register (e.g., AR 3,4). |
| RS | Register-Storage. The first operand is in a register, and the second operand is in main storage. Memory contents are accessed by D(B) addressing (e.g., LM 4,6,12(13)). |
| RX | Register-Storage. The first operand is in a register, and the second operand is in main storage. Memory contents are accessed by D(X,B) addressing (e.g., LA 11,10(3,9)). |
| SI | Storage-Immediate. The first operand is in main storage, and the second operand is an eight-bit (one byte) *immediate* operand. Memory contents are accessed by D(B) addressing (e.g., CLI 0(20),C’a’). |
| SS | Storage-Storage. Both operands are in main storage. Memory contents are accessed using D(B) addressing. Often a length L is specified (e.g., MVC 0(5,3),0(2)). |
| X\* | An instruction whose mnemonic is preceded by an “X” is a special macro instruction that actually comprises several instructions. Memory contents are accessed using D(X,B) addressing (e.g., XDECI 3,0(1)). |

Table A.1.2: ASSIST/I Instructions to be implemented.

|  |  |  |  |
| --- | --- | --- | --- |
| **Mnemonic** | **Description** | **Type** | **Basic Form** |
| A | Add | RX | A R1,D2(X2,B2) |
| AP | Add Packed | SS | AP D1(L1,B1),D2(L2,B2) |
| AR | Add Register | RR | AR R1,R2 |
| BAL | Branch and Link | RX | BAL R1,D2(X2,B2) |
| BALR | Branch and Link Register | RR | BALR R1,R2 |
| BC | Branch on Condition | RX | BC B’mask’,D2(X2,B2) |
| BCR | Branch on Condition Register | RR | BCR B’mask’,R2 |
| BCT | Branch on Count | RX | BCT R1,D2(X2,B2) |
| BCTR | Branch on Count Register | RR | BCTR R1,R2 |
| BXH | Branch on Index High | RS | BXH R1,R2,D3(B3) |
| BXLE | Branch on Index Low or Equal | RS | BXLE R1,R2,D3(B3) |
| C | Compare | RX | C R1,D2(X2,B2) |
| CLC | Compare Logical Characters | SS | CLC D1(L1,B1),D2(B2) |
| CLI | Compare Logical Immediate | SI | CLI D1(B1),I2 |
| CP | Compare Packed | SS | CP D1(L1,B1),D2(L2,B2) |
| CR | Compare Register | RR | CR R1,R2 |
| D | Divide | RX | D R1,D2(X2,B2) |
| DP | Divide Packed | SS | DP D1(L1,B1),D2(L2,B2) |
| DR | Divide Register | RR | DR R1,R2 |
| ED | Edit | SS | ED D1(L1,B1),D2(B2) |
| EDMK | Edit and Mark | SS | EDMK D1(L1,B1),D2(B2) |
| L | Load | RX | L R1,D2(X2,B2) |
| LA | Load Address | RX | LA R1,D2(X2,B2) |
| LM | Load Multiple | RS | LM R1,R2,D3(B3) |
| LR | Load Register | RR | LR R1,R2 |
| M | Multiply | RX | M R1,D2(X2,B2) |
| MP | Multiply Packed | SS | MP D1(L1,B1),D2(L2,B2) |
| MR | Multiply Register | RR | MR R1,R2 |
| MVC | Move Characters | SS | MVC D1(L1,B1),D2(B2) |
| MVI | Move Immediate | SI | MVI D1(B1),I2 |
| N | Bitwise AND | RX | N R1,D2(X2,B2) |
| NR | Bitwise AND Register | RR | NR R1,R2 |
| O | Bitwise OR | RX | O R1,D2(X2,B2) |
| OR | Bitwise OR Register | RR | OR R1,R2 |
| PACK | Pack | SS | PACK D1(L1,B1),D2(L2,B2) |
| S | Subtract | RX | S R1,D2(X2,B2) |
| SP | Subtract Packed | SS | SP D1(L1,B1),D2(L2,B2) |
| SR | Subtract Register | RR | SR R1,R2 |
| ST | Store | RX | ST R1,D2(X2,B2) |
| STM | Store Multiple | RS | STM R1,R2,D3(B3) |
| UNPK | Unpack | SS | UNPK D1(L1,B1),D2(L2,B2) |
| XDECI | Convert Input to Decimal | X\* | XDECI R1,D2(X2,B2) |
| XDECO | Convert Output to Decimal | X\* | XDECO R1,D2(X2,B2) |
| XPRNT | Print Output | X\* | XPRNT D1(X1,B1),L2 |
| XREAD | Read Input | X\* | XREAD D1(X1,B1),L2 |
| ZAP | Zero, Add Packed | SS | ZAP D1(L1,B1),D2(L2,B2) |

## A.2 The Two-Pass Assembly Process

This appendix contains a high-level description of the two-pass assembly process. The ASSIST assembler is a two-pass assembler, so the ASSIST/UNA emulator will also be two-pass.

**A.2.1 Pass 1**

1. Maintain the location counter
   1. Make use of the machine operations table (see Appendix A.1)
   2. Process DS and DC statements
      1. Know sizes in bytes
         1. Character = 1 byte
         2. Halfword = 2 bytes
         3. Fullword = 4 bytes
         4. Doubleword = 8 bytes
      2. Enforce boundary alignments
   3. Process literals and create the literal table
2. Construct the symbol table
   1. Anything in columns 1-8 (i.e., labels) and the current location counter are entered into the symbol table
3. *(Design Option)* Create an intermediate file for Pass 2
4. Process assembler directives
   1. START
   2. END
   3. USING
   4. DS and DC
   5. SPACE
   6. TITLE
   7. EJECT

**A.2.2 Pass 2**

In Pass 2, use the intermediate file from Pass 1 or the original source file.

1. Create object code for each line
   1. Use the machine operations table
   2. Use the symbol table
   3. Use the literal table
2. Create a listing line in the .PRT file (see Appendix A.3)
3. Print any errors encountered in Pass 1 or Pass 2
4. Create an object program, which the emulator will execute

## A.3 The .PRT File Format

This appendix contains the basic format of the generated .PRT file. The .PRT file is constituted by source listings (rows) and columns. The columns are the location counter, the object code, the line number, and the source statement. The ASSIST/I .PRT file format has a column for an address; however, this will be omitted for simplicity. The .PRT filename will be the same as the source program filename with the .PRT file extension. Table A.3.1 provides an excerpt of a generated .PRT file.

Table A.3.1: An excerpt from a sample .PRT file.

|  |  |  |  |
| --- | --- | --- | --- |
| LOC | OBJECT CODE | STMT | SOURCE STATEMENT |
| 000000 |  | 1 | MYPROG START |
| 000000 | 90EC D00C | 2 | STM 14,12,12(13) |
| 000004 | 05C0 | 3 | BALR 12,0 |
| 000006 |  | 4 | USING SAVING,12 |
| 000006 | 50D0 C07E | 5 | SAVING ST 13,SAVEAREA+4 |
| 00000A | 41D0 C07A | 6 | LA 13,SAVEAREA |
| ... | ... | ... | ... |
| 000080 |  | 50 | SAVEAREA DS 18F |
|  |  | 51 | END MYPROG |

## A.4 Supported Assembler Errors

This appendix contains basic descriptions of the types of errors that may be encountered during assembly. The following errors are supported: Addressing exception; data exception; decimal-overflow exception; decimal-divide exception; fixed-point overflow exception; fixed-point divide exception; operation exception; protection exception; and, specification exception. Table A.4.1 summarizes which types of errors are possible for each mnemonic (see Appendix A.1).

**A.4.1 Addressing Exception (A)**

An addressing exception occurs when attempting to reference an address outside of actual memory.

**A.4.2 Data Exception (D)**

A data exception is the result of an attempt to manipulate a field that should contain a valid packed-decimal number but does not.

**A.4.3 Decimal-overflow Exception (DO)**

A decimal-overflow exception occurs when the result of a decimal arithmetic instruction cannot be represented in the receiving field.

**A.4.4 Decimal-divide Exception (DD)**

A decimal-divide exception occurs when the quotient from the execution of a DP statement (see Appendix A.1) is too large to be represented in the allotted space or when an attempt is made to divide by zero.

**A.4.5 Fixed-point Overflow Exception (FO)**

A fixed-point overflow exception occurs when the result of an arithmetic operation cannot be represented in a 32-bit word.

**A.4.6 Fixed-point Divide Exception (FD)**

A fixed-point divide exception occurs when the quotient of D or DR (see Appendix A.1) cannot be represented in a 32-bit word. This error also occurs in the event of division by zero.

**A.4.7 Operation Exception**

An operation exception occurs when an attempt is made to execute a memory value that is not a valid instruction. An operation exception can occur if a constant is included in the middle of a routine, if a section of a program is altered during execution, or if a branch address is incorrect.

**A.4.8 Protection Exception (P)**

A protection exception occurs when an executed instruction attempts to access or modify memory outside the limits of the program.

**A.4.9 Specification Exception (S)**

A specification exception occurs when boundary alignment is unobserved or when an unexpected value is encountered in an index register.

Table A.4.1: The assembler errors supported by each mnemonic (see Appendix A.1).

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Mnemonic** | **A** | **D** | **DO** | **DD** | **FD** | **FO** | **P** | **S** |
| A | X |  |  |  | X |  | X | X |
| AP | X | X | X |  |  |  | X |  |
| AR |  |  |  |  | X |  |  |  |
| BAL |  |  |  |  |  |  |  |  |
| BALR |  |  |  |  |  |  |  |  |
| BC |  |  |  |  |  |  |  |  |
| BCR |  |  |  |  |  |  |  |  |
| BCT |  |  |  |  |  |  |  |  |
| BCTR |  |  |  |  |  |  |  |  |
| BXH |  |  |  |  |  |  |  |  |
| BXLE |  |  |  |  |  |  |  |  |
| C | X |  |  |  |  |  | X | X |
| CLC | X |  |  |  |  |  | X |  |
| CLI | X |  |  |  |  |  | X |  |
| CP | X | X |  |  |  |  | X |  |
| CR |  |  |  |  |  |  |  |  |
| D | X |  |  |  |  | X | X | X |
| DP | X | X |  | X |  |  | X | X |
| DR |  |  |  |  |  | X |  | X |
| ED | X | X |  |  |  |  | X |  |
| EDMK | X | X |  |  |  |  | X |  |
| L | X |  |  |  |  |  | X | X |
| LA |  |  |  |  |  |  |  |  |
| LM | X |  |  |  |  |  | X | X |
| LR |  |  |  |  |  |  |  |  |
| M | X |  |  |  |  |  | X | X |
| MP | X | X |  |  |  |  | X | X |
| MR |  |  |  |  |  |  |  | X |
| MVC | X |  |  |  |  |  | X |  |
| MVI | X |  |  |  |  |  | X |  |
| N | X |  |  |  |  |  | X | X |
| NR |  |  |  |  |  |  |  |  |
| O | X |  |  |  |  |  | X | X |
| OR |  |  |  |  |  |  |  |  |
| PACK | X |  |  |  |  |  | X |  |
| S | X |  |  |  | X |  | X | X |
| SP | X | X | X |  |  |  | X |  |
| SR |  |  |  |  | X |  |  |  |
| ST | X |  |  |  |  |  | X | X |
| STM | X |  |  |  |  |  | X | X |
| UNPK | X |  |  |  |  |  | X |  |
| ZAP | X | X | X |  |  |  | X |  |

## A.5 ASSIST/I Assembler Options

This appendix contains a list of the ASSIST/I assembler options that a user may modify for the ASSIST/UNA assembler. Table A.5.1 provides a summary of the options’ names, descriptions, and lower and upper bounds. Each option’s upper bound is meant to support the memory efficiency constraint. The upper bounds are the original bounds enforced by the ASSIST/I assembler.

Table A.5.1: ASSIST/I assembler options that may be modified.

|  |  |  |  |
| --- | --- | --- | --- |
| **Option Name** | **Option Description** | **Lower Bound** | **Upper Bound** |
| Maximum Number of Source Lines | This option limits the length of the program source code. | 1 | 9999 |
| Maximum Number of Source Instructions | This option limits the number of source instructions within the program source code. | 1 | 9999 |
| Maximum Number of Printout Lines | This option limits the number of lines in the .PRT file. | 1 | 9999 |
| Maximum Size in Bytes | This option limits the program size in bytes. | 1 | 9999 |

## A.6 Prototype Graphical User Interface Screen Captures

This appendix contains screen captures of the initial ASSIST/UNA prototype. Note: This prototype is *not* final and is subject to change. This prototype is meant to serve as inspiration and as a point of discussion. Note that the file input/output dialogs are the standard Windows-provided dialogs; hence, screen captures of these dialogs are omitted. Table A.5.1 lists the appropriate custom RGB color values for the user interface.

Table A.6.1: The custom RGB color values for the user interface.

|  |  |  |
| --- | --- | --- |
| **RGB Value** | **Color Description** | **Locations Used** |
| 180, 100, 255 | Light purple | The Registers Display background panel color and button background colors |
| 152, 0, 230 | Medium purple | All dialog background colors |
| 120, 0, 120 | Dark purple | The Registers Display registers content text and the status bar on the status strip (main dialog) |
| 255, 255, 162 | Light gold | The Registers Display registers content background and the status label background on the status strip (main dialog) |

Figure A.6.1: The default IDE state with a program loaded.

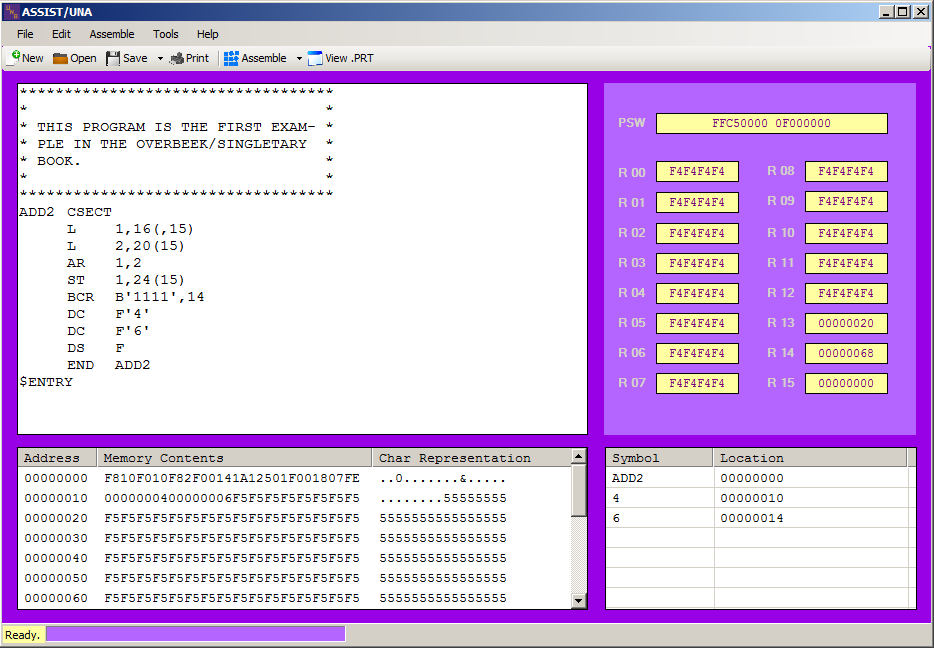


Figure A.6.2: The “File” menu options.

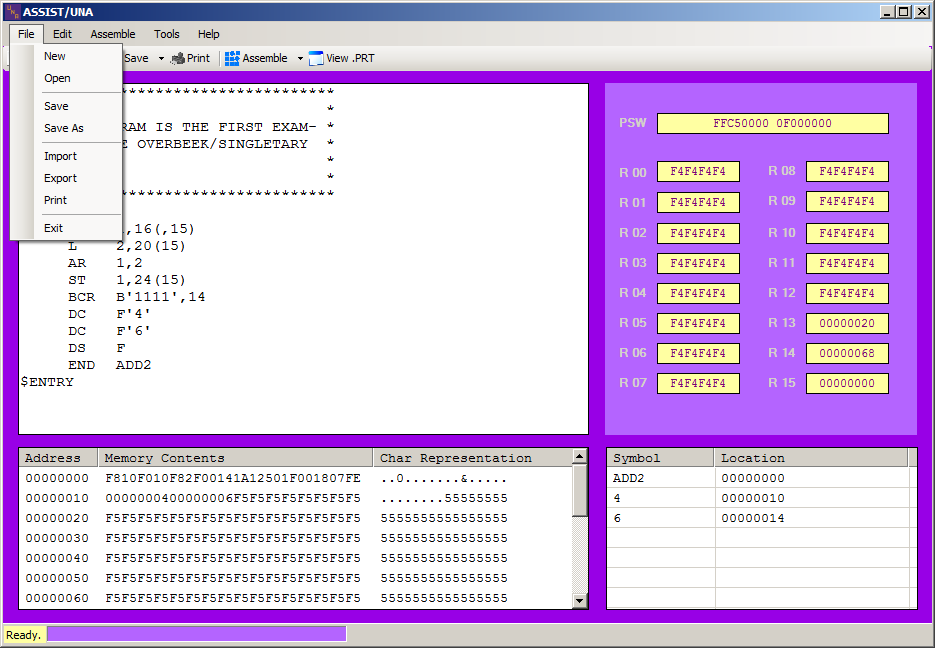


Figure A.6.3: The “Edit” menu options.

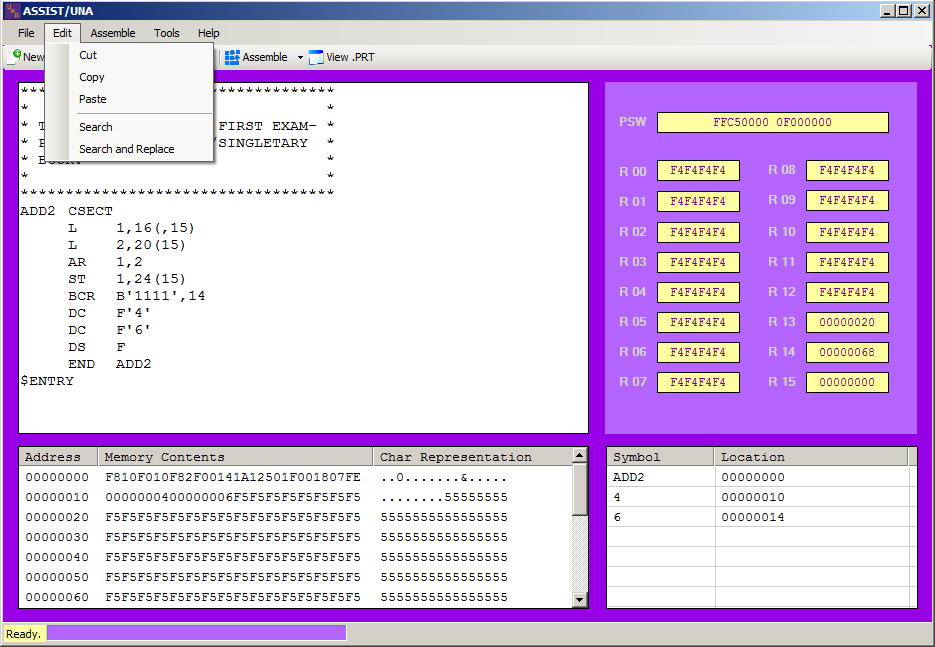


Figure A.6.4: The “Assemble” menu options.

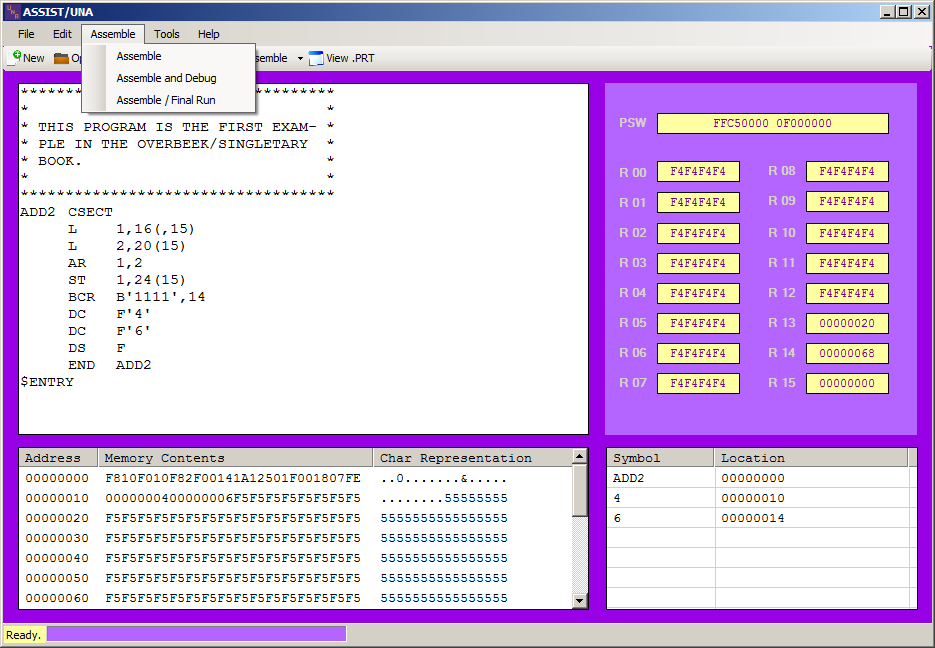


Figure A.6.5: The “Tools” menu options.

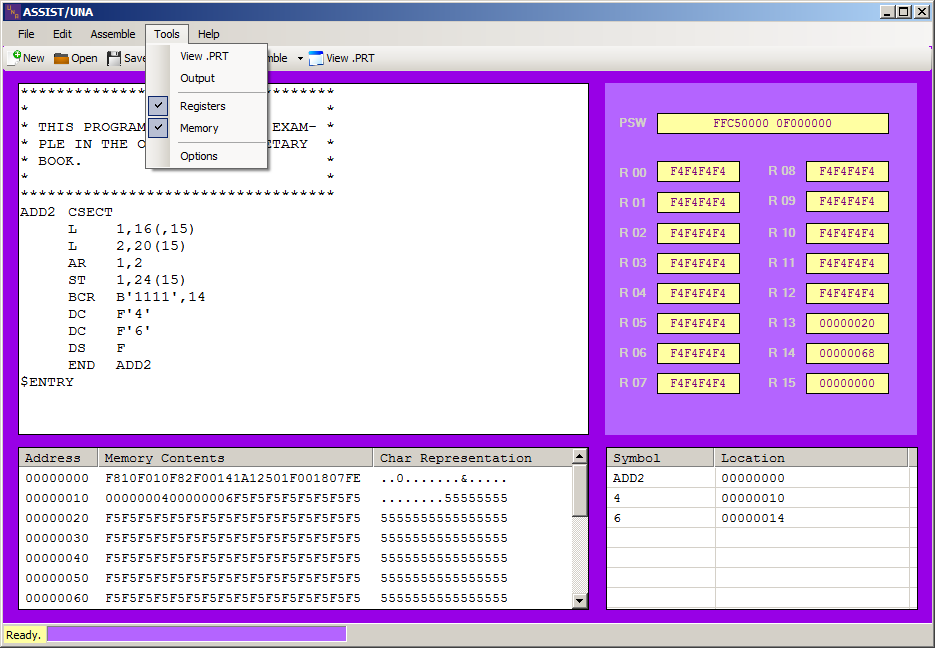


Figure A.6.6: The “Help” menu options.

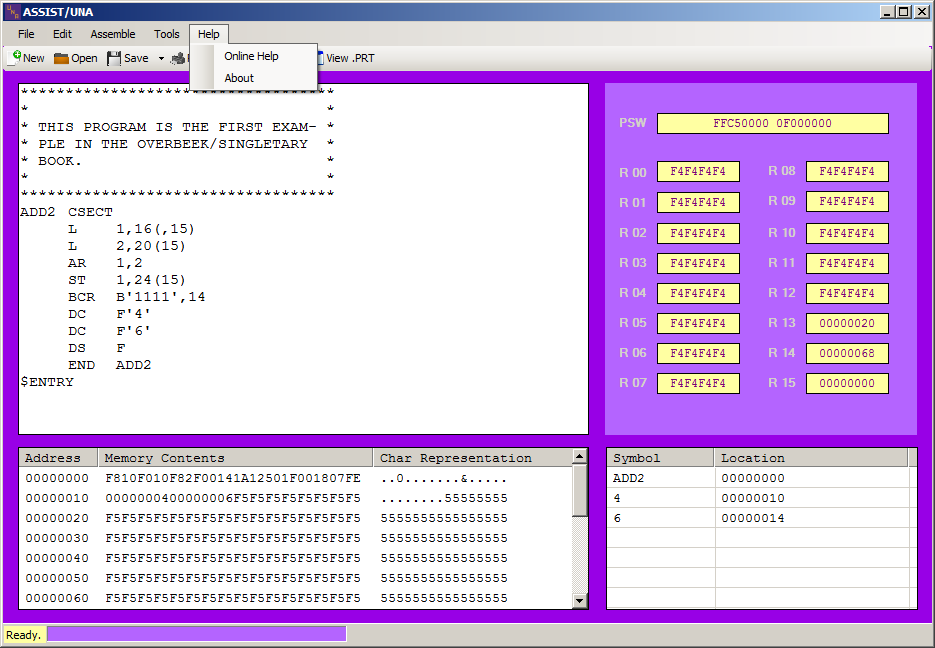


Figure A.6.7: The “Save” toolbar options.

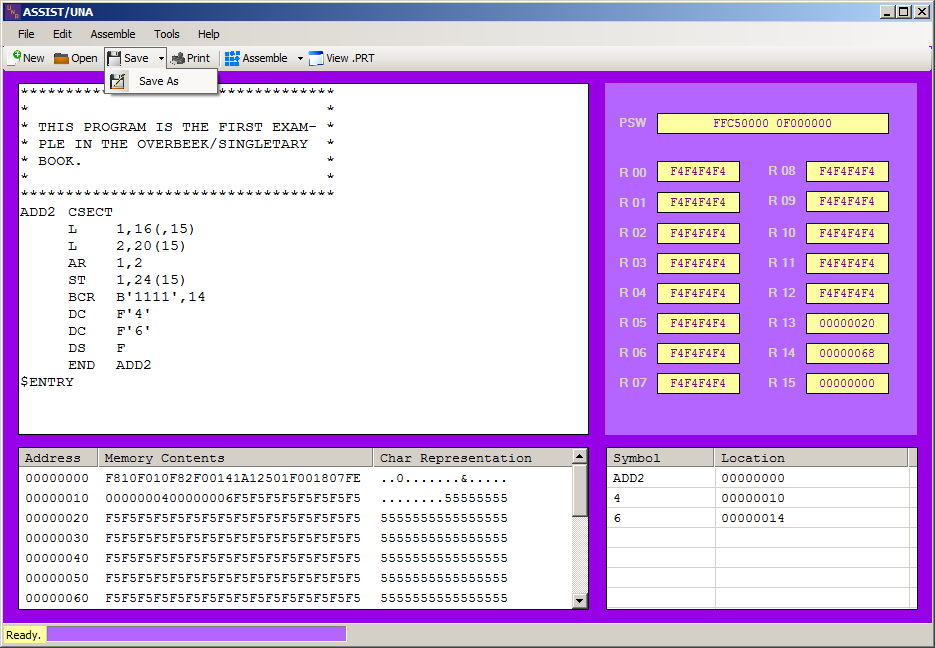


Figure A.6.8: The “Assemble” toolbar options.

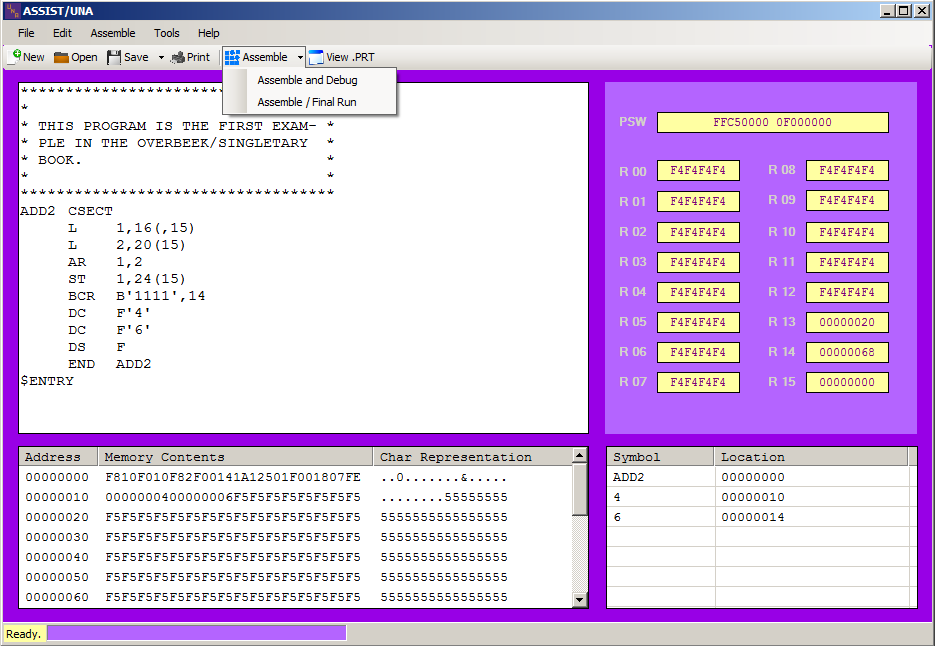


Figure A.6.9: The Options menu.

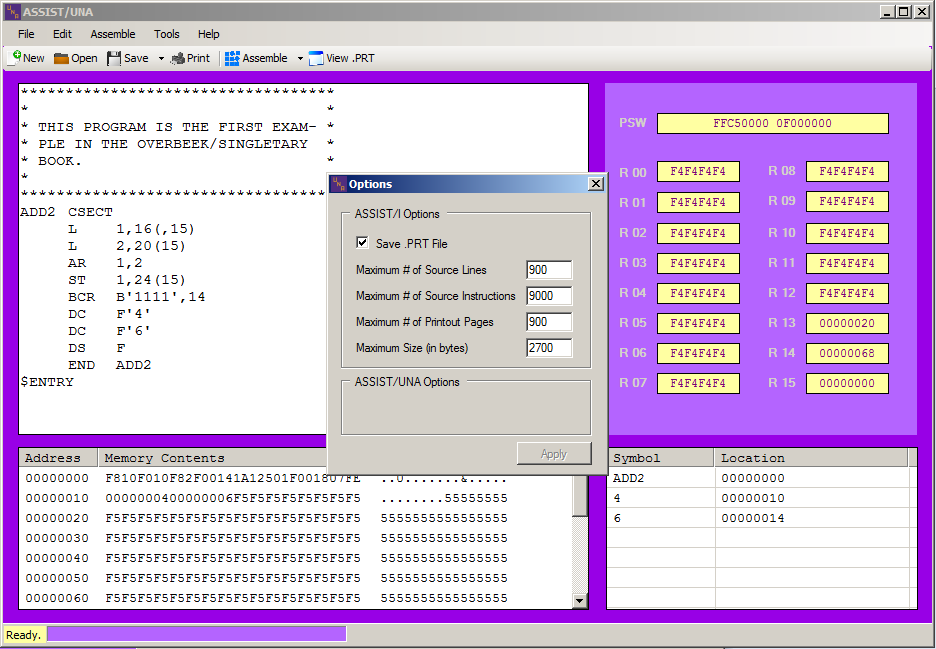


Figure A.6.10: The View .PRT window.

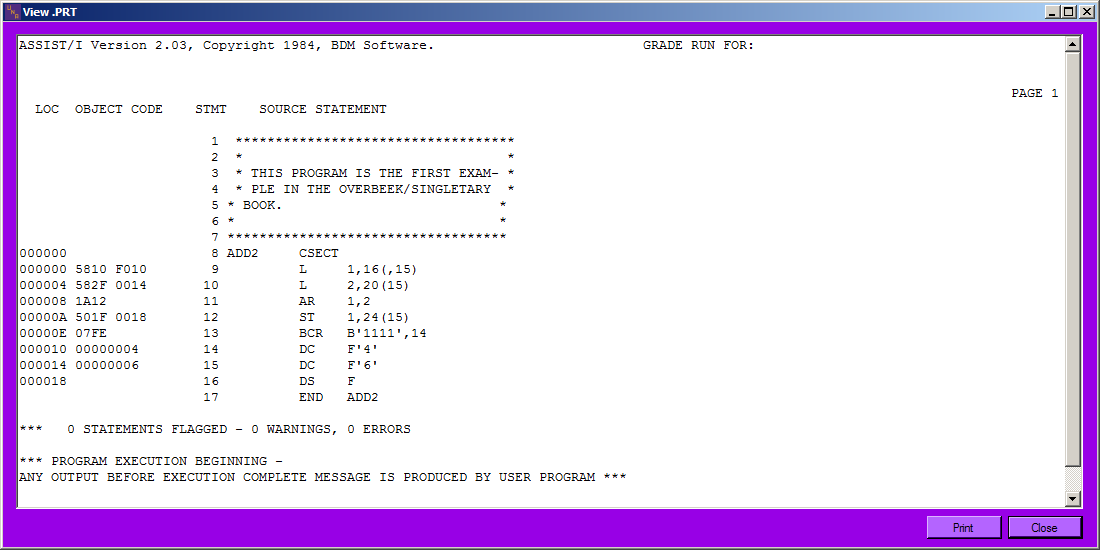
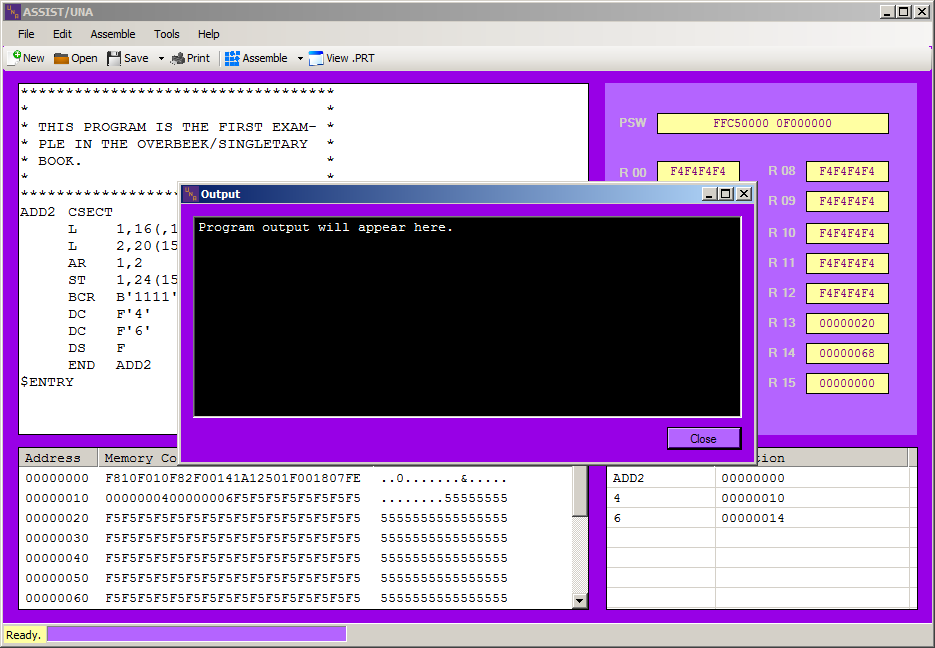


Figure A.6.11: The program Output Dialog Window.



## A.7 Features for Future Consideration

This appendix lists ASSIST/UNA software features that are not required but may be considered for implementation at a later date.

**A.7.1 Formatted Tabs**

When the user presses the tab key in the Source Code Editor (see Section 3.1.1.1), the cursor within the editor moves the appropriate amount of spaces.

**A.7.2 Opcode Completion or Hinting**

When the user begins entering an instruction’s mnemonic, the Source Code Editor (see Section 3.1.1.1) automatically completes the instruction. Alternatively, the user could be presented with a list of potential instruction mnemonics.

**A.7.3 Embedded Calculator**

The user presses a certain shortcut or hotkey to bring up a calculator within the IDE. The calculator should have the ability to calculate with and convert between hexadecimal, decimal, octal, and binary values.

**A.7.4 Disassembly**

The user imports an assembled program that may be disassembled back into its original program source code.

**A.7.5 Reference Asterisk**

The user may use an asterisk within the program source code to refer to the location counter.

**A.7.6 Floating-point Registers**

The ASSIST/UNA assembler should be able to emulate the IBM/360’s four floating-point registers.

**A.7.7 Breakpoints**

The user will be able to set and run to breakpoints in the debugging mode.

1. See http://www.github.com. [↑](#footnote-ref-1)
2. See http://www.visualstudio.com. [↑](#footnote-ref-2)
3. Available on ANGEL. See Appendix A.1. [↑](#footnote-ref-3)
4. Given as a handout. [↑](#footnote-ref-4)
5. See footnote 1. See Sections 2 and 3. [↑](#footnote-ref-5)
6. See footnote 1. See Sections 2 and 3. [↑](#footnote-ref-6)
7. Available at http://standards.ieee.org/findstds/standard/830-1998.html. This IEEE guide is used throughout this software requirements specification document. [↑](#footnote-ref-7)
8. Available at http://msdn.microsoft.com/en-us/library/8z6watww%28v=vs.110%29.aspx. See Section 2.5. [↑](#footnote-ref-8)
9. Students should have completed CS 155 and CS 245. [↑](#footnote-ref-9)
10. The exact syntax highlighting color scheme will be negotiated with the client at a later date. [↑](#footnote-ref-10)