Component 1 | 1.2.2| Application generation

Specification & learning objectives

A Level	Specification point description
1.2.2a	The nature of applications, justifying suitable applications for a specific purpose
1.2.2b	Utilities
1.2.2c	Open source vs Closed source
1.2.2d	Translators: interpreters, compilers and assemblers
1.2.2e	Stages of compilation (Lexical analysis, Syntax analysis, Code generation and Optimisation)
1.2.2f	Linkers and loaders and use of libraries

Resources

PG Online textbook page ref: 39-50

Hodder textbook page ref: 97-99, 107-115

CraignDave videos for SLR 5

Component 1 | 1.2.2| Application generation

Application- Any program which is made for the user.
Utilities- Looks after the PC's with a firewall, disk defragmentation and zips(file compressions).
Open source- A piece of software that comes with the original source code, normally free of charge and used for creativity.
Closed open- A piece of software that doesn't come with the source code and is copyrighted
Source code- The original code for the software.

Open source		✓	Closed source
✓	Free license	✓	Free support
✓	Can modify as the user	✓	Free updates
✓	Source code is available	✓	Software is covered by trade description act (must work)
✓	Can share with others	o	Software can't be altered
✓	New versions must be shared	o	The source code isn't easy to get
o	Pay for support	o	Software can't be shared with other users
o	Finished product may not look professional		

Types of applications
(Don't name brands like Microsoft)

- Word processor
- Desktop publisher
- Spreadsheets
- Data base management
- Social networking
- Email clients
- Web browsing
- Gaming
- Slideshow and presenting
- Multimedia and video editing
- Photo and graphic manipulation
- Communication, chat and IM

Utility	Description
Disk Defragmenters	Over time files on hard drives can become split up and spread apart making retrieval of files slower. This software helps to consolidate the parts of the files back together.
Anti-virus Programs	Helps to detect and remove malicious programs which have often been designed to harm a computer in some way.
Compression Utilities	Reduces the amount of space information takes up on a storage device.
Backup Utilities	Provides a way to recover data in case the original copy gets lost, deleted or corrupted.
File Managers	Allows directories, folders and files and to created, moved, copied, deleted and renamed.

Key question: In what ways do typical businesses use applications software?

Although open source software is very popular with private citizens, open source applications are also widely used in commercial organisations. Businesses do this when it makes commercial sense to do so. This usually means the open source applications they choose to use are mature, stable and well supported.

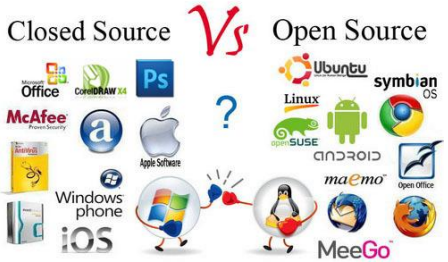
For example web sites are hosted on fully commercial servers that are maintained under contract - they are not free! But at the same time, the software applications running on the server may be open source.

A typical web server could be running:-

- Apache (open source web server application - approx 50% of all web sites use this)
- MySQL (database application)
- PHP (web programming language)
- Linux (open source operating system)

Non-technical businesses in general do not want the burden of maintaining and updating open source software and so they will pay for technical support from a company that will maintain it for them.

Key question: What are the considerations for a school between choosing an open or closed learning platform?



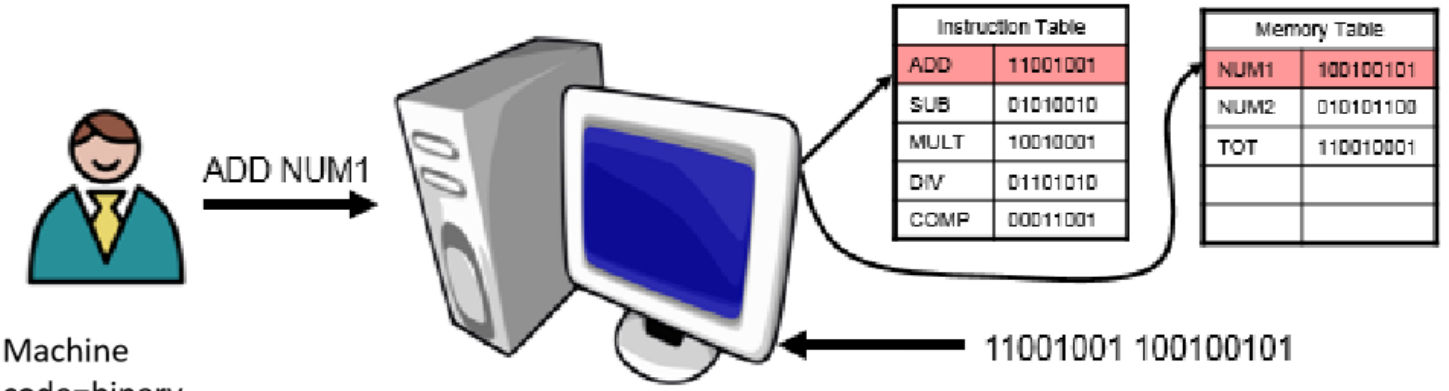
Open	Closed
The source code is publicly available	Source code is a trade secret
Application is free	Application is not free
Licence allows code to be copied and modified	Licence restricts copying and modifying
Less polished interface	Quite polished to attract customers
Written by expert volunteers	Written by paid software programmers
Documentation and technical support limited	Documentation usually good and support is formally available
Arguably less secure as the source code can be deliberately examined for weaknesses and taken advantage of.	Arguably more secure as the source code is not available so weaknesses are not so apparent
Counter argument - popular open source applications have hundreds of volunteers looking for security flaws in the source code and are quickly patched, but many people will still be using unpatched executables from their original binaries and never install the latest version.	Many applications automatically look for software updates back to the vendor's servers. So fixes are propagated rapidly to customers without any technical knowledge required.
No enforced deadlines as the code is written by volunteers	The company will have strict deadlines for programmers to make new releases
Lower costs of development as the programmers work for free	Higher development costs as programmers are paid

Key question: How does a VB program become the binary code that a computer can execute?

Translator- **These consist of assemblers, interpreters and compilers to convert source code into machine code.**
Interpreter- **This converts high level source code into machine code.**
Compiler- **Converts high level source code into object code then into machine code.**
Assembler- **Converts low level code directly into machine code.**

Assemblers

- Convert low level languages directly into machine code
- There is a 1 to 1 relationship
- Assembly code isn't very portable over platforms, as assembly code for one sever may not run on another CPU
- Translates a program written in assembly language into machine code.



Machine code=binary

Interpreters and compilers

- Convert high level code such as VB, python and C#
- Have a one to many relationship(1 line of source code = many lines of machine code)

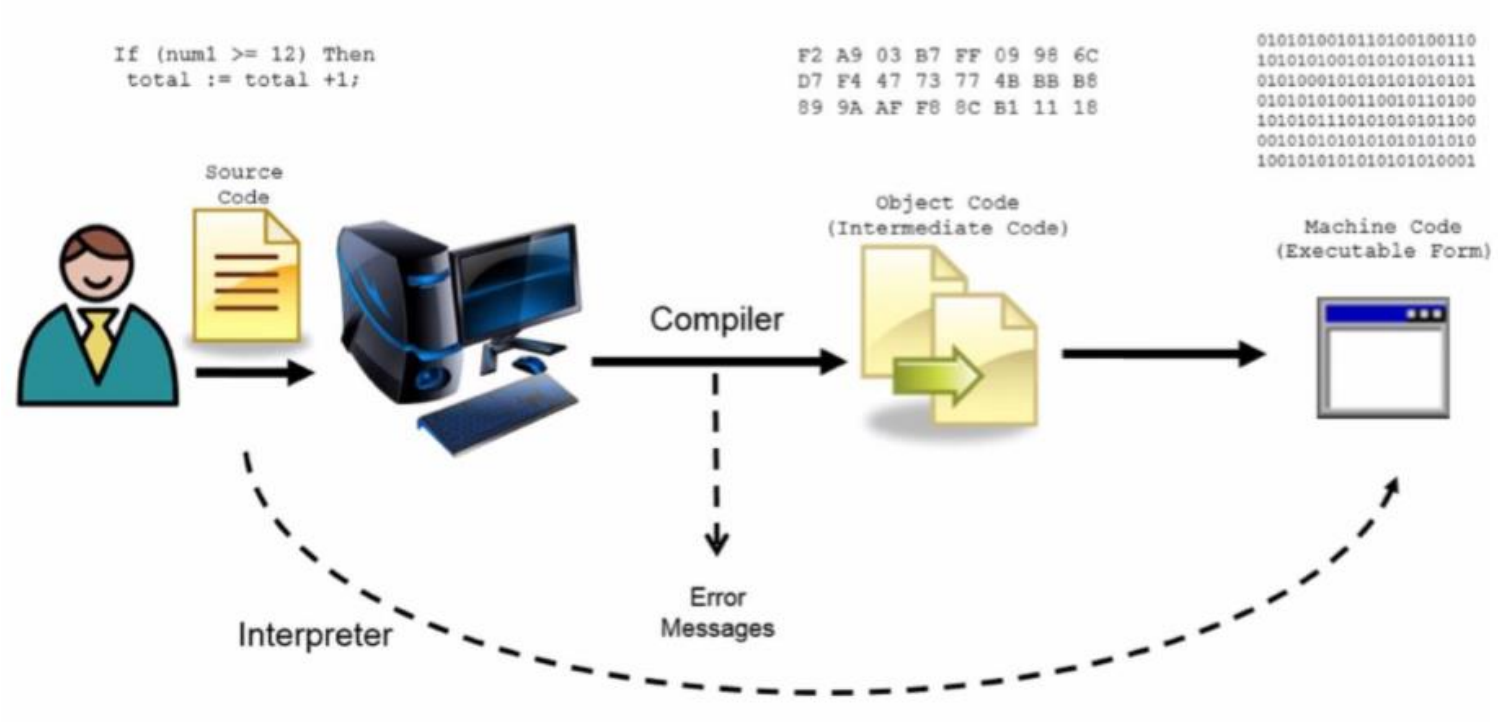
Key question: How does a VB program become the binary code that a computer can execute?

Interpreters

- Source code ---machine code
- Good for debugging
- Runs one line of code at a time
- Stops at first error
- Some security issues
- Can be slower
- Good for long large blocks of code
- Takes one line of code, translates it, then runs it right away.

Compiler

- Source code---object---machine code
- Can be slow
- Is more compatible with platforms
- Object code is faster
- For an update source code is changed then the object code is
- Not good for debugging and it runs code all at once
- Takes source code, translates it all into object code before allowing it to run

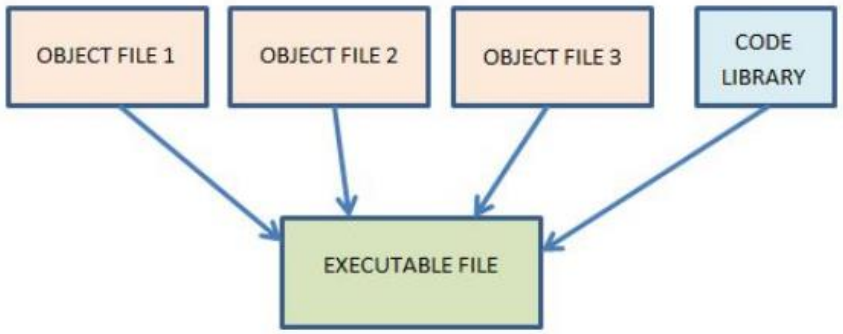


Key question: What is the purpose of a linker?

In compilation, we have a group of object files and libraries. These need to be combined together to create a functioning program. The program that does this is called the **linker**.

The linker is provided with a text file listing all of the object files and libraries that need to be connected together. This text file is called a 'make file', and is generally produced by the compiler automatically.

The result is an executable file - a machine code translation of the full program that can now be understood and carried out by the CPU.



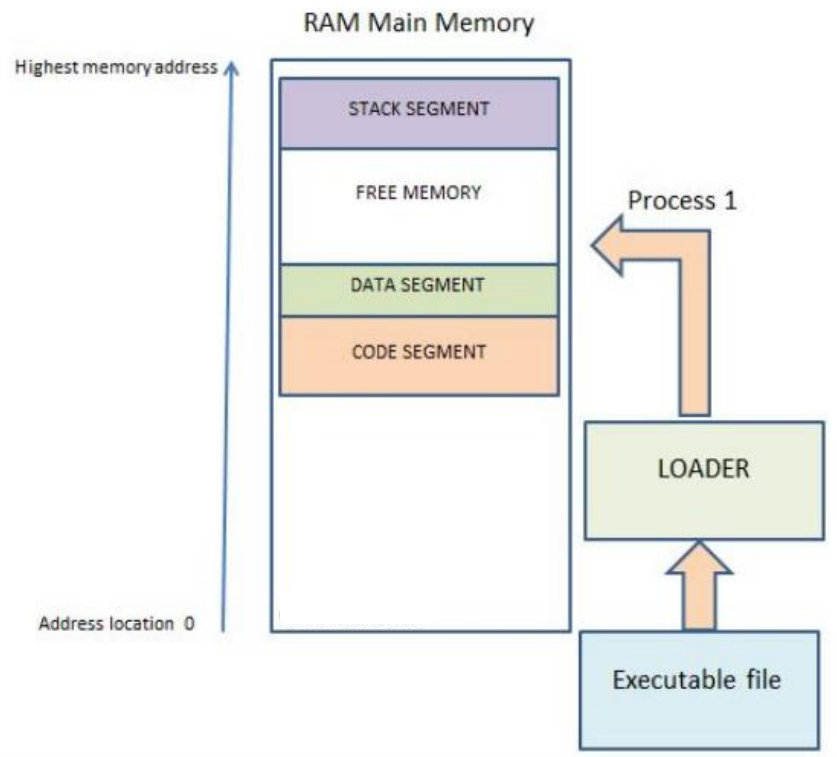
Key question: What is the purpose of a loader?

After an executable file has been created, it has to be loaded into main memory in order to run it. This is done by a utility program within the operating system called a **loader**.

The loader is part of the memory segmentation role of the operating system. It first creates a code segment to hold the program itself, then it creates a data segment for the constant and variables it is going to use and also a stack segment for handling procedure calls.

The machine code of the executable file defines the memory addresses of each line of code relative to one another, allowing it to jump from one point to another according to the program flow.

One of the duties of the loader is to translate these 'relative addresses' into 'absolute addresses', pointing the CPU to the exact locations in main memory.



Key question: What are the advantages of function libraries to a programmer?

- ✓ No need to write some procedures from scratch which saves time.
- ✓ Because the procedures have already been tested, they are unlikely to contain errors.
- ✓ The application itself can remain small and compact.
- ✓ Allows code to be shared with other applications that make use of the same procedures.
- ✓ An external library procedure can be updated without needing to re-compile the application.
- ✓ Library functions can be written in the most efficient language for the job.
- ✓ Library routines are connected to the program using a linker.
- ✓ The addresses of library routines are handled by the loader when the program is run.

X The library has to be well-written and robust or it will impair all applications making use of it.
X Specialist libraries for engineering, science and finance can be very expensive.
X For run-time loading the library has to be present.

Typical exam questions

1. State two pieces of utility software a secondary school would need to install on its computer systems. For each piece of software justify why it would be needed. **[6]**

Utility software 1:

Justification:

Utility software 2:

Justification:

2. One of the stages of code generation by a compiler is optimisation. Describe what optimisation does. **[2]**

3. Explain why interpreters are used in a computer system. **[2]**

4. Interpreters and Compilers are both examples of Translators. Explain one way in which they are similar and one way in which they differ. **[4]**

Component 1 | 1.2.2| Application generation

Target: Overall grade:

Minimum expectations & learning outcomes

☐	Terms 58-74 from your A Level Key Terminology should be included and formatted.
☐	You must describe several examples of application and utility software.
☐	You must include a clear comparison (e.g. in table form) of the advantages and disadvantages of open vs closed source software.
☐	You must include a diagram that shows the relationship between the following terms: Translators, Interpreters, Compilers, Assemblers, Linkers, Loaders, Libraries, Immediate Code, Source Code, Machine Code, Object Code
☐	You must explain what happens at each stage of the compilation process including lexical and syntax analysis.
☐	You must explain at least two scenarios in which an interpreter and a compiler might be used.
☐	You must identify the advantages of using library routines to the programmer and explain why a linker is needed.
☐	Answer the exam questions.

Feedback

Breadth	Depth	Presentation	Understanding
☐ All	☐ Analysed	☐ Excellent	☐ Excellent
☐ Most	☐ Explained	☐ Good	☐ Good
☐ Some	☐ Described	☐ Fair	☐ Fair
☐ Few	☐ Identified	☐ Poor	☐ Poor

Comment & action required

Component 1 | 1.2.2| Application generation

Reflection & Revision checklist

Confidence	Clarification
☹ 😐 ☺	Candidates need to understand the purpose of applications, and should have knowledge and experience of a range of different application software (for example database, word processor, web browser, graphics manipulation etc.).
☹ 😐 ☺	Candidates should be able to recommend the use of specific and generic applications for given scenarios, justifying their use and function(s) for a scenario.
☹ 😐 ☺	Candidates need to understand the purpose and role of utility software in a computer system.
☹ 😐 ☺	Candidates should be familiar with a range of utility software (e.g. disk defragmentation, file management, device driver, system cleanup, security etc.)
☹ 😐 ☺	Candidates need to be able to explain the differences between open and closed source software, the benefits and drawbacks to creator and user of each of the licensing models, and be able to recommend which is used (with justification) for a specific scenario.
☹ 😐 ☺	Candidates need to understand the need for translators when writing programs.
☹ 😐 ☺	Candidates need to have knowledge of the differences in operation of interpreters and compilers, from these they need to be able to assess the benefits and drawbacks of using each type, and recommend with justification which should be used in a specific scenario.
☹ 😐 ☺	Candidates need to understand the role of an assembler and how it differs from interpreters and compilers.
☹ 😐 ☺	Candidates need to understand that there are a number of stages involved in compilation.
☹ 😐 ☺	Candidates need to understand how lexical analysis works and how the code is converted into tokens with the removal of unnecessary elements (e.g. comments and whitespace).
☹ 😐 ☺	Candidates need to understand how syntax errors are identified and reported at the end of the syntax analysis.
☹ 😐 ☺	Candidates need to understand how the abstract syntax tree will be fed into the next stage of code generation, and that the object code is then created.
☹ 😐 ☺	Candidates need to understand why optimisation is important and how the results of lexical analysis feeds into syntax analysis, and how the tokens are checked to ensure they meet the during (and after) code generation.
☹ 😐 ☺	Candidates need to understand what code libraries are, how they are used and the benefits and drawbacks from using libraries.
☹ 😐 ☺	Candidates should have experience of using libraries to write programs.