Лабораторна робота № 1
Технології розробки Windows-програм з використання Windows API

Мета роботи: вивчити особливості програмування для операційної системи Windows і отримати навички розробки Windows-програм у системі Microsoft Visual Studio із використанням Windows API.

Теоретичні відомості

Особливості Windows-додатків

Консольні програми взаємодіють з користувачами за допомогою введення даних з клавіатури та виведення результатів на екран, що працює в символьному режимі.

Windows-додатки мають структуру, що відрізняється від звичайних консольних додатків. Це пов’язано з тим, що Windows-додаток може працювати із засобами введення та виведення комп'ютера тільки через функції, надані операційною системою Windows; ніякого прямого доступу до апаратних ресурсів не допускається. Оскільки в Windows можуть бути активними одночасно кілька програм, Windows визначає, яка з них повинна отримати введення – клацання миші або натискання клавіші – і відповідно сповіщає потрібну програму. Тобто загальне управління усією взаємодією з користувачем здійснюється операційною системою Windows.

До того ж природа інтерфейсу між користувачем та Windows-додатком така, що будь-якої миті часу можливе найрізноманітніший вплив на програму.

Користувач може вибрати будь-який з пунктів меню, клацнути на кнопці панелі інструментів або клацнути кнопкою миші в будь-якому місці вікна програми. Добре спроектований Windows-додаток повинен бути готовим обробляти введення будь-якого роду в будь-який момент часу, оскільки заздалегідь невідомо, яка саме дія відбудеться.

Всі ці дії користувача приймаються насамперед операційною системою та розглядаються нею як події (events). Подія, сприйнята графічним користувацьким інтерфейсом (Graphical User Interface - GUI) Windows-додатку, зазвичай призводить до виконання певного фрагмента коду програми. Таким чином, порядок виконання окремих частин програми визначається послідовністю дій користувача, а, зрештою, – тими подіями, які пов'язані з цими діями.

Програми, що працюють таким чином, називаються програмами, керованими подіями (event-driven programs), і відрізняються від традиційних

процедурних програм, які реалізують безперервну послідовність виконання операторів.

Windows-додаток складається, перш за все, з фрагментів коду, що реагують на події, викликані діями користувача або самої операційної системи Windows. Структура таких програм показана на рис.1.

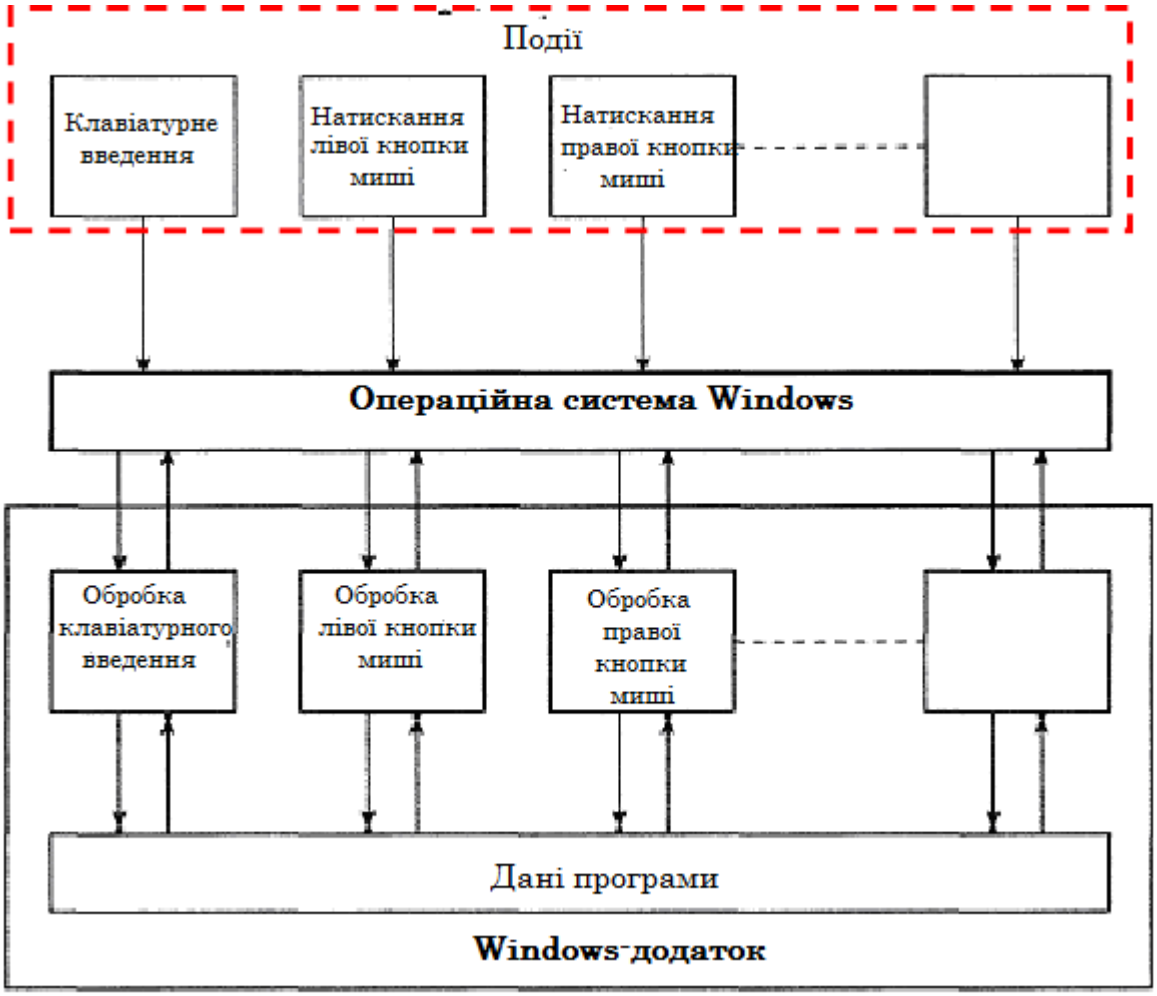


Рисунок 1. Структура обробки в Windows-додатку

На рис.1 кожен блок Windows-програми позначає фрагмент коду, написаний спеціально для опрацювання певної події. Програма може виглядати сильно фрагментованою через безліч не пов'язаних між собою блоків коду, але основний елемент, що зв'язує її в єдине ціле – це операційна система Windows. Можна, вважати таку програму індивідуальною оболонкою операційної системи Windows для виконання певного набору процесів.

Всі блоки, які обслуговують різні зовнішні події, як вибір пункту меню або клацання миші, мають доступ до загального набору даних, специфічних для конкретної програми. Ці дані програми містять інформацію, що стосується того, для чого програма призначена (наприклад, рядки тексту в

текстовому редакторі), а також інформацію про деякі події, що відбулися під час виконання програми.

Ця загальна колекція даних дозволяє різним частинам програми, які виглядають незалежними, взаємодіяти узгодженим чином.

Звичайно, можна розробляти програми, які не вимагають операційної системи Windows, і ігрові програми часто використовують такий підхід, коли потрібно досягти максимальної продуктивності графіки.

Розробка Windows-програми з використанням Windows API

Раніше була відзначена характерна особливість Windows-програми, яка полягає в тому, що така програма, по суті, постійно чекає, коли станеться якась подія. Значна частина коду, необхідного програмі Windows, призначена для обробки подій, викликаних зовнішніми діями користувача, але дії, не асоційовані безпосередньо з програмою Windows, можуть також вимагати виконання деякого фрагмента коду в програмі Windows. Наприклад, якщо користувач перетягує вікно іншої програми, яка працює паралельно з програмою Windows, і це дія відкриває частину області вікна Windows-програми, додаток повинен перемалювати частину свого вікна.

Повідомлення Windows

Події в Windows-додатку є обставинами, подібними до клацання кнопкою миші або натискання клавіші або закінчення певного періоду часу. Операційна система Windows записує кожну подію в повідомлення та поміщає його в чергу повідомлень програми, якій це повідомлення призначене. Таким чином, повідомлення Windows – це просто запис даних, що стосуються події, а черга повідомлень програми – це просто послідовність таких повідомлень, що очікують обробки додатком. Відправляючи повідомлення, Windows може повідомити програму про те, що щось має бути зроблено, або про те, що деяка інформація стала доступною, або про те, що сталося подія на кшталт клацання кнопкою миші. Якщо програму правильно організовано, вона відповідним чином реагує на повідомлення. Може існувати безліч видів повідомлень, і вони можуть надходити дуже часто - багато разів на секунду, - наприклад, коли пересувається миша.

Програма Windows повинна містити функцію, спеціально призначену для обробки таких повідомлень. Ця функція часто називається WndProc() або WindowProc(), хоча вона не повинна мати якесь певне ім'я, тому що Windows звертається до функції через покажчик, що надається програмою Windows.

Тому надсилання повідомлення Windows додатку зводиться до виклику операційною системою Windows наданої функції, зазвичай з іменем WindowProc(), і до передачі необхідних даних Windows-додатку через аргументи цієї функції. Обробка повідомлення, переданого з даними цієї функції, повністю задається програмістом.

Програміст не повинен писати код для обробки кожного повідомлення. Можна обробляти тільки ті з них, які цікавлять програму, передаючи інші назад операційної системи Windows. Повідомлення передається назад Windows за допомогою стандартної Windows-функції на ім'я DefWindowProc(), яка забезпечує їхню обробку за замовчуванням.

Прикладний програмний інтерфейс Windows (Windows API)

Взаємодія між будь-яким Windows-додатком і системою Windows використовує прикладний програмний інтерфейс Windows, відомий також під назвою Windows API. Він складається з багатьох сотень стандартних функцій, за допомогою яких програма взаємодіє з операційною системою Windows і навпаки. Windows API був розроблений у ті часи, коли основною мовою був С, і з цієї причини в ньому часто використовуються структури замість класів для передачі деякого роду даних між Windows і додатком.

Windows API покриває всі аспекти взаємодії між Windows та програмою.

Структура Windows-програми

Для найпростішої Windows-програми, яка використовує Windows API, слід написати дві функції:

1) функція WinMain(), з якої починається виконання програми та відбувається її основна ініціалізація;

2) функція WindowProc(), що викликається операційною системою Windows для обробки повідомлень програми. Частина Windows-програми, реалізована функцією WindowProc(), зазвичай виявляється більшою її частиною, оскільки є тим місцем, в якому знаходиться специфічний для програми код, що відповідає на повідомлення, ініційовані різними діями користувача.

Хоча ці дві функції утворюють повну програму, вони непов'язані між собою безпосередньо. WinMain() не викликає WindowProc(), оскільки це робить операційна система Windows, яка викликає і WinMain()

Функція WinMain ( )

Функція WinMain() – це еквівалент функції main() консольної програми. Саме тут починається виконання та відбувається базова

ініціалізація решти програми. Щоб дозволити операційній системі Windows передати дані функції, WinMain() приймає чотири параметри та повертає значення типу int. Її прототип виглядає так:

```
int WINAPI WinMain(HINSTANCE hInstance,
                   HINSTANCE hPrevInstance,
          LPSTR lpCmdLine,
          int nCmdShow);
```

За специфікатором типу повернення int знаходиться специфікація WINAPI, яка змушує обробляти ім'я функції та її аргументи спеціальним чином, що відповідає обробці функцій мовами Pascal і Fortran. Це відрізняється від звичайного способу обробки функцій C++.

Перший аргумент, hInstance, має тип HINSTANCE, що представляє дескриптор екземпляра. Дескриптор (handle) – це ціле значення, що ідентифікує об'єкт певного роду, в даному випадку – екземпляр програми (додатки). Справжнє значення дескриптора не важливе. У будь-який час під керуванням Windows може виконуватись кілька програм. Це означає, що кілька копій однієї програми можуть бути активними одночасно, і їх необхідно якось розрізняти. Тому дескриптор hInstance ідентифікує конкретну копію. Якщо запущено більше однієї копії програми, кожна з них отримує своє власне значення hInstance. Дескриптори також застосовуються для ідентифікації багатьох інших речей. Звичайно, всі дескриптори, що знаходяться в певному контексті, наприклад, дескриптори екземплярів програми, повинні відрізнятися один від одного.

Функція WinMain() повинна виконувати чотири дії, які наведені нижче.

1. Повідомляти операційну систему Windows, якого виду (класу) вікно потрібно Windows-програмі, тобто. встановити специфікацію вікна програми.
2. Створювати вікно програми.
3. Ініціалізувати вікно програми.
4. Виймати повідомлення операційної системи Windows, призначені Windows-програмі, з використанням так званого циклу обробки повідомлень

Специфікація вікна програми

Перший крок у створенні вікна передбачає визначення класу вікна, яке потрібно створити. У Windows визначено спеціальний тип структури на ім'я

WNDCLASSEX, яка містить дані, що описують вікно. Дані, що зберігаються в екземплярі структури, описують клас вікна, що визначає його тип. Необхідно створити змінну типу WNDCLASSEX і надати значення кожному з її членів. Після того, як усі члени отримають значення, цю структуру можна передати Windows через спеціальну функцію, щоб зареєструвати клас вікна. Коли це зроблено, то щоразу, коли потрібно створити вікно цього класу, можна вказувати Windows, щоб вона шукала класу, який вже був зареєстрований.

Визначення структури WNDCLASSEX виглядає так:

```
struct WNDCLASSEX
{
UINT cbSize; // Розмір цього об'єкта в байтах
UINT style; // Стиль вікна
WNDPROC lpfnWndProc; // Покажчик на функцію обробки повідомлень
int cbClsExtra; // Додатковий байт після класу вікна
int cbWndExtra; // Додаткові байти після екземпляра вікна
HINSTANCE hInstance; // Дескриптор екземпляра програми
HICON hIcon; // Піктограма програми
HCURSOR hCursor; // Курсор вікна
HBRUSH hbrBackground; // Пензлик, що визначає колір тла
LPCTSTR lpszMenuName; // Покажчик на ім'я ресурсу меню
LPCTSTR lpszClassName; // Покажчик на ім'я класу вікна
HICON hIconSm; // Мала піктограма, пов'язана з вікном
};
```

Змінна типу WNDCLASSEX створюється звичайним способом, наприклад:

```
WNDCLASSEX WindowClass;
```

Створення вікна програми

Після того, як усім членам структури WNDCLASSEX надано потрібні значення, наступний крок полягає в тому, щоб повідомити про це Windows за допомогою функції Windows API RegisterClassEx(). Якщо структура називається WindowClass, то оператор, за допомогою якого це робиться, має бути таким:

```
RegisterClassEx(&WindowClass);
```

У такий спосіб адреса структури передається в функцію, а операційна система Windows витягує та зберігає всі значення, встановлені в членах структури. Цей процес називається реєстрацією класу вікна. Тут термін

"клас" використовується в сенсі класифікації вікна, і ніяк не пов'язаний із класом C++, так що не слід плутати їх. Кожен екземпляр програми повинен забезпечити реєстрацію свого класу вікна.

Після того, як Windows дізнається про характеристики потрібного вікна і функцію, призначену для обробки його повідомлень, можна створити вікно. Для цього використовується функція CreateWindow(). Певний клас вікна описує широкий діапазон параметрів вікна, а додаткові аргументи функції CreateWindow() додають до них ще деякі.

Оскільки програма в загальному випадку може складатися з декількох вікон, функція CreateWindow() повертає дескриптор створеного вікна, який можна зберегти, щоб пізніше звертатися до конкретного вікна.

Типове використання функції CreateWindow():

```
HWND hWnd; // Дескриптор вікна

hWnd = CreateWindow(
szAppName, // Ім'я класу вікна
L "Вікно Windows-програми",// Заголовок вікна
WS_OVERLAPPEDWINDOW, // Стиль вікна - перекривається
CW_USEDEFAULT, // Позиція на екрані за замовчуванням
CW_USEDEFAULT, // лівого верхнього кута як x, y...
CW_USEDEFAULT, // Стандартний розмір вікна - ширина...
CW_USEDEFAULT, // ...і висота
0, // Немає батьківського вікна
0, // Немає меню
hInstance, // Дескриптор екземпляра програми
0        // Жодних даних для створення вікна
);
```

Змінна hWnd типу HWND – це 32-розрядний цілий дескриптор вікна. Ця змінна використовується для запису значення, що повертається функцією CreateWindow() та ідентифікує вікно.

Перший аргумент, який передається у функцію CreateWindow(), це ім'я класу вікна. Воно використовується Windows для ідентифікації структури WNDCLASSEX, яка була передана до цього операційній системі при виклику функції RegisterClassEx(), так що інформація з цієї структури може бути використана у процесі створення вікна.

Другий аргумент визначає текст, який має з'явитись у заголовку вікна.

Третій аргумент специфікує стиль, що має вікно після створення. Вказана тут опція, WS_OVERLAPPEDWINDOW, насправді комбінує кілька опцій. Вона визначає вікно як стилі WS_OVERLAPPED, WS_CAPTION, WS_SYSMENU, WS_THICKFRAME, WS_MINIMIZEBOX та

WS_MAXIMIZEBOX. Це дає в результаті вікно, що перекривається, призначене для того, щоб служити головним вікном програми, із заголовком на товстій рамці, що має піктограму системного меню, а також кнопки розгортання та згортання. Специфіковане вікно має товсту лінію рамки, за допомогою якої можна змінювати розмір вікна.

Наступні чотири аргументи визначають положення та розмір вікна на екрані.

Перші два – це екранні координати лівого верхнього кута вікна, а другі – ширина і висота вікна. Значення CW_USEDEFAULT повідомляє Windows, що операційна система повинна задати положення і розмір вікна за замовчуванням, а також розташовувати вікна каскадом - послідовно один за одним. Значення CW_USEDEFAULT застосовується лише до вікна, специфіковані як WS_OVERLAPPED.

Наступне значення аргументу – нуль, що говорить про те, що вікно не є дочірнім (тобто вікном, що залежить від батьківського вікна). Якщо потрібно створити дочірнє вікно, то для цього аргументу необхідно вказати дескриптор батьківського вікна. Наступний аргумент також встановлений у нуль, і це свідчить, що жодного меню не потрібно. Потім задається дескриптор поточного екземпляра програми, який був переданий цій програмі від Windows через параметр hInstance функції WinMain(). Останній аргумент під час створення вікна дорівнює нулю, оскільки у цьому прикладі потрібно створити просте вікно.

Після виклику функції CreateWindow() вікно існує, але ще не відображено на екрані. Щоб відобразити його, потрібно викликати іншу функцію Windows API:

```
ShowWindow(hWnd, nCmdShow); // Відобразити вікно
```

Тут потрібні лише два аргументи. Перший ідентифікує вікно та дорівнює дескриптору, поверненому функцією CreateWindow(). Другий має значення, передане від Windows через параметр nCmdShow функції WinMain(), і ідентифікує спосіб відображення вікна на екрані.

Ініціалізація вікна програми

Після виклику функції ShowWindow() вікно з'являється на екрані, але все ще немає вмісту, тому потрібно змусити програму намалювати клієнтську область вікна.

Можна просто зібрати весь необхідний для цього код безпосередньо у функції WinMain(), але цього буде недостатньо, оскільки вміст клієнтської області вікна залишиться статичним і після виведення у вікно необхідної

інформації доведеться піклуватися про її відтворення в початковому вигляді. Будь-яка дія користувача, яка модифікує вікно будь-яким чином (наприклад, перетягуванням границі або вікна повністю), зазвичай вимагає перемальовки вікна та його клієнтської області.

Коли клієнтська область має бути перемальована з будь-якої причини, Windows надсилає певне повідомлення програмі, на яке функція WindowProc() має відреагувати, реконструюючи клієнтську область вікна. Таким чином, найкращий спосіб забезпечити перемальовування клієнтської області у першій інстанції – це помістити код перемалювання клієнтської області у функцію WindowProc() і дозволити Windows відправляти програмі необхідні запити на перемалювання клієнтської області вікна. Щоразу, коли програма вирішує, що вікно має бути перемальовано (наприклад, коли щось змінилося у вікні), то вона повинна повідомити Windows про необхідність зворотного надсилання повідомлення про те, що вікно має бути перемальовано.

Програміст може попросити Windows надіслати його програмі повідомлення про перемальовування клієнтської області, викликаючи спеціальну функцію Windows API – UpdateWindow().

Необхідний для цього оператор виглядає таким чином:

```
UpdateWindow(hWnd); // Викликати перемальовування клієнтської області вікна
```

Ця функція має лише один аргумент, який представляє дескриптор вікна hWnd, який ідентифікує певне вікно програми.

Цикл обробки повідомлень

Windows поміщає в чергу повідомлення про такі дії користувача, як введення з клавіатури, переміщення миші, натискання кнопок миші, а також повідомлення від таймера та повідомлення Windows, які вимагають перемалювання вікна. Функція WinMain() повинна витягувати їх із черги повідомлень для подальшої обробки. Після отримання повідомлення, яке операційна система помістила в чергу Windows-програми, потрібно просити операційну систему викликати функцію WindowProc() цієї програми для того, щоб обробити вилучене повідомлення.

Виймання повідомлень із черги виконується у функції WinMain() із застосуванням стандартного механізму програмування Windows, який називається циклом обробки повідомлень. Необхідний для цього програмний код має виглядати так:

```
MSG msg; // Структура повідомлення Windows
while(GetMessage(&msg, 0, 0, 0)==TRUE) // Отримати повідомлення з черги
{
TranslateMessage(&msg); // Трансляція повідомлення
DispatchMessage(&msg); // Диспетчеризація повідомлення
}
```

Обробка повідомлення складається із трьох кроків.

GetMessage() – отримує повідомлення із черги.

TranslateMessage() – виконує всі необхідні перетворення отриманого повідомлення.

DispatchMessage() – змушує операційну систему Windows викликати для обробки повідомлення функцію WindowProc(), що входить до складу програми Windows.

Функція GetMessage() витягує повідомлення, розміщене в черзі вікна програми, і зберігає інформацію про повідомлення в змінній msg, яка є першим аргументом. Змінна msg являє собою структуру типу MSG, один з членів якої – член message – містить ідентифікатор повідомлення, що є цілим значенням, яке може бути одним з набору значень, визначених у заголовному файлі <windows.h> у вигляді символічних констант. Усі константи починаються з WM_, наприклад: WM_PAINT – для перемалювання екрана, WM_QUIT – завершення програми.

Функція GetMessage() завжди повертає TRUE, якщо не отримано повідомлення WM_QUIT, для якого функція повертає FALSE. Таким чином, цикл while триває доти, поки не з'явиться повідомлення WM_QUIT для закриття програми. Після виходу з циклу виконання функції WinMain() завершується оператором return, який передає назад у Windows значення, що міститься в члені wParam змінної msg. Завершення виконання функції WinMain() означає закінчення роботи Windows-програми.

Другий аргумент під час виклику GetMessage() – це дескриптор вікна, для якого потрібно отримати повідомлення. Цей параметр може бути використаний для отримання повідомлень для одного вікна, окремо від іншого. Якщо аргумент дорівнює 0, як тут, GetMessage() витягує всі повідомлення для програми. Це простий спосіб отримання всіх повідомлень для програми, незалежно від того, скільки вікон воно має. Крім того, це також найбільш безпечний спосіб, оскільки гарантує отримання всіх повідомлень для додатку.

Останні два аргументи GetMessage() – цілі числа, що містять мінімальне та максимальне значення ідентифікаторів повідомлень, які ви хочете вилучити з черги.

Це дозволяє вибирати повідомлення вибірково. Зазвичай, діапазони вказуються символічними константами. Використання WM_MOUSEFIRST і WM_MOUSELAST як цих двох аргументів, наприклад, дозволяє вибрати лише повідомлення миші. Якщо обидва аргументи дорівнюють нулю, як у цьому прикладі, то витягуються всі повідомлення.

Всередині циклу while виклик функції TranslateMessage() запитує операційну систему Windows виконати деяку роботу з перетворення повідомлень, що мають відношення до клавіатури. Потім виклик функції DispatchMessage() змушує Windows здійснити диспетчеризацію повідомлення, тобто викликати функцію WindowProc() у Windows-програмі для обробки повідомлення. Повернення з функції DispatchMessage() не відбувається доти, доки WindowProc() не закінчить обробку повідомлення. Отримання повідомлення WM_QUIT говорить про те, що програма має завершитися, тому при цьому функція GetMessage() повертає до програми FALSE, що призводить до виходу з циклу обробки повідомлень

Функція WindowProc ( )

Функція WinMain() не містить нічого специфічного для програми, крім загального зовнішнього вигляду вікна цієї програми. Весь код, який змушує програму поводитися так, як хоче програміст, включається до частини Windows-програми, зайнятої обробкою повідомлень. Цією частиною є функція WindowProc(), яка вказана для операційної системи Windows у структурі WindowClass, що заповнена у функції WinMain().

Операційна система Windows викликає функцію WindowProc() щоразу, коли здійснюється диспетчеризація повідомлення для головного вікна програми Windows.

В простій програмі весь код обробки повідомлень зазвичай міститься в одній функції – WindowProc(). У загальному випадку функція WindowProc() аналізує повідомлення, що надійшло, визначає, якому конкретному вікну воно адресовано, і потім викликає одну з набору функцій, кожна з яких призначена для обробки певного повідомлення в контексті конкретного вікна. Однак у тому, що стосується загальної послідовності операцій та способу, яким функція WindowProc() аналізує повідомлення, що надходить, то тут все однаково для більшості Windows-додатків.

Прототип функції WindowProc() виглядає так:

```
LRESULT CALLBACK WindowProc(HWND hWnd, UINT message,
```

```
    WPARAM wParam, LPARAM lParam);
```

Тип повернення LRESULT – це тип, визначений у Windows, і зазвичай еквівалентний long. Оскільки функція викликається операційною системою Windows через покажчик, який встановлено у функції WinMain(), коли заповнювалася структура WNDCLASSEX, її слід кваліфікувати як CALLBACK. Це ще один специфікатор, визначений у Windows, який служить для визначення способу обробки аргументів функції

Повідомлення ідентифікуються значенням message, переданим функції WindowProc(). Можна порівняти це значення з символічними константами, кожна з яких відноситься до певного типу повідомлення. Типовим прикладом є константа WM_PAINT, що відповідає запиту на перемальовування частини клієнтської області вікна, та WM_LBUTTONDOWN, що вказує, що натиснута ліва кнопка миші.

Декодування повідомлення Windows

Процес декодування повідомлення, яке надсилає Windows, зазвичай реалізується оператором switch функції WindowProc() на основі значення message. Вибір типів повідомлень, які передбачається обробляти, зводиться до розміщення оператора case для кожного з них усередині switch. Типовий вигляд такого оператора switch з включенням вибраних міток case виглядає так:

```
switch (message)
{
case WM_PAINT:
// Код малювання клієнтської області
break;
case WM_LBUTTONDOWN:
// Код обробки натискання лівої кнопки миші
break;
case WM_LBUTTONUP:
// Код обробки відпускання лівої кнопки миші
break;
 case WM_DESTROY:
// Код обробки знищення вікна
break;
default:
// Код обробки всіх інших повідомлень
}
```

Кожна конструкція case відповідає певному значенню ідентифікатора повідомлення та здійснює відповідну обробку повідомлення. Будь-яке повідомлення, яке програма не збирається обробляти індивідуально, обробляється в операторі default, який повинен надіслати повідомлення Windows викликом DefWindowProc(). Це функція Windows API, що забезпечує обробку стандартного повідомлення (Default).

Малювання в клієнтській області вікна

Windows надсилає повідомлення WM_PAINT у програму, щоб просигналізувати, що клієнтська область програми має бути перемальована. Тому в прикладі необхідно перемалювати текст вікна у відповідь на повідомлення WM_PAINT.

Перш ніж програма зможе малювати у вікні програми, необхідно повідомити Windows про цей намір і отримати від Windows дозвіл на це викликом функції Windows API BeginPaint(), яка повинна бути викликана лише у відповідь WM_PAINT. Вона використовується так:

```
HDC HDC; // Дескриптор дисплейного контексту
PAINTSTRUCT PaintSt;// Структура, що визначає область для перемальовки
hDC = BeginPaint(hWnd,&PaintSt); // Підготуватися до малювання у вікні
```

Тип HDC визначає те, що називається дисплейним контекстом, або ширшому сенсі – контекстом пристрою. Контекст пристрою представляє зв'язок між незалежними від пристроїв функціями Windows API виводу інформації на екран або принтер і драйверами пристроїв, що підтримують виведення на конкретні пристрої, визначені до даного комп’ютера. Контекст пристрою можна трактувати як маркер авторизації, який Windows видає програмі на запит і дозволяє виводити деяку інформацію. Без контексту пристрою програма не може генерувати будь-яке виведення.

Функція BeginPrint() повертає дисплейний контекст і має два аргументи. Вікно, де програма збирається малювати, ідентифікується його дескриптором hWnd, який передається першим аргументом. Другий аргумент – адреса змінної PaintSt типу PAINTSTRUCT, куди Windows розміщає інформацію про область, яку необхідно перемалювати у відповідь на повідомлення WM_PAINT. Для простоти прикладу програма перемальовуватиме всю клієнтську область. Координати клієнтської області можна записати до структури RECT так:

```
RECT aRect; // Робочий прямокутник
GetClientRect(hWnd,&aRect);
```

Функція GetClientRect() запитує координати лівого верхнього та правого нижнього кутів клієнтської області для вікна, специфікованого першим аргументом hWnd.

Ці координати зберігаються у структурі aRect типу RECT, яка передається у другому аргументі як покажчика. Потім можна використовувати отриману інформацію про розташування клієнтської області вікна, щоб вивести у вікно необхідний текст функцією DrawText(). Оскільки вікно має сірий фон, потрібно зробити фон прозорим для тексту; інакше, текст з'явиться на білому тлі. Це можна зробити наступним викликом функції Windows API:

```
SetBkMode(hDC, TRANSPARENT); // Встановити режим тла тексту
```

Перший аргумент ідентифікує контекст пристрою, а другий режим відображення фону (текст на прозорому фоні). За замовчанням прийнято режим повної прозорості OPAQUE.

Після цього можна виводити текст за допомогою наступного оператора:

```
DrawText(hDC, // Дескриптор контексту пристрою
L"Текст виводиться при перемалюванні вікна",
-1, // Індикатор рядка, обмеженого null
&aRect,// Прямокутник, у якому виконується малювання тексту
DT_SINGLELINE | // Формат тексту – один рядок
DT_CENTER | // - центрування у рядку
DT_VCENTER // - центрування за висотою aRect
);
```

Перший аргумент функції DrawText() позначає отриманий дозвіл на малювання у вікні; це дисплейний контекст hdc. Наступний аргумент – текстовий рядок, який слід вивести. З тим самим успіхом ви могли б визначити її в змінній і передати вказівник на текст як другий аргумент при виклику функції. Наступний аргумент, що має -1, вказує на те, що текстовий рядок обмежений символом null. Інакше тут слід було б вказати кількість символів рядка. Четвертий аргумент – покажчик на структуру RECT, що визначає прямокутник, у якому програма виводитиме текст. У цьому випадку це повна клієнтська область вікна, визначена в aRect.

Останній аргумент визначає формат тексту прямокутнику. Тут скомбіновані специфікації, об'єднані бітовим АБО (|). Рядок виводиться в одну лінію з центруванням тексту по горизонталі та по вертикалі всередині прямокутника. Це красиво розподіляє рядок по центру вікна. Існує також

безліч інших опцій, які включають можливість розміщення тексту у верхній або нижній частині прямокутника з вирівнюванням ліворуч або праворуч.

Після того, як програма вивела все, що потрібно відобразити, слід повідомити Windows, що малювання в клієнтській області завершено. Для кожного виклику BeginPaint() повинен існувати відповідний виклик EndPaint(). Тому для завершення обробки повідомлення WM_PAINT потрібен такий оператор:

```
EndPaint(hWnd, &PaintSt); // Завершити операцію перемалювання вікна
```

Аргумент hWnd ідентифікує вікно програми, а другий аргумент – адресу структури PAINTSTRUCT, яку було заповнено функцією BeginPaint().

Задача. У центрі робочої області вікна вивести рядок " Вивід при обробці WM_PAINT ". Після натискання лівої клавіші миші вміст цього рядка змінити на текст "Натиснута ліва клавіша миші".

Частина коду (функція WndProc)

```
LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM
lParam)
{
 static short cx, cy;
 switch (msg) {
   case WM_SIZE:
   { cx=LOWORD(lParam);  cy=HIWORD(lParam);
     return 0;
   }
   case WM_PAINT:
    { char szText[ ]="Вивід при обробці WM_PAINT";
      PAINTSTRUCT ps;
      HDC hdc=BeginPaint(hwnd,&ps);
       //Колір виведення символів малиновий
      SetTextColor(hdc,RGB(255,0,255));
       //Колір фону виводу символів кремовий
      SetBkColor(hdc,RGB(255,251,240));
      SetTextAlign(hdc,TA_CENTER);
      TextOut(hdc,cx/2,cy/2,szText,strlen(szText));
      EndPaint(hwnd,&ps);
      return 0;
     }
   case WM_LBUTTONDOWN:
    { char szText[]="Обробка WM_LBUTTONDOWN";
      HDC hdc=GetDC(hwnd);
      // Колір виведення символів синій
      SetTextColor(hdc,RGB(0, 0,255));
      // Колір фону виводу символів жовтий
      SetBkColor(hdc,RGB(255,255,0));
```

```
      SetTextAlign(hdc,TA_CENTER);
      TextOut(hdc,cx/2,cy/2-60,szText,strlen(szText));
      ReleaseDC(hwnd, hdc);
     return 0;
     }
  case WM_DESTROY: {
    PostQuitMessage(0);
    return 0;
    }
  }
  return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

Завершення роботи програми

Закриття вікна зазвичай завершує роботу програми, але для того, щоб отримати таку поведінку, необхідно додати ще трохи коду. Причина того, що програма за замовчуванням не завершується із закриттям вікна, полягає в тому, що програмі може знадобитися виконати деякі заключні операції. Крім того, може статися, що програма має більше одного вікна. Коли користувач закриває вікно подвійним клацанням на піктограмі в заголовку або клацанням на кнопці Close (Закрити), то в черзі зв'язаних з цією повідомлень останнім генерується повідомлення WM_DESTROY. Тому, щоб закрити програму, програма повинна обробити повідомлення WM_DESTROY в функції WindowProc(). Обробка цього повідомлення полягає у генерації повідомлення WM_QUIT наступним оператором:

```
PostQuitMessage(0);
```

Тут аргумент функції означає код повернення з програми. Ця функція Windows API робить саме те, про що говорить її ім'я – надсилає повідомлення WM_QUIT у чергу повідомлень Windows-програми. У цьому випадку викликана WinMain() функція GetMessage(), отримавши повідомлення WM_QUIT з черги, повертає FALSE, в результаті чого відбувається вихід з циклу обробки повідомлень, і потім завершується програма.

ЛАБОРАТОРНЕ ЗАВДАННЯ

1. Вивчити теоретичні відомості, наведені в лабораторній роботі.
2. Письмово відповісти на 2 контрольні питання.
    a. Номер першого питання визначається за номером за списком.
    b. Номер другого питання визначається за формулою
    (N+15) mod 30
    де N – номер за списком
3. Створити та виконати приклад програми Windows, що використовує функції виклику Win API (створення проекту описується в Додатку 1). Назва вікна повинна співпадати з прізвищем студента, в клієнтській області потрібно вивести текст такого зразка

"Привіт, мене створив < Ім'я, по-батькові >

4. Модифікувати Windows-програму відповідно до варіанта завдання та виконати модифіковану Windows-програму.
5. Записати у звіт інформацію про структуру проекту, що відображається у вікні браузера рішення, оформити звіт, в якому продемонструвати результати виконання лабораторної роботи.

Варіанти завдань

Інформація про символічні константи мови Visual C++, що відповідають ідентифікаторам повідомлень, наведена у табл.2. Для виведення інформації про подію можна використовувати функцію MessageBox, наприклад:

```
MessageBox(0,L"Натиснута ліва кнопка",L"Миша", MB_OK)
```

Значення констант для повідомленні наведені в додатку 1

№ варіанта	Зміст завдання
1.	Змінити програмний код програми, передбачивши обробку повідомлення про відпускання лівої кнопки миші та виведення інформації про цю подію
2.	Змінити програмний код програми, щоб клієнтська область вікна фарбувалася в білий колір
3.	Змінити програмний код програми, щоб вміст та розташування напису в клієнтській області вікна відрізнялися від вказаних у прикладі

4.	Змінити програмний код таким чином, щоб при натисканні клавіші F1 з'являлось повідомлення про автора програми
5.	Змінити програмний код програми, передбачивши обробку повідомлення про натискання клавіші та виведення інформації про цю подію
6.	Змінити програмний код програми, передбачивши обробку повідомлення про натискання лівої кнопки миші та виведення інформації про цю подію
7.	Змінити програмний код програми, передбачивши обробку повідомлення про відпускання правої кнопки миші та виведення інформації про цю подію
8.	Змінити програмний код програми, щоб клієнтська область вікна фарбувалася у світлосірий колір
9.	Змінити програмний код програми, передбачивши обробку повідомлення про відпускання клавіші та виведення інформації про цю подію
10.	Змінити програмний код програми, передбачивши обробку повідомлення про натискання правої кнопки миші та виведення інформації про цю подію
11.	Змінити програмний код програми, щоб у заголовок вікна містив інформацію про номер групи та прізвище студента
12.	Змінити програмний код програми, передбачивши обробку повідомлення про натискання клавіші F1 та виведення інформації про цю подію.
13.	Змінити програмний код програми, передбачивши обробку повідомлення про переміщенні колеса миші та виведення інформації про цю подію
14.	Змінити програмний код програми, передбачивши обробку повідомлення про подвійне клацання лівої кнопки миші та виведення інформації про цю подію
15.	Змінити програмний код програми, щоб в клієнтській області вікна в лівому верхньому куті з'являлось повідомлення про назву дисципліни
16.	Змінити програмний код програми, щоб вміст в клієнтській області змінював положення при натисканні клавіші
17.	Змінити програмний код таким чином, щоб при натисканні правої кнопки миші вікно клієнтська частина перефарбовувалась в сірий колір
18.	Змінити програмний код програми, передбачивши обробку натискання лівої кнопки миші та зміни тексту в клієнтській частині вікна
19.	Змінити програмний код програми, передбачивши обробку відпускання лівої кнопки миші та зміни положення тексту в клієнтській частині вікна при цій події

20.	Змінити програмний код програми, передбачивши обробку повідомлення про зміну положення вікна та виведення інформації про цю подію
21.	Змінити програмний код програми, передбачивши обробку зміни розмірів вікна та виведення повідомлення про цю подію
22.	Змінити програмний код програми, передбачивши зміну тексту у клієнтській частині при натисканні клавіші
23.	Змінити програмний код програми, передбачивши обробку повідомлення про натискання правої кнопки миші та виведення у верхній частині вікна інформації про лабораторну роботу
24.	Змінити програмний код програми, щоб у клієнтській частині змінювалось положення тексту при прокручування колеса миші
25.	Змінити програмний код програми, передбачивши обробку повідомлення про натискання клавіші F1 та виведення інформації лабораторну роботу
26.	Змінити програмний код програми, передбачивши обробку повідомлення про відпускання лівої кнопки миші та виведення інформації про назву навчального закладу
27.	Змінити програмний код програми, щоб при відпусканні правої кнопки миші змінювалось положення тексту в клієнтській частині вікна
28.	Змінити програмний код програми, щоб при подвійному клацанні миші у клієнтській частині вікна додавалась інформація про лабораторну роботу
29.	Змінити програмний код програми, передбачивши зміну тексту в клієнтській частині при відпусканні клавіші
30.	Змінити програмний код програми, передбачивши обробку повідомлення про натискання правої кнопки миші та виведення інформації про лабораторну роботу

Контрольні питання

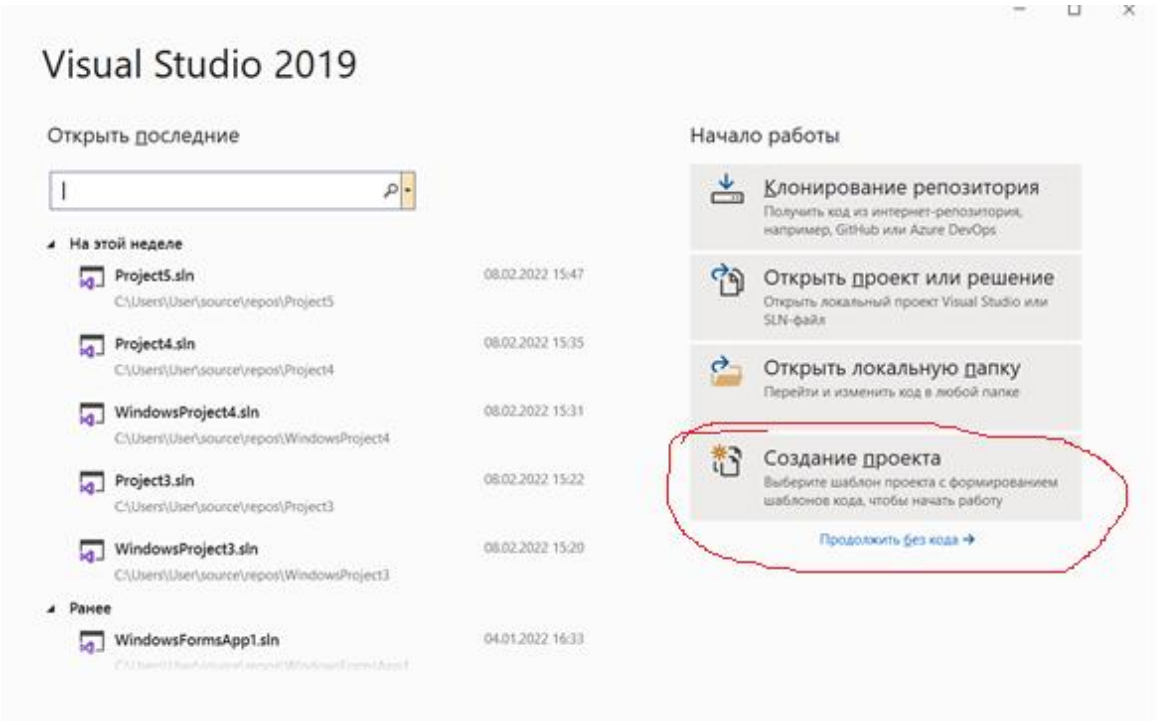
1. Вкажіть особливості консольної програми та Windows-програми, що стосуються роботи з засобами введення та виведення комп'ютера.
2. Яка програма називається програмою, що керується подіями?
3. Яку структуру має програма, керована подіями?
4. Які основні варіанти розробки програм можливі в системі Microsoft Visual C++?
5. Які основні технології створення інтерактивних Windows-програм доступні в системі Microsoft Visual C++? Порівняйте їх за трудомісткістю програмування.

6. Що являють собою повідомлення операційної системи Windows і черга повідомлень програми?
7. Для чого операційна система Windows надсилає повідомлення програмі?
8. Яка функція обов'язково повинна входить до складу Windows-програми і чи існує обмеження на вибір її імені?
9. Як має поводитися Windows-додаток з тими отриманими повідомленнями, обробка яких у додатку не передбачена?
10. Для чого використовується прикладний програмний інтерфейс Windows і з чого він складається?
11. Який спосіб запису імен використовується у Windows-програмах та з якою метою?
12. Яку структуру має Windows-програма, навіщо призначені основні частини цієї програми та як організовано взаємодію її частин між собою?
13. Який вигляд має прототип функції WinMain() і що означають аргументи функції?
14. Які дії має виконувати функція WinMain()?
15. Як встановити специфікацію вікна програми?
16. Як задається зв'язок вікна програми з функцією обробки повідомлень?
17. Як зареєструвати клас вікна програми?
18. Як створити вікно програми та відобразити його на екрані?
19. Де знаходиться клієнтська область вікна і як забезпечується виведення інформації в цю область під час ініціалізації вікна програми?
20. Яким програмним кодом реалізується цикл обробки повідомлень у функції WinMain()?
21. Як діє функція API GetMessage(), викликана в циклі обробки повідомлень?
22. Як виглядає прототип функції обробки повідомлень і що означають аргументи цієї функції?
23. Як у функції обробки повідомлень програмується декодування повідомлення, яке надсилає Windows?
24. Чому для завершення програми Windows необхідно у функції обробки повідомлень викликати API-функцію PostQuitMessage()?
25. Навіщо розуміти особливості організації Windows-додатків?
26. Типи даних у програмах Windows.
27. Що таке Функція зворотного виклику?
28. Як обробляти повідомлення про натискання клавіші на клавіатурі?
29. У чому специфіка WM_PAINT?
30. Як змусити систему перемалювати вікно

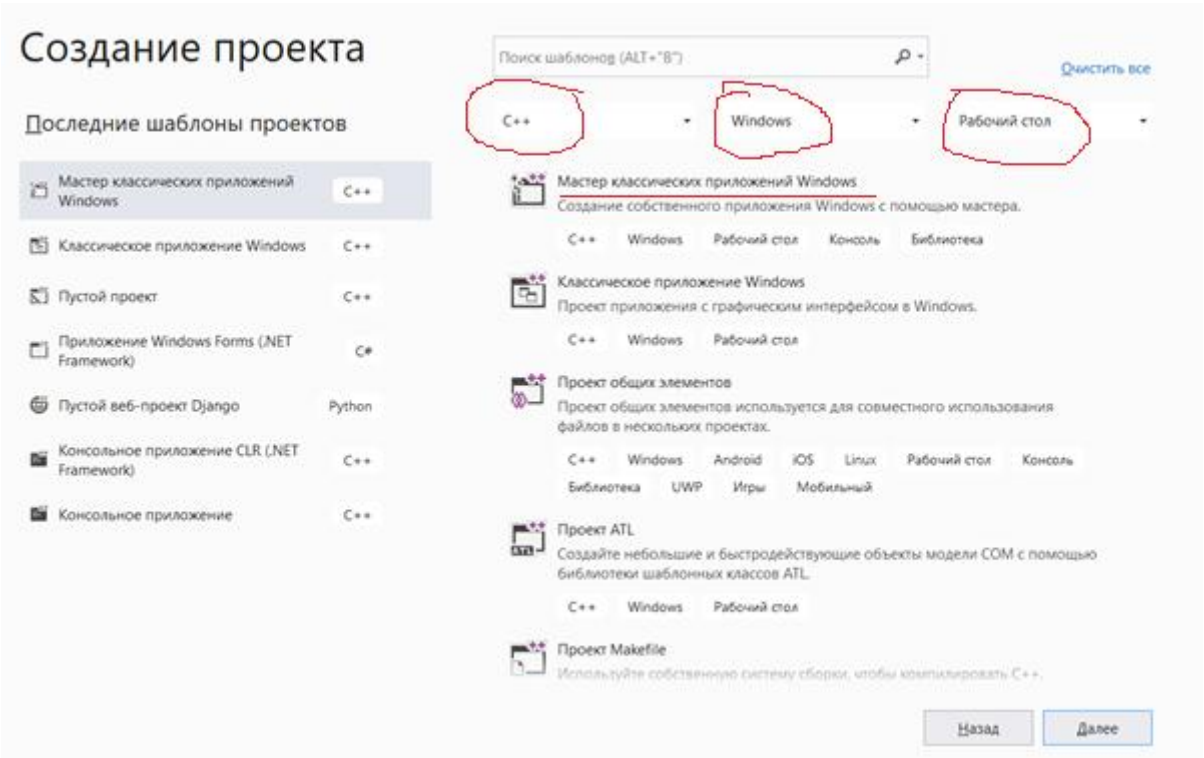
Додаток 1

Створення проекту з використанням Win API в Visual tudio

1. Запустити систему VC++
2. Створити проект для Windows-програми:
Вибрати кнопку **Создание проекта**

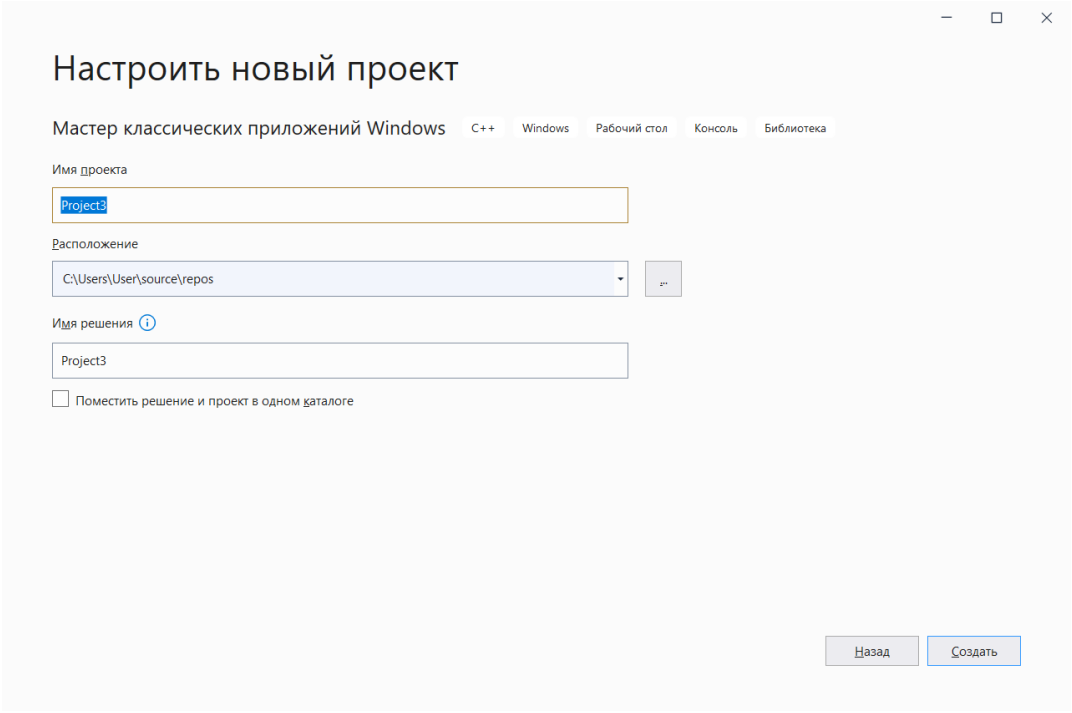


3. У наступному вікні встановити опції C++, Windows, Рабочий стол. Вибрати дію Мастер классических приложений Windows



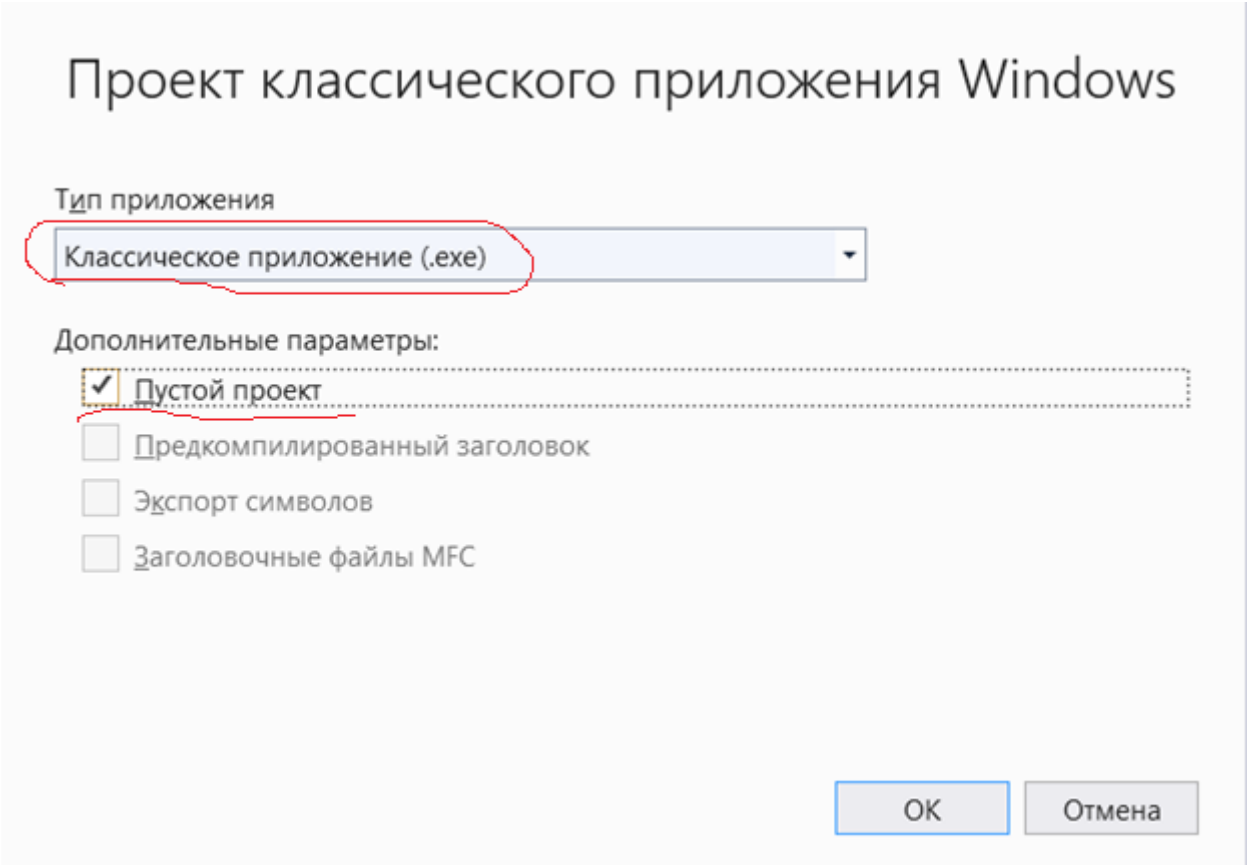
4. Натиснути кнопку Далее

5. У вікні, що з’явилось ввести ім’я проекту



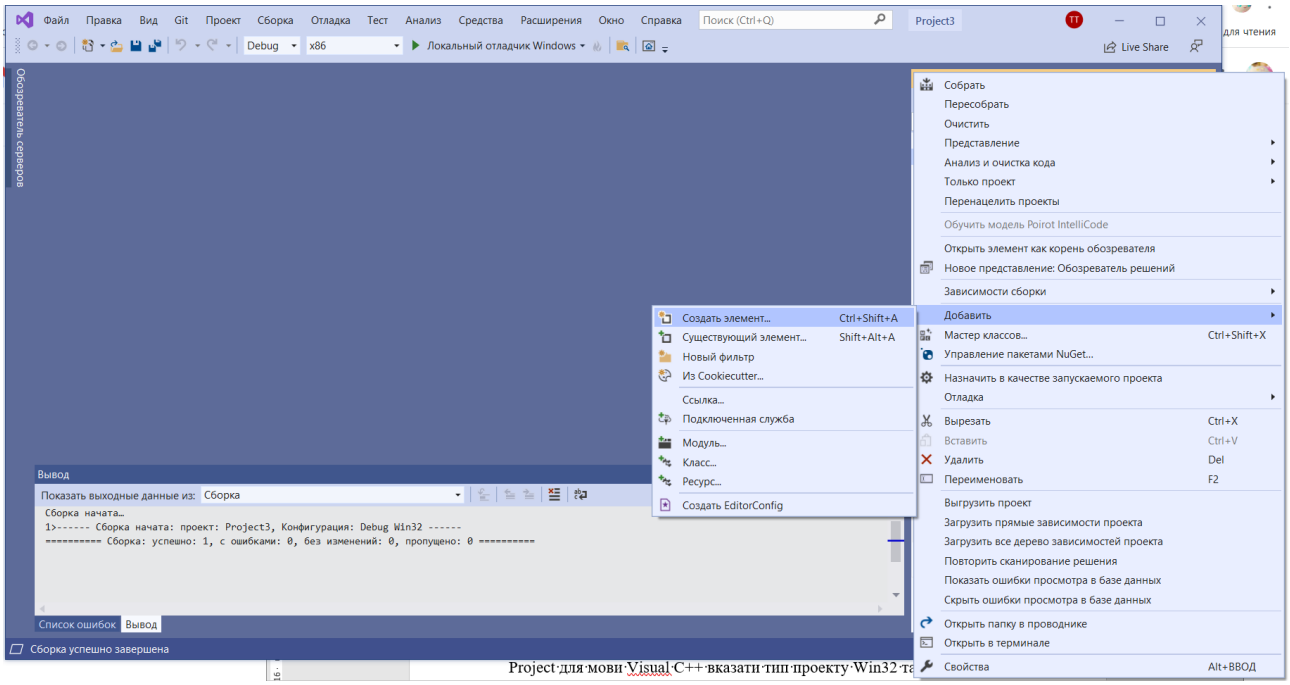
6. Натиснути кнопку Создать

7. У новому вікні обрати Классическое приложение (.exe) та Пустой проект

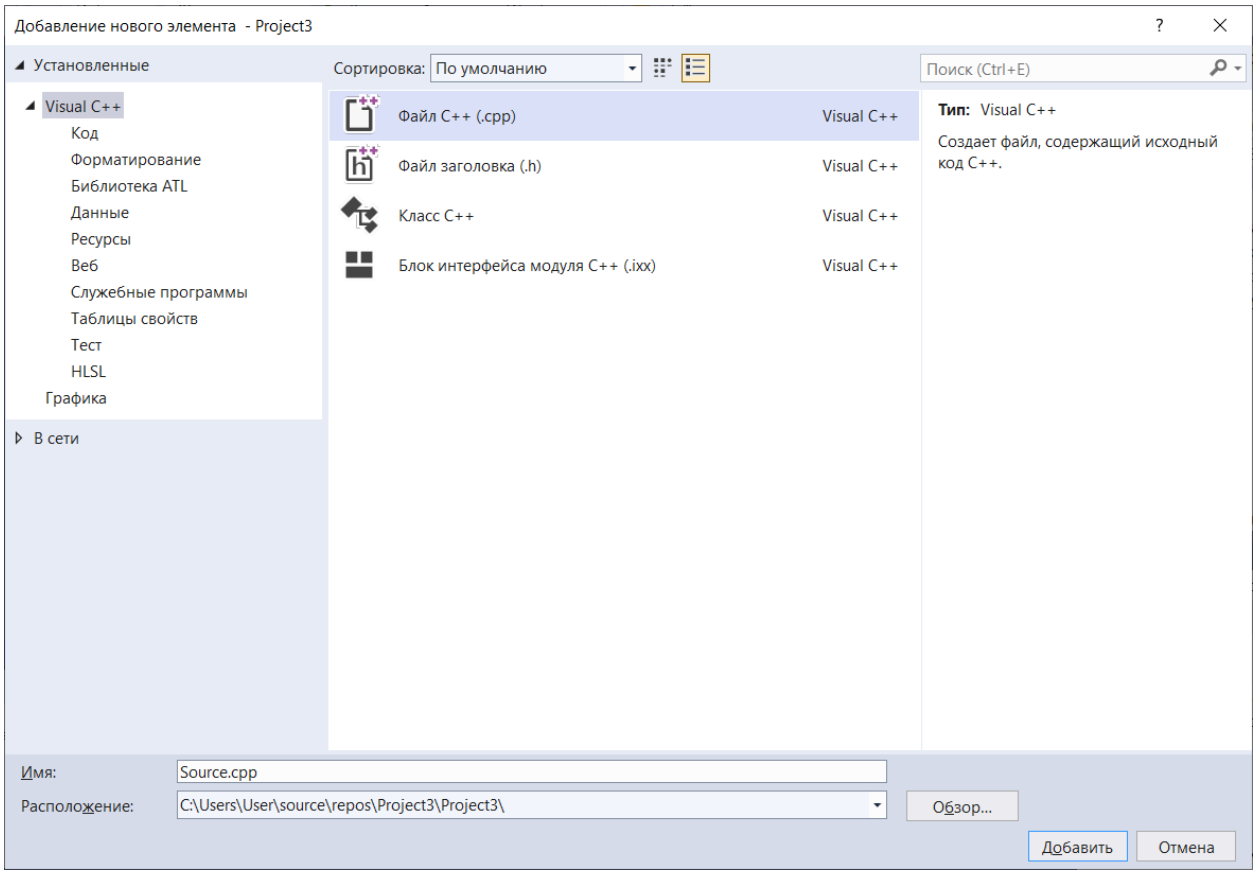


Натиснути кнопку ОК

8. У вікні проекту в Оглядачі рішень натиснути праву кнопку миші на імені проекту. В контекстному меню обрати команду **Добавить | Создать элемент**



9. У вікні, що з’явилось, обрати Файл С++, задати його ім’я та натиснути кнопку Добавить



Додаток 2

Символічні константи для основних повідомлень

Повідомлення від миші

WM_LBUTTONDBLCLK Подвійне натискання лівої кнопки
WM_LBUTTONDOWN Натискання лівої кнопки
WM_LBUTTONUP Відпускання лівої кнопки
WM_MBUTTONDBLCLK Подвійне натискання середньої кнопки
WM_MBUTTONDOWN Натискання середньої кнопки
WM_MBUTTONUP Відпускання середньої кнопки
WM_MOUSEMOVE Переміщення миші
WM_MOUSEWHEEL Переміщення колеса миші
WM_RBUTTONDBLCLK Подвійне натискання правої кнопки
WM_RBUTTONDOWN Натискання правої кнопки
WM_RBUTTONUP Відпускання правої кнопки

Повідомлення від клавіатури

WM_HELP Натискання клавіші F1
WM_KEYDOWN Натиснута клавіша
WM_KEYUP Відпущено клавішу

Віконні повідомлення

WM_ACTIVATE Вікно активне
WM_CAPTURECHANGED Вікно втратило можливість перехоплювати сповіщення від миші
WM_CREATE Вікно було створено
WM_DESTROY Вікно було знищено
WM_KILLFOCUS Вікно втратило фокус уведення
WM_MOVE Вікно було переміщено
WM_SETFOCUS Вікно отримало фокус введення
WM_SIZE Вікно змінило розмір
WM_CLOSE Закрити (знищити) вікно
WM_PAINT Перемалювати клієнтську область вікна

Завершення роботи програми

WM_QUIT Вимога завершити програму