1. Present sample C++ source code to demonistrate windows device driver code (Any device driver)

// mydevice.h

#ifndef \_MYDEVICE\_H\_

#define \_MYDEVICE\_H\_

#include <ntddk.h>

class MyDevice {

public:

MyDevice(PDEVICE\_OBJECT deviceObject);

~MyDevice();

NTSTATUS CreateDevice();

NTSTATUS DispatchRead(PIRP Irp);

NTSTATUS DispatchWrite(PIRP Irp);

NTSTATUS DispatchClose(PIRP Irp);

private:

PDEVICE\_OBJECT m\_deviceObject;

UNICODE\_STRING m\_deviceName;

UNICODE\_STRING m\_symbolicLinkName;

ULONG m\_bufferSize;

PVOID m\_buffer;

};

#endif // \_MYDEVICE\_H\_

--------------------------------------

// mydevice.cpp

#include "mydevice.h"

MyDevice::MyDevice(PDEVICE\_OBJECT deviceObject) {

m\_deviceObject = deviceObject;

m\_bufferSize = 1024;

m\_buffer = ExAllocatePoolWithTag(NonPagedPool, m\_bufferSize, 'MYDT');

}

MyDevice::~MyDevice() {

ExFreePoolWithTag(m\_buffer, 'MYDT');

}

NTSTATUS MyDevice::CreateDevice() {

UNICODE\_STRING devName, symLinkName;

NTSTATUS status;

RtlInitUnicodeString(&devName, L"\\Device\\MyDevice");

RtlInitUnicodeString(&symLinkName, L"\\??\\MyDevice");

status = IoCreateDevice(

DriverObject,

sizeof(MyDevice),

&devName,

FILE\_DEVICE\_UNKNOWN,

0,

FALSE,

&m\_deviceObject

);

if (NT\_SUCCESS(status)) {

status = IoCreateSymbolicLink(&symLinkName, &devName);

if (!NT\_SUCCESS(status)) {

IoDeleteDevice(m\_deviceObject);

}

}

return status;

}

NTSTATUS MyDevice::DispatchRead(PIRP Irp) {

ULONG bytesRead = 0;

PIO\_STACK\_LOCATION irpStack = IoGetCurrentIrpStackLocation(Irp);

if (irpStack->Parameters.Read.Length <= m\_bufferSize) {

RtlCopyMemory(Irp->AssociatedIrp.SystemBuffer, m\_buffer, irpStack->Parameters.Read.Length);

bytesRead = irpStack->Parameters.Read.Length;

}

Irp->IoStatus.Status = STATUS\_SUCCESS;

Irp->IoStatus.Information = bytesRead;

IoCompleteRequest(Irp, IO\_NO\_INCREMENT);

return STATUS\_SUCCESS;

}

NTSTATUS MyDevice::DispatchWrite(PIRP Irp) {

ULONG bytesWritten = 0;

PIO\_STACK\_LOCATION irpStack = IoGetCurrentIrpStackLocation(Irp);

if (irpStack->Parameters.Write.Length <= m\_bufferSize) {

RtlCopyMemory(m\_buffer, Irp->AssociatedIrp.SystemBuffer, irpStack->Parameters.Write.Length);

bytesWritten = irpStack->Parameters.Write.Length;

}

Irp->IoStatus.Status = STATUS\_SUCCESS;

Irp->IoStatus.Information = bytesWritten;

IoCompleteRequest(Irp, IO\_NO\_INCREMENT);

return STATUS\_SUCCESS;

}

NTSTATUS MyDevice::DispatchClose(PIRP Irp) {

Irp->IoStatus.Status = STATUS\_SUCCESS;

IoCompleteRequest(Irp, IO\_NO\_INCREMENT);

return STATUS\_SUCCESS;

}

***This code*** demonstrates a simple character device driver for Windows. The MyDevice class encapsulates the device-specific functionality, including the creation of the device object and symbolic link, as well as the handling of read, write, and close I/O requests.

The CreateDevice() method creates the device object and the symbolic link, allowing the device to be accessed from user mode. The DispatchRead(), DispatchWrite(), and DispatchClose() methods handle the corresponding I/O requests, reading and writing to the device's internal buffer.

2. What is difference between OpenGL and DirectX. (only 2 differences)

**Cross-platform vs. Windows-specific:**

- OpenGL is a cross-platform graphics API, which means it can be used on various operating systems, including Windows, macOS, and Linux.

- DirectX is a Microsoft-specific graphics API, primarily designed for Windows operating systems.

**Hardware Abstraction:**

- OpenGL provides a lower-level, more hardware-agnostic interface, allowing developers to have more control over the graphics hardware.

- DirectX is more tightly integrated with the Windows operating system and provides a higher-level, more abstracted interface, making it easier for developers to work with Windows-specific hardware and features.

1. List out C++ based Automotive Embedded software tools

**- AUTOSAR (Automotive Open System Architecture)**: AUTOSAR is a standardized software architecture for automotive electronic control units (ECUs). It is developed and maintained by a partnership of automotive manufacturers, suppliers, and tool vendors, and it is written in C++.

**- MISRA C++**: MISRA C++ is a set of coding guidelines and best practices for the use of the C++ programming language in the development of safety-critical and reliability-critical systems, including automotive applications.

**- Qt Automotive Suite**: Qt Automotive Suite is a C++ software framework developed by The Qt Company, specifically designed for building user interfaces and applications for automotive infotainment systems and digital instrument clusters.

**- CAN (Controller Area Network) Libraries**: There are several C++ libraries and frameworks available for working with the CAN bus protocol, which is widely used in automotive systems, such as SocketCAN, CANopen, and CANLIB.

**- Boost.Asio**: Boost.Asio is a C++ library for network and low-level I/O programming, which can be used for building automotive communication protocols and systems.

**- Automotive Grade Linux (AGL)**: Automotive Grade Linux is a collaborative open-source project that provides a Linux-based software stack for automotive applications, including a C++ based development environment.

**- AUTOSAR Adaptive Platform**: The AUTOSAR Adaptive Platform is a new standard in the AUTOSAR ecosystem, which is designed for more complex, high-performance automotive applications, and it is primarily implemented in C++.

**- MATLAB/Simulink**: While not a pure C++ tool, MATLAB and Simulink are widely used in the automotive industry for model-based design, simulation, and code generation, which can include C++ code generation for embedded systems.

1. Explain the pitfalls with C files compare to DBMS with example programs.

**Pitfalls of Using C Files vs. DBMS:**

1. **Data Integrity**: C files lack built-in mechanisms for ensuring data integrity, unlike DBMS which provide features like transactions and data constraints.

Example: A file write operation interruption can lead to data loss in C files, whereas a DBMS maintains data consistency.

1. **Data Querying and Manipulation**: C files require manual file operations, making it tedious and error-prone to work with complex data structures and queries. DBMS offer declarative query languages and optimized data access.

Example: Retrieving users by email in C files requires linear search, while a DBMS can perform this efficiently using SQL.

1. **Concurrency and Scalability**: Managing concurrent access is challenging in C files, requiring custom locking mechanisms. DBMS provide built-in concurrency control and can scale to handle large data and high loads.

Example: Two programs writing to the same C file simultaneously can corrupt the data, whereas a DBMS would handle this scenario gracefully.

5.Gcc Vs CLANG

| **Feature** | **GCC** | **Clang** |
| --- | --- | --- |
| **Compiler Architecture** | Monolithic design, with all components (front-end, optimizer, and back-end) tightly integrated. | Modular design, with the front-end, optimizer, and back-end as separate components based on the LLVM infrastructure. |
| **Compilation Speed** | Generally slower, especially for large codebases. | Faster at compiling code compared to GCC, due to its modular design and efficient parsing and code generation. |
| **Diagnostics and Error Messages** | Error messages can be more technical and harder to understand for developers. | Provides more helpful and user-friendly error messages, making it easier for developers to debug their code. |
| **Language Support** | Supports a wider range of programming languages, including C, C++, Objective-C, Fortran, Ada, and others. | Primarily focuses on C, C++, Objective-C, and Objective-C++, with limited support for other languages. |
| **Conformance to Standards** | Sometimes provides extensions or non-standard features, which can be both a blessing and a curse. | Generally more strict in its adherence to language standards. |
| **Compiler Optimization** | Provides a wide range of optimization flags and settings, but the resulting optimized code may differ slightly from Clang. | Provides a wide range of optimization flags and settings, with the potential for slightly different optimized code compared to GCC. |
| **Cross-platform Support** | Available on a wide range of platforms, including Linux, macOS, and Windows (through MinGW). | Primarily focused on Linux and macOS, with limited support for Windows. |

6.CLANG vs LLVM

| **Feature** | **Clang** | **LLVM** |
| --- | --- | --- |
| **Purpose** | Clang is a C, C++, Objective-C, and Objective-C++ compiler front-end. | LLVM is a collection of modular and reusable compiler and toolchain components. |
| **Architecture** | Clang is a front-end for the LLVM compiler infrastructure. | LLVM is a collection of compiler and toolchain components, including the LLVM Intermediate Representation (IR) and a set of optimizations and code generation tools. |
| **Language Support** | Clang primarily focuses on C, C++, Objective-C, and Objective-C++. | LLVM supports a wide range of programming languages, including C, C++, Objective-C, Rust, Swift, and more, through various front-ends. |
| **Compilation Process** | Clang handles the front-end tasks of the compilation process, such as parsing, semantic analysis, and generating LLVM IR. | LLVM handles the back-end tasks of the compilation process, such as optimization, code generation, and target-specific code generation. |
| **Diagnostics** | Clang is known for providing more helpful and user-friendly error messages. | LLVM does not directly handle diagnostics, as this is the responsibility of the front-end (like Clang) that feeds into the LLVM infrastructure. |
| **Licensing** | Clang is released under the University of Illinois/NCSA Open Source License, which is compatible with the GPL. | LLVM is released under the Apache License, Version 2.0, which is compatible with the GPL and other open-source licenses. |
| **Project Scope** | Clang is focused on being a high-performance C, C++, Objective-C, and Objective-C++ compiler. | LLVM is a broader project that includes not only the compiler infrastructure but also other tools and libraries, such as a linker, debugger, and static analyzer. |

7.What is FPGA, applications of FPGA. Difference between FPGA vs ASIC

**PGA (Field-Programmable Gate Array)** is a type of semiconductor device that can be programmed and reprogrammed to perform various digital functions. It consists of an array of configurable logic blocks (CLBs) and programmable interconnects that can be used to implement different digital circuits and systems.

**Applications of FPGA:**

1. Prototyping and Rapid Prototyping: FPGAs are widely used for prototyping new hardware designs, as they can be quickly programmed and reprogrammed to test different hardware configurations.
2. Digital Signal Processing (DSP): FPGAs are well-suited for DSP applications, such as image and video processing, due to their parallel processing capabilities.
3. Embedded Systems: FPGAs are used in embedded systems, where their flexibility and reconfigurability are valuable for custom hardware designs.
4. Networking and Communications: FPGAs are used in various networking and communication applications, such as network switches, routers, and wireless base stations.
5. Military and Aerospace: FPGAs are used in military and aerospace applications, where their reliability, flexibility, and radiation tolerance are important.
6. Cryptocurrency Mining: FPGAs are used in cryptocurrency mining, as they can provide a balance between power efficiency and performance.

**The difference between FPGA and ASIC**

| **Feature** | **FPGA** | **ASIC** |
| --- | --- | --- |
| **Programmability** | FPGAs are programmable and can be reconfigured to perform different functions after manufacturing. | ASICs are designed for a specific function and cannot be easily reconfigured or reprogrammed after manufacturing. |
| **Development Time and Cost** | Developing an FPGA design is generally faster and less expensive compared to developing an ASIC. | Developing an ASIC is a longer and more expensive process, as it requires custom design and manufacturing. |
| **Performance and Power Efficiency** | FPGAs can provide good performance, but they may not be as power-efficient as ASICs for specific applications. | ASICs are generally more power-efficient and can offer higher performance for their intended applications, as they are designed for a specific function. |
| **Flexibility and Reconfigurability** | FPGAs are highly flexible and can be reconfigured, allowing for changes and updates to the design. | ASICs are not reconfigurable, as they are designed for a specific function and cannot be easily modified. |
| **Volume and Cost** | FPGAs are suitable for low to medium volume production, as the per-unit cost is generally higher compared to ASICs. | ASICs are more suitable for high-volume production, as the per-unit cost can be significantly lower compared to FPGAs. |

1. Write Caesar cipher Encryption and Decryption algorithms in C language. The algorithm shall able to encrypt file content and decrypt file content

2. Write C program to read Image file and change the Image file with encrypt the message “Meet on August 20” using Substitution cipher algorithm. Write program to decrypt message same

1. Write compilation stages & procedure in c++

**Preprocessing:**

- Preprocessor handles directives, expands macros, and includes headers.

- Output: preprocessed source code (.i)

g++ -E source\_file.cpp -o preprocessed\_file.cpp

**Compilation:**

- Compiler generates assembly code from preprocessed source.

- Output: assembly language file (.s)

g++ -c preprocessed\_file.cpp -o object\_file.o

**Assembly:**

- Assembler converts assembly to object file.

- Output: object file (.o)

g++ -S source\_file.cpp -o assembly\_file.s

**Linking:**

- Linker combines object files and resolves external references.

- Output: executable file

g++ object\_file.o -o executable\_file

**Combine the compilation and linking stages into a single step: g++ source\_file.cpp -o executable\_file**

**Run file: ./excutable\_file**

1. Why we need to compile the c++ code ?
2. **Machine-Readable Code**: C++ is human-readable, but computers need machine code to execute.
3. **Optimization**: Compilers can optimize the code for better performance.
4. **Linking**: Compilers link external libraries to create the final executable.
5. **Error Checking**: Compilation includes checks to identify and report errors in the source code.
6. **Platform-Independence**: Compiled executables can run on different hardware and software platforms.
7. **Memory Management**: Compilers generate code to handle memory allocation and deallocation.
8. Can i use c++ compiler to compile C code?

**Yes**, we can use a C++ compiler to compile C code. This is possible because C++ is a superset of the C programming language, meaning that any valid C code is also valid C++ code.

1. Can i use C compiler to compile C++ code?

**Yes**, It is possible to use a C compiler to compile some C++ code, but this is not recommended because the C compiler cannot handle the advanced features and libraries of C++. Therefore, it is best to use a dedicated C++ compiler, such as g++ or clang++, to ensure that all language-specific parameters are handled correctly and that optimized and correct executables are produced.

5. Is the Python object oriented or non-object oriented programming language?

Python is an object-oriented programming (OOP) language, but it also supports non-object-oriented programming paradigms.