1. Write a program to convert ANSI string to Wide character string and vice versa. Use `MultiByteToWideChar` and `WideCharToMultiByte` for conversion and `IsTextUnicode` for testing the results. [Note that `IsTextUnicode` output may not be always correct.]
2. Implement a program where command line arguments passed to program are read via `GetCommandLine` and converted to argv style by `CommandLineToArgvW`.
3. Write a program to print environment variables passed to the console application. Also demonstrate the usage of `ExpandEnvironmentStrings` API.
4. Write a program to open an existing file with `CreateFile` and use `ReadFile` to read the contents of file till EOF and print the contents to console.
5. Write a program to create a new file with `CreateFile` and use `WriteFile` to write text to file. Verify the written text by manually opening file in notepad/wordpad etc.
6. Write a program to create a process using `CreateProcess` API. Child process can be any your previously implemented programs.
7. Write a program which uses the `CreateProcess` API to create two child processes (say calc.exe and notepad.exe). Print HANDLE value and process ID of each child process.
8. Create a named event object with CreateEvent. Verify the presence of the created event with WinObj (Hint: Run WinObj as Administrator and Look for event in path similar to: \ > Sessions > * > BaseNamedObjects. * can be integer). Also verify how kernel object is deleted when no longer used.
9. Write a program to create a child process. Use `GetCurrentProcessId`, `GetCurrentThreadId`, `GetProcessId`, `GetThreadId`, `GetProcessIdOfThread` to print process and thread information in each of the process.
10. Write a program to create child process which inherits handle of the object (say file object or event object) created in the parent process. This can be done via setting `SECURITY_ATTRIBUTES` structures `lpSecurityDescriptor` parameter when creating object and CreateProcess's `bInheritHandles` parameter when creating child process. Note that child process has no idea of what handles are inherited by it.
11. Implement two programs say program1 and program2. Open a file from within program1 and make file object handle inheritable. Execute the program2 as a child process of program1 using CreateProcess (enable handle inheritance for child process). Pass the handle of the opened file in process1 to process2 by using `lpCommandLine` parameter of `CreateProcess` API. Collect the handle of file in process2 (this will be in the `argv[]`) and write data to the file from within proces2. [Hint: Use `CreateProcess(argv[2], (LPTSTR)&hFile,...)` to pass handle to child process and collect handle in child process with `_tprintf( _T( "argv[0] %lu\n"), (unsigned long)*argv[0] );`]
12. Write a program to create a thread via `CreateThread` API. Print some message in the thread function.
13. Write a program to create a thread in suspended state. Resume the thread after 5 seconds wait. The thread should print integers from 1 to 1000 and return normally. Use `getchar()` in process's primary thread to wait for user input and allow time for thread to return.
14. Write a program to create a secondary thread and wait in the process's primary thread for the secondary thread to return using `WaitForSingleObject`. Terminate secondary thread by using `ExitThread` API from within secondary thread. Print the exit code of the secondary thread using `GetExitCodeThread` API in process's primary thread.
15. Write a program to create a secondary thread and wait in the process's primary thread for the secondary thread to return using `WaitForSingleObject`. Terminate the secondary thread using `TerminateThread` API from within process's primary thread. Print the exit code of the secondary thread using `GetExitCodeThread` API in process's primary thread.
16. Write a program to create a child process with `HIGH_PRIORITY_CLASS`. Change the priority of child process to `BELOW_NORMAL_PRIORITY_CLASS` from within child process. Also create a thread within

the child process with `THREAD_PRIORITY_LOWEST` relative priority. Verify the results with Process Explorer from Sysinternals. [Hint: use two program scenero as used in question 11].

17. Write a program to increment the value of a shared integer by two threads of the same process. Print the final value of shared integer in the process's primary thread. Print the case when final value of shared integer is inconsistent.
18. Write a program to increment a shared integer via `Interlocked`* functions by two threads of the same process. Print the final value of shared integer. Verify if final value of shared integer is inconsistent.
19. Write a program to modify shared variable between two threads guarded by critical section. Use `InitializeCriticalSection` or `InitializeCriticalSectionAndSpinCount`, `DeleteCriticalSection`, `EnterCriticalSection`, `LeaveCriticalSection`. Wait for both the threads to exit before exiting process primary thread.
20. Write a program where there is a shared queue of fixed size between reader and writer threads. Say the queue size is 5, readers are 2 and writers are 4. Synchronize the execution of reader and writer threads via Slim Reader-Writer locks and condition variables. Use `InitializeSRWLock`, `AcquireSRWLockExclusive`, `ReleaseSRWLockExclusive`, `AcquireSRWLockShared`, `ReleaseSRWLockShared`, `SleepConditionVariableSRW`, `WakeConditionVariable`, `WakeAllConditionVariable` APIs.
21. Implement a dynamic link library (DLL) exporting four functions add, subtract, divide and multiply each taking 2 integer arguments and returning an integer value. Implement an application which implicit links to dll and uses the add, subtract, divide and multiply functions of DLL.
    a. Common header file format
    ```
    #pragma once
    #ifdef MYLIB_EXPORTS
    #define MYLIBAPI extern "C" __declspec(dllexport)
    #else
    #define MYLIBAPI extern "C" __declspec(dllimport)
    #endif
    MYLIBAPI int add( int nLeft, int nRight );
    MYLIBAPI int sub( int nLeft, int nRight );
    ```
    b. Use VC++ Directories > Include directories to include path of DLL header file
    c. Use VC++ Directories > Library directories to include path of DLL .lib file
    d. Use Linker > Additional Dependencies to link application with .lib file of DLL
    e. Verify functions exported by DLL with `<dumpbin -exports file.lib>`
    f. Verify functions imported by application using DLL with `<dumpbin -imports appfile.exe>`
22. Implement a dynamic link library exporting four functions add, subtract, divide and multiply and DllMain entry point to handle `DLL_PROCESS_ATTACH`, `DLL_THREAD_ATTACH`, `DLL_THREAD_DETACH`, `DLL_PROCESS_DETACH` notifications. Implement an application which explicitly links to DLL and uses the add, subtract, divide and multiply functions of DLL.
23. Implement static library exporting two functions add and subtract. Implement an application which uses the add and subtract functionality of static library.
24. Write a program to create a thread and print the reason for `WaitForSingleObject` return. Print results with.
    a. timeout interval `INFINITE`
    b. timeout interval `1000`
    c. timeout interval `1000` with `Sleep(2000)` inside thread
25. Write a program to create two threads and print the reason for `WaitForMultipleObjects` return. Print results with.

   a. `bWaitAll = TRUE` and timeout interval `INFINITE`
   b. `bWaitAll = FALSE` and timeout interval `INFINITE`, thread1 `Sleep(4000)` and thread2 `Sleep(3000)`
   c. `bWaitAll = FALSE` and timeout interval `INFINITE`, thread1 `Sleep(2000)` and thread2 `Sleep(3000)`
   d. `bWaitAll = FALSE` and timeout interval `2000`, thread1 `Sleep(4000)` and thread2 `Sleep(3000)`

26. Write a program to create a manual reset event in non-signaled state. Program should spawn 3 dummy threads for a) counting words b) checking spellings and c) checking grammar. All the three threads wait for an event to be set by process main thread which sets the event after opening a dummy file so that all three threads can run. Wait for all threads to exit in process's primary thread.
27. Change the above program for a scenario where all three threads can't access the data of dummy file in shared mode.
28. Write a program where 8 threads execute to get exclusive access to shared data via mutex. Make sure mutex object is initially not owned. Use the shared resource for 1 sec with `Sleep(1000)`. Use `WaitForSingleObject` to get the ownership of mutex in threads.
29. Write a program where only 2 out of 4 threads at a time are allowed to access shared resources. Use semaphore object to count the no of allowed threads to access shared resources. Use the shared resource for 200 ms with `Sleep(200)`. Use `WaitForSingleObject` to get the ownership of semaphore object.
30.