Section 05.2 - Language Translators

Layer 4: Operating System

Syllabus Content Section 05: System Software

S05.2.1 Show understanding of the need for:

- assembler software for the translation of an assembly language program
- a compiler for the translation of a high-level language program
- an interpreter for translation and execution of a high-level language program

---

Assembly language code made from two parts: Opcode & Operand.
This assembly language code then has to be translated into machine code (binary), done by the Assembler.
There are two types of assembler:

> Load-and-go Assembler

1. You write your assembly code
2. The assembler looks at each line
3. Right away it executes each line

- Fast
- Don't chech errors
- no forward referencing

> One Pass Assembler

1. Reads all your code one time
2. As it reads your code it builds two tables. An Opcode Table and a Symbol table
3. When it finishes building the tables, then your code is run

- Fast, only one pass
- If there’s any errors it can see
- Cannot handle forward referencing

> Two Pass Assembler

1. removal of comments

2. replacement of a macro name used in an instruction by the list of instructions that constitute the macro definition
3. removal and storage of directives to be acted upon later

S05.2.2 Explain the benefits and drawbacks of using either a compiler or interpreter and justify the use of each

---

Compiler = All at once
Interpreter = Line by line

> Interpreter:

- Translates line by line
- Quick to start
- If it finds an error it stops right away
- Not portable. Your code and the interpreter must be together

> Compiler

- Reads all your code first
- Translates everything in one time
- Slow to start
- If there's an error, too bad.
- Portable. When the translation is done, you can take your program to another computer and use it.
- .exe files have been compiled.

S05.2.3 Show awareness that high-level language programs may be partially compiled and partially interpreted, such as Java

---

> for some High-level language:

SOURCE CODE -> COMPILER -> MACHINE CODE

> for JAVA:

SOURCE -> COMPILER -> JAVA BYTECODE -> JAVA VIRTUAL MACHINE -> MACHINE CODE

S05.2.4 Describe features found in a typical Integrated Development Environment (IDE)

Including:

- for coding, including context-sensitive prompts
- for initial error detection, including dynamic syntax checks
- for presentation, including prettyprint, expand and collapse code blocks
- for debugging, including single stepping, breakpoints, i.e. variables, expressions, report window

---

> IDE:

1. Software you can type your code into (text editor)
2. Compiler or interpreter
3. Automation tools. To help you code quicker
4. Debugger. To check for errors

> Some features:

- Context – Sensitive Prompt
  When something comes up on screen based on what you are doing.
  Example, that menu came up when I was typing in “print”

- Initial Error Detection
  Your IDE will check for errors when you ask for it

- Dynamic Syntax Checks
  Your IDE checks for syntax (grammar) errors as you type. So, if you wanted to PRINT but you wrote PNT instead, it will show this as an error as you are typing.

- PrettyPrint
  When you use indents (spaces) to make your code look more beautiful

- Expand and Collapse Blocks
  When you can show or hide pieces of code

> IDE Debugging

A debugger looks at your code and checks it for mistakes.
Do not confuse with compiler or interpreter which translates your code
Its just that a debugger will usually run first then your compiler or interpreter will translate if your code is perfect.

- Single Stepping:
  Each block of your code is checked one by one.

- Breakpoint
  You put special code into your code that tells the debugger “when you get to this point, stop and open the debugging / report window”

- Debugging window / report window
  Its where the debugger shows you where the mistake is.