## Тема 4. Консольні програми у Windows

План
1. Принцип дії консолі Windows. Вхідний та екранні буфери консолі.
2. Консольна програма. Створення та налаштування консолі.
3. Високорівневі та низькорівневі функції для роботи з консоллю.

Windows дозволяє створювати програми, які не мають графічного інтерфейсу. Це додатки, які, по суті, є символьними програмами командного рядка. Такі програми називаються консольними додатками (Win32 console application), оскільки спілкування з ними відбувається через клавіатуру і екран у символьному режимі. Для запуску консольного додатка Windows створює окреме вікно – консоль.

Не слід плутати консольні додатки Windows з 16-розрядними DOS-програмами.

DOS-програми призначені для операційної системи DOS, виконуються в реальному режимі і мають доступ до 1 Мб пам'яті. Для запуску DOS-програм Windows використовує віртуальну DOS-машину VDM (Virtual DOS Machine).

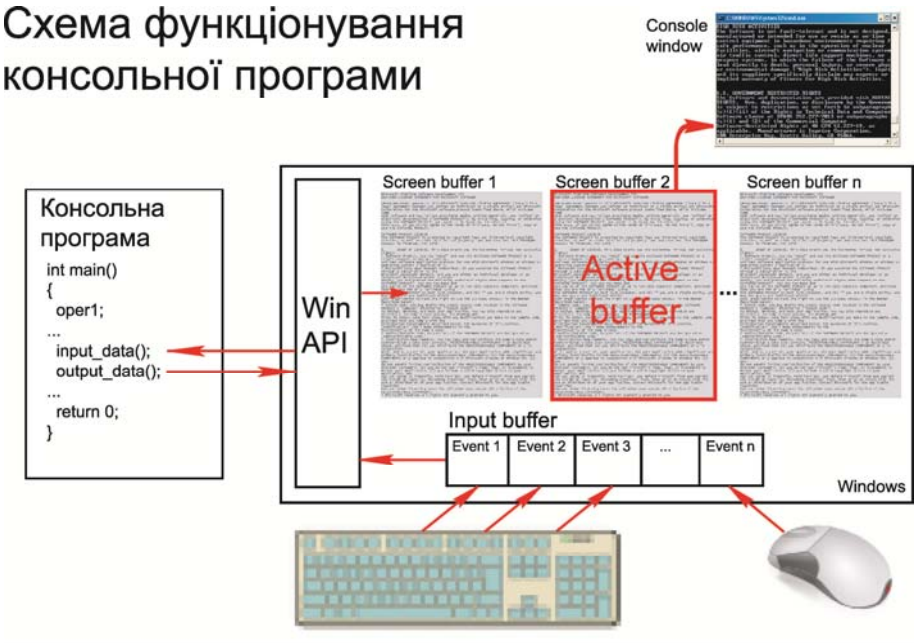
Консольні програми Windows виконуються в захищеному режимі, використовують плоску модель пам'яті, мають доступ до 4 Гб віртуального адресного простору і використовують будь-які ресурси Windows, за винятком графічного інтерфейсу. Це повноцінні 32-розрядні програми Windows.

Останнім часом з'явився ще один тип консольних додатків (CLR console application). Це консольні додатки для ***загальномовного виконавчого середовища*** (Common Language Runtime), яке реалізується на платформі .NET.

### 1. Принцип дії консолі Windows. Вхідний та екранні буфери консолі

Консоль складається з вхідного буфера і одного або більше буферів екрану. Вхідний буфер містить чергу вхідних записів, кожен з яких містить інформацію про вхідну подію. Вхідна черга завжди включає події натискання та відпускання клавіш. Вона може також включати події миші (переміщення вказівника і натискання та відпускання кнопок) і події, впродовж яких дії користувача впливають на розмір активного буфера екрану. Буфер екрану - двомірний масив даних символів і кольорів для виведення у вікно консолі.

Консоль працює з трьома стандартними пристроями вводу/виводу: INPUT, OUTPUT і ERROR. Це пристрої вводу даних, виводу даних і виводу повідомлень про помилки.



### *Вхідний буфер консолі*

Кожна консоль має ***вхідний буфер***, який містить *чергу записів* вхідних подій. Коли вікно консолі має фокус, то будь-яка вхідна подія (натискання клавіші, рух миші, клацання кнопки миші) формується як вхідний запис, який розміщується у вхідному буфері консолі.

***Вхідний запис*** це структура, що містить інформацію про тип події, яка відбулася (подія клавіатури, миші, зміни розмірів вікна, фокусу, або меню), а також специфічні деталі про подію.

```
typedef struct _INPUT_RECORD {
    WORD EventType;
    union {
        KEY_EVENT_RECORD KeyEvent;
        MOUSE_EVENT_RECORD MouseEvent;
        WINDOW_BUFFER_SIZE_RECORD WindowBufferSizeEvent;
        MENU_EVENT_RECORD MenuEvent;
        FOCUS_EVENT_RECORD FocusEvent;
    } Event;
} INPUT_RECORD;
```

Елемент EventType в структурі INPUT_RECORD вказує на тип події, який міститься в записі (wincon.h).

```
KEY_EVENT                0x0001    // Event містить запис key event
MOUSE_EVENT              0x0002    // Event містить запис mouse event
WINDOW_BUFFER_SIZE_EVENT 0x0004    // Event містить запис window change event
MENU_EVENT               0x0008    // Event містить запис menu event
FOCUS_EVENT              0x0010    // Event містить запис focus change
```

***Події клавіатури*** генеруються, коли натискається або звільняється будь-яка клавіша, включаючи керуючі клавіші. Правда, клавіша ALT має спеціальне значення для Windows, коли вона натискається і звільняється без комбінації з іншим символом, тому ця подія не передається в додаток. Також, комбінація клавіш CTRL+C не передається, якщо вхідний дескриптор перебуває в режимі обробки.

***Події миші*** генеруються кожного разу, коли користувач переміщає мишу або натискає чи звільняє одну із клавіш миші. Події миші поміщаються у вхідний буфер, якщо виконуються наступні умови:

- Вхідний режим консолі встановлений в ENABLE_MOUSE_INPUT (заданий за умовчанням режим).
- Вікно консолі має фокус.
- Покажчик миші в межах границь вікна консолі.

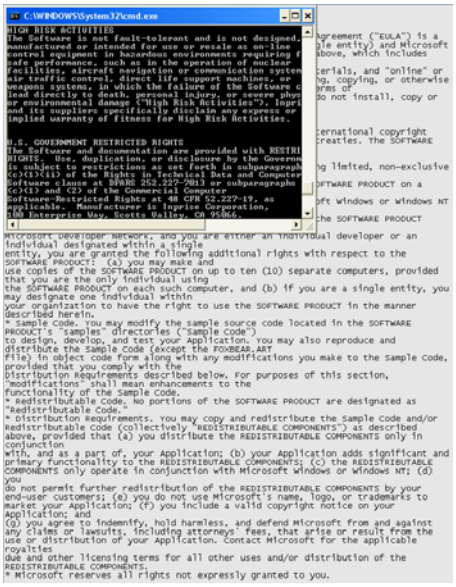
***Події фокусу і меню*** розміщуються у вхідному буфері консолі для внутрішнього використання системою і мають бути проігноровані додатками.

### *Екранний буфер консолі*

***Буфер екрану*** це двомірний масив символів і їх кольорів для виведення у вікно консолі. Консоль може мати кілька буферів екрану. Активний буфер екрану – той, який відображається на екрані.

Система створює буфер екрану кожного разу, коли створює нову консоль. Можна використати функцію CreateConsoleScreenBuffer, щоб створити додаткові буфери екрану для консолі. Новий буфер екрану не активний, доки його дескриптор не буде вказаний у виклику функції SetConsoleActiveScreenBuffer. Проте, буфери екрану можуть бути доступні для читання і запису незалежно від того, чи вони активні чи ні.

Кожен буфер екрану має свій власний двомірний масив даних про символи. Дані кожного символу зберігаються в структурі CHAR_INFO, яка визначає символ Unicode або ANSI і кольори символу і фону, в яких символ відображається.



Набір властивостей екранного буферу, може бути встановлений незалежно для кожного буфера екрану. Правда це також означає, що заміна активного буфера екрану може мати драматичний ефект при появі вікна консолі. Властивості буферу екрану, включають:

- *Розмір буферу екрана* (кількість рядків і стовпчиків символів).
- *Текстові атрибути* (кольори символу і фону для відображення тексту, який записується функціями WriteFile або WriteConsole).
- *Розмір вікна і розташування* (прямокутна область буфера екрану, який відображується у вікні консолі).
- Позиція *курсора*, вигляд, і видимість.
- Режими виводу (ENABLE_PROCESSED_OUTPUT і ENABLE_WRAP_AT_EOL_OUTPUT). Для більш конкретної інформації див. Високорівневі Режими Консолі.

Коли створюється буфер екрану, він містить пропуски у всіх позиціях. Його курсор видимий і розташований на початку координат буфера (0,0), і вікно консолі розташоване своїм верхнім лівим кутом на початку буфера. Розмір буфера екрану, розмір вікна, текстові атрибути, і вигляд курсора, визначаються користувачем або системними значеннями по умовчанню. Щоб отримати поточні значення різних властивостей буфера екрану, використовують функції GetConsoleScreenBufferInfo, GetConsoleCursorInfo, і GetConsoleMode.

## 2. Консольна програма. Створення та налаштування консолі

Консольна програма як і будь-яка стандартна програма C++ (ANSI/ISO) обов'язково містить *головну функцію* **main()**, із якої починається виконання програми, і яка викликає інші функції. Головна функція **main()** може мати також назву **wmain()**, якщо програма використовує символи Unicode та узагальнену назву **_tmain()**, тоді програма може бути скомпільована як для кодування ANSI, так і Unicode.

Головна функція реалізує основний алгоритм програми, який полягає у послідовному виконанні операторів програми і виклику інших функцій. Після виконання кожної функції керування повертається головній функції. Завершується виконання програми оператором **return()**, який передає керування операційній системі і повертає код завершення.

***Особливістю консольної програми*** є те, що операції вводу/виводу відбуваються у моменти, визначені програмою. Оскільки прямий доступ до портів вводу/виводу у Win32 заборонений, використовуються функції WinAPI, призначені для роботи з консоллю.

### *Створення консолі*

Microsoft Windows створює нову консоль, коли вона запускає процес символьного режиму, точкою входу якого є функція main(), наприклад, командний процесор cmd.exe. Коли командний процесор запускає новий процес консолі, можна визначити чи система створює нову консоль для нового процесу чи вона успадковує консоль командного процесора. GUI-програми Windows не мають своєї консолі за умовчанням, але її можна створити, якщо вона потрібна.

Процес може *створити консоль*, використовуючи один з наступних методів:

- Процес з графічним інтерфейсом (GUI) або консольний процес, не підключений до консолі, може використати функцію **AllocConsole()**, щоб створити нову консоль.
- Процес GUI або консоль може використати функцію **CreateProcess**, щоб створити новий процес і встановити прапорець CREATE_NEW_CONSOLE, аби дати вказівку системі створити нову консоль.

Якщо запустити консольний додаток із командного рядка, то він успадковує батьківську, тобто "чужу" консоль. Для створення своєї консолі потрібно від'єднатися від чужої консолі і створити власну. Для цього використовуються наступні функції:

```
BOOL FreeConsole(VOID)     – від'єднує процес від консолі
BOOL AllocConsole(VOID)    – виділяє нову консоль для процесу
```

Для роботи з консольними функціями потрібно одержати дескриптор консолі:

```
HANDLE GetStdHandle(DWORD nStdHandle);
```

де *nStdHandle* визначає пристрій, для якого одержують дескриптор:

```
STD_INPUT_HANDLE    (–10)  стандартний пристрій введення
STD_OUTPUT_HANDLE   (–11)  стандартний пристрій введення
STD_ERROR_HANDLE    (–12)  стандартний пристрій для помилок
```

**Наприклад:**

```
hStdIn = GetStdHandle(STD_INPUT_HANDLE);
hStdOut = GetStdHandle(STD_OUTPUT_HANDLE);
```

Після цього усі операції із консоллю здійснюються через її ***дескриптор***. Якщо функція GetStdHandle завершується фатально, то вона повертає `INVALID_HANDLE_VALUE`.

В UNIX та MS-DOS дескриптори ПВВ фіксовані (0, 1, 2).

### *Налаштування консолі*

Для налаштування параметрів консолі використовують такі функції:

| Функція | Опис |
|---|---|
| GetConsoleTitle()<br>SetConsoleTitle() | Одержати заголовок вікна консолі<br>Встановити заголовок вікна консолі |
| GetConsoleMode()<br>SetConsoleMode() | Одержати режим консолі<br>Встановити режим консолі |
| SetConsoleScreenBufferSize()<br>SetConsoleActiveScreenBuffer() | Змінити розмір екранного буфера.<br>Встановити активний буфер екрана |
| SetConsoleCursorPosition() | Встановити координати курсора |
| SetConsoleTextAttribute() | Встановити атрибути кольору тексту |

***Закрити консоль*** можна за допомогою функції **FreeConsole()**. Якщо інші процеси підключені до консолі, вона зберігається. Коли останній процес від'єднується від консолі, вона закривається.

## 3. Високорівневі та низькорівневі функції для роботи з консоллю

Функції Win32 API допускають два рівні доступу до консолі: *високорівневий* і *низькорівневий*.

***Високорівневі функції*** надають доступ до вхідного і екранного буферів консолі на рівні потоку символів.

***Низькорівневі функції*** надають можливість додаткам обмінюватися даними з вхідним і екранним буферами консолі.

### *Високорівневий підхід*

***Високорівневі функції введення*** повертають потік символів з клавіатури, відкидаючи службову інформацію та інші події.

```
BOOL ReadConsole(
  HANDLE hConsoleInput,               // дескриптор вхідного буфера консолі
  LPVOID lpBuffer,                    // адреса буфера прийому даних
  DWORD nNumberOfCharsToRead,         // кількість символів для читання
  LPDWORD lpNumberOfCharsRead,        // адреса числа прочитаних символів
  LPVOID lpReserved                   // зарезервовано, завжди NULL
 );
```

***Високорівневі функції виведення*** записують потік символів у буфер екрану, які відображаються в поточному розташуванні курсору.

```
BOOL WriteConsole(
  HANDLE hConsoleOutput,              // дескриптор екранного буфера консолі
  CONST VOID *lpBuffer,               // адреса буфера даних для виводу
  DWORD nNumberOfCharsToWrite,        // кількість символів для запису
  LPDWORD lpNumberOfCharsWritten,     // вказівник числа прочитаних символів
  LPVOID lpReserved                   // зарезервовано, завжди NULL
 );
```

Консольна програма може також використовувати файлові функції **ReadFile()** і **WriteFile()**. Різниця в тому, що **ReadFile()** і **WriteFile()** працюють із будь-якими потоковими даними (файли, канали) і оперують побайтно. Функції **ReadConsole** і **WriteConsole** працюють лише з дескрипторами консолі і оперують посимвольно (ANSI або Unicode).

### *Низькорівневий підхід*

***Низькорівневі функції введення*** забезпечують прямий доступ до вхідного буфера і дають можливість додаткам отримувати детальні дані про події клавіатури, миші та інші події. Зокрема, вони дозволяють:

- Одержати дані про події миші
- Одержати розширену інформацію про введені з клавіатури дані
- Записати дані до вхідного буфера
- Читати дані без видалення з вхідного буфера
- Визначити число подій у вхідному буфері
- Очистити вхідний буфер

Win32 API надає п'ять низькорівневих функцій для доступу до вхідного буфера:

| | |
|---|---|
| **ReadConsoleInput** | Читає і видаляє записи із вхідного буфера. Прочитані записи переміщаються в буфер викликаючого процесу. Непрочитані записи залишаються у вхідному буфері для наступної операції читання даних. Функція повідомляє загальне число записів, які були прочитані. |
| **PeekConsoleInput** | Читає вхідні записи без видалення із вхідного буфера. Якщо жоден запис недоступний, функція повертається негайно. Функція повідомляє загальне число записів, які були прочитані. |
| **GetNumberOfConsole-InputEvents** | Визначає число непрочитаних вхідних записів у вхідному буфері. |
| **WriteConsoleInput** | Розміщує записи у вхідному буфері позаду наявних записів. Вхідний буфер зростає динамічно, якщо необхідно утримувати додаткові вхідні записи. |
| **FlushConsoleInputBuffer** | Відкидає всі непрочитані події у вхідному буфері (очищає буфер). |

***Низькорівневі функції виведення*** дають можливість читати або записувати певну послідовність комірок символів у буфер екрану, зокрема:

- Читати і записувати рядки символів у вказану позицію буферу екрану
- Читати і записувати атрибути кольору у вказану позицію буфера екрану
- Читати і записувати прямокутні блоки символів і атрибути кольору у вказану позицію буфера екрану.
- Записати один символ або атрибути кольору у задане число послідовних комірок, що починаються у вказаній позиції буфера екрану

| | |
|---|---|
| **ReadConsoleOutputCharacter** | Копіює рядок символів із буфера екрану |
| **WriteConsoleOutputCharacter** | Записує рядок символів до буфера екрану |
| **ReadConsoleOutputAttribute** | Копіює атрибути кольору з буфера екрану |
| **WriteConsoleOutputAttribute** | Записує атрибути кольору до буферу екрану |
| **FillConsoleOutputCharacter** | Записує єдиний символ до вказаної кількості послідовних комірок в буфері екрану |
| **FillConsoleOutputAttribute** | Записує атрибути кольору до вказаної кількості послідовних комірок в буфері екрану |

Розглянемо приклади низькорівневих функцій:

```
BOOL ReadConsoleInput(

    HANDLE hConsoleInput,                // handle of a console input buffer
    PINPUT_RECORD lpBuffer,              // address of the buffer for read data
    DWORD nLength,                       // number of records to read
    LPDWORD lpNumberOfEventsRead         // address of number of records read
   );
```

Структура запису вхідного буферу (подія миші):

| **EventType** | **Event** | | | | | |
|---|---|---|---|---|---|---|
| MOUSE EVENT | MOUSE_EVENT_RECORD | | | | | |
| MOUSE_-EVENT | dwMouse-Position | | dwButtonState | dwControlKeyState | | dwEventFlags |
| | wX | wY | | | | |
| 0 | 4 | 6 | 8 | 12 | | 16 |
| KEY EVENT | KEY_EVENT_RECORD | | | | | |
| KEY_EVENT | bKeyDown | wRepeat Count | wVirtual-KeyCode | wVirtual-ScanCode | uChar | dwControl-KeyState |
| 0 | 4 | 8 | 10 | 12 | 14 | 16 |

| **dwButtonState:** | **dwControlKeyState:** |
|---|---|
| FROM_LEFT_1ST_BUTTON_PRESSED<br>RIGHTMOST_BUTTON_PRESSED<br>FROM_LEFT_2ND_BUTTON_PRESSED<br>FROM_LEFT_3RD_BUTTON_PRESSED<br>FROM_LEFT_4TH_BUTTON_PRESSED | RIGHT_ALT_PRESSED<br>LEFT_ALT_PRESSED<br>RIGHT_CTRL_PRESSED<br>LEFT_CTRL_PRESSED<br>SHIFT_PRESSED<br>NUMLOCK_ON<br>SCROLLLOCK_ON<br>CAPSLOCK_ON<br>ENHANCED_KEY |
| **dwEventFlags:**<br>DOUBLE_CLICK<br>MOUSE_MOVED<br>MOUSE_WHEELED | |

Наступні функції *читають або записують до прямокутних блоків* символьних комірок у вказаному розташуванні в буфері екрану.

| | |
|---|---|
| **ReadConsoleOutput** | Копіює дані символу і кольору із вказаного блоку комірок буфера екрану в заданий блок у буфері призначення |
| **WriteConsoleOutput** | Записує дані символу і кольору до вказаного блоку комірок буфера екрану із заданого блоку в початковому буфері |

Ці функції розглядають буфер екрану і буфери джерела або одержувача як двомірні масиви структур типу CHAR_INFO (містить дані символу і кольору для кожної комірки).

```
typedef struct _CHAR_INFO {
    union {                              // Unicode or ANSI character
        WCHAR UnicodeChar;
        CHAR AsciiChar;
    } Char;
    WORD Attributes;                     // text and background colors
} CHAR_INFO, *PCHAR_INFO;
```

Атрибути символу – NULL або будь-яка комбінація значень:

| | |
|---|---|
| **BACKGROUND_BLUE** | **FOREGROUND_BLUE** |
| **BACKGROUND_GREEN** | **FOREGROUND_GREEN** |
| **BACKGROUND_RED** | **FOREGROUND_RED** |
| **BACKGROUND_INTENSITY** | **FOREGROUND_INTENSITY** |

Високорівневі і низькорівневі методи I/O не взаємовиключні, і програма може використовувати будь-яку комбінацію цих функцій. Проте, зазвичай прикладні програми використовують або один підхід або інший.