Створення і використання функцій DLL

Стандартні типи даних Windows

Функції, поміщені в DLL, в загальному випадку розраховані на використання в різномовних середовищах програмування. Кожна алгоритмічна мова має свої особливості будови типів даних. Для правильної взаємодії функцій, складених різними алгоритмічними мовами, треба звести до спільного знаменника типи даних, які передають як параметри функції і які отримують як результат виконання функції.

Перше, що треба зробити, - при створенні бібліотеки DLL визначити параметри функції так, щоб їх пізніше можна було легко узгодити при виклику функцій незалежно від алгоритмічної мови. Для розв'язання цієї проблеми Windows надає свої стандартні типи даних, які доцільно використати для будови DLL.

В таблиці нижче подано основні стандартні типи даних:

Windows	**означення**	**аналог C**
`BOOL`	логічний (4-байтовий) має два значення - 0 або 1. При використанні WINAPI прийнято вживати замість 0 специфікатор NULL	`bool` (1-байтовий)
`BYTE`	байт, або восьмибітне беззнакове ціле число	`unsigned char`
`DWORD, UINT`	32-бітове беззнакове ціле	`unsigned long int`
`INT, LONG`	32-бітове ціле	`long int`
`DOUBLE`	8-байтове дійсне число	`double`
`NULL`	нульовий вказівник (4-байтовий) або ціле число 0;	`void * NULL=0;` `int NULL=0;`
`HANDLE`	дескриптор – ідентифікатор якого-небудь об'єкта (4-байтовий); для різних типів об'єктів існують різні дескриптори	
`HCURSOR`	дескриптор курсора (4-байтовий)	
`HDC`	дескриптор контексту пристрою	
`HINSTANCE`	дескриптор екземпляра додатка (програми)	
`HWND`	дескриптор вікна (4-байтовий)	
`LPINT`	вказівник на ціле (4-байтовий)	`int *`
`LPSTR`	вказівник на будь-який рядок, що закінчується нуль-кодом (4-байтовий)	`char *` (4-байтовий)
`LPTSTR`	вказівник на рядок без юнікоду; це надбудова функції `LPSTR`	`char *`
`LPWSTR`	вказівник на UNICODE рядок; це надбудова функції `LPSTR`	`wchar_t *` (4-байтовий)
`CHAR, TCHAR`	символьний тип (1-байтовий)	`char`
`WCHAR`	символьний тип (2-байтовий)	`wchar_t` (2-байтовий)
`LPARAM`	довгий параметр (4-байтовий); використовують разом з `WPARAM` в деяких функціях	
`WPARAM`	параметр-слово (4 байти); використовують разом з `LPARAM` в деяких функціях	
`LRESULT`	значення, яке повертає віконна процедура (4-байтове)	`long`

Підготовка функцій для DLL

Першим кроком треба виконати проектування, програмування і тестування функцій, які будуть поміщені в DLL. Для цього можна створити проект типу “Empty Project”, який не зв’язаний з конкретними типами проектів.

Приклад 1. Функція виконує заміну в рядку літер STRN кожної букви LA на LB. Крім того, обчислює кількість виконаних замін.

Приклад 2. Задано координати масиву точок (x_i,y_i), i=1,...,*N*, на площині. Знайти точку масиву, яка найближча до заданої окремої точки (*x0*,*y0*), а саме: обчислити мінімальну відстань *min* від (*x0*,*y0*) до знайденої точки масиву та номер *k* цієї точки в масиві.

Крок 1. Тестування з використанням типів C++.

Виконуємо розробку і тестування в межах одного середовища програмування C++. В цьому разі достатньо використати стандартні типи даних мови C++, бо виклики функцій відбуваються так само в межах мови C++, і проблеми узгодження типів параметрів не виникає.

```
#pragma once
// 1) повні визначення функцій і тестування з використанням типів C++
#include <iostream>
#include <cmath>
using namespace std;

// прототипи функцій
int (__stdcall ReplLt) (char * STRN, char LA, char LB);
double (__stdcall MinDset) (int N, double xi[], double yi[],
  double x0, double y0, int * k);

// тестування
int main()
{
  // тестування ReplLt()
  cout << "test ReplLt():\n";
  char one[] = "abcdabcdaarrrad";
  cout << one << endl;
  int k = ReplLt(one, 'a', '+');
  cout << one << endl << k << endl;

  // тестування MinDset()
  cout << "\ntest MinDset():\n";
  double xi[6] = { -0.15,  7.6,  2.4, 2.5, -4.4,  2.3 };
  double yi[6] = { -0.58, -0.01, 3.4, 2.9,  1.4, -6.0 };
  double x0 = 1.0;  double y0 = 1.3;
  int N;  // номер найближчої точки з масивів xi[],yi[]
  double minDist;
  minDist = MinDset(6, xi, yi, x0, y0, &N);
  cout << "Min= " << minDist << '\t' << "NumP= " << N << endl;

  system("pause");  return 0;
}
```

```
// повні визначення функцій
int(__stdcall ReplLt)(char * STRN, char LA, char LB)
{
  int cnt = 0;  int i = 0;
  while (STRN[i])
  {
    if (STRN[i] == LA) { STRN[i] = LB; cnt++; }
    i++;
  }
  return cnt;
}

double(__stdcall MinDset)(int N, double xi[], double yi[],
  double x0, double y0, int * k)
// N - кількість точок; xi,yi - масиви координат заданих точок;
// x0,y0 - задана точка;  k - номер знайденої точки;
// результат функції - мінімальна відстань
{
  int i;  double min, d;
  // приймаємо за основу найпершу точку - нумерація від 0
  *k =0;  min = sqrt((xi[0]-x0)*(xi[0]-x0) + (yi[0]-y0)*(yi[0]-y0));
  for (i = 1; i < N; i++)
  {
    d = sqrt((xi[i] - x0)*(xi[i] - x0) + (yi[i] - y0)*(yi[i] - y0));
    if (d < min) { min = d; *k = i; }
  }
  return min;
}
```

Приклад отриманих результатів:

```
test ReplLt():
abcdabcdaarrrad
+bcd+bcd++rrr+d
5

test MinDset():
Min= 2.19317    NumP= 3
```

Крок 2. Тестування з використанням типів Windows.

Як було зазначено раніше, функції, поміщені в DLL, в загальному випадку розраховані на використання в різномовних середовищах програмування. Тому варто організувати повторне тестування тих самих функцій, використавши для параметрів стандартні типи Windows. При цьому треба одночасно замінити типи деяких локальних змінних функцій.

Прототипи функцій за кроком 2 зазвичай подають як інструкцію до використання функцій нашої бібліотеки. Тоді користувач має узгодити типи даних іншої А-мови програмування за схемою:

типи Windows ↔ типи А-мови

Кроку 2 можна не виконувати, тоді інструкцією до бібліотеки будуть прототипи функцій мовою С++, і узгодження відбувається за схемою:

типи С++ ↔ типи А-мови

```
#pragma once
// 2) тестування функцій з використанням типів Windows
#include <iostream>
#include <cmath>
```

```
#include <Windows.h>  // щоб використати типи Windows
using namespace std;

// прототипи функцій
INT (__stdcall ReplLt) (LPTSTR STRN, CHAR LA, CHAR LB);
DOUBLE (__stdcall MinDset) (INT N, DOUBLE xi[], DOUBLE yi[],
  DOUBLE x0, DOUBLE y0, LPINT k);

int main()  // тестування
{
  // тестування ReplLt()
  cout << "test ReplLt():\n";
  char one[] = "abcdabcdaarrrad";
  cout << one << endl;
  int k = ReplLt(one, 'a', '+');
  cout << one << endl << k << endl;

  // тестування MinDset()
  cout << "\ntest MinDset():\n";
  double xi[6] = { -0.15,  7.6,  2.4, 2.5, -4.4,  2.3 };
  double yi[6] = { -0.58, -0.01, 3.4, 2.9,  1.4, -6.0 };
  double x0 = 1.0;  double y0 = 1.3;
  int N;  // номер найближчої точки з масивів xi[],yi[]
  double minDist;
  minDist = MinDset(6, xi, yi, x0, y0, &N);
  cout << "Min= " << minDist << '\t' << "NumP= " << N << endl;

  system("pause");  return 0;
}

// повні визначення функцій за типами Windows
INT (__stdcall ReplLt)(LPTSTR STRN, CHAR LA, CHAR LB)
{
  INT cnt = 0;  int i = 0;
  while (STRN[i])
  {
    if (STRN[i] == LA) { STRN[i] = LB; cnt++; }
    i++;
  }
  return cnt;
}

DOUBLE (__stdcall MinDset)(INT N, DOUBLE xi[], DOUBLE yi[],
  DOUBLE x0, DOUBLE y0, LPINT k)
// N - кількість точок; xi,yi - масиви координат заданих точок;
// x0,y0 - задана точка;  k - номер знайденої точки;
// результат функції - мінімальна відстань
{
  int i;  DOUBLE min, d;
  // приймаємо за основу найпершу точку - нумерація від 0
  *k =0;  min = sqrt((xi[0]-x0)*(xi[0]-x0) + (yi[0]-y0)*(yi[0]-y0));
  for (i = 1; i < N; i++)
  {
    d = sqrt((xi[i] - x0)*(xi[i] - x0) + (yi[i] - y0)*(yi[i] - y0));
    if (d < min) { min = d; *k = i; }
  }
  return min;
}
```

Компіляція функцій в бібліотеку DLL (створення DLL)

Побудовані попереднім кроком функції тепер використаємо для компіляції в бібліотеку DLL. Процедура створення DLL регламентована в кожній системі програмування відповідною інструкцією, отже треба її отримати і виконати.

Після запуску Visual Studio:

New Project → Windows Desktop → Dynamic Link Library (DLL) → визначити Name і Location → Create directory for solution → OK

Нехай ім'я Name файла проекту DLLCreate.cpp.

У вікні редактора коду файла DLLCreate.cpp друкуємо (копіюємо з файла тестування) тексти наших функцій, визначаючи додатково директиви експорту функцій. При цьому вирішуємо, чи додати в бібліотеку функцію входу DllMain. Ця функція має шаблон визначення в зв'язаному файлі dllmain.cpp, який редагуємо за потреби (відкриваємо через Solution Explorer).

```
// DLLCreate.cpp : Defines the exported functions for the DLL application.
#include "stdafx.h"
#include <cmath>
using namespace std;

// список експортованих функцій з використанням типів C++
extern "C" __declspec(dllexport)
    int(__stdcall ReplLt) (char * STRN, char LA, char LB);
extern "C" __declspec(dllexport)
    double(__stdcall MinDset) (int N, double xi[], double yi[],
       double x0, double y0, int * k);

// повні визначення функцій
int(__stdcall ReplLt)(char * STRN, char LA, char LB)
{
  int cnt = 0;  int i = 0;
  while (STRN[i])
  {
    if (STRN[i] == LA) { STRN[i] = LB; cnt++; }
    i++;
  }
  return cnt;
}

double(__stdcall MinDset)(int N, double xi[], double yi[],
  double x0, double y0, int * k)
// N - кількість точок; xi,yi - масиви координат заданих точок;
// x0,y0 - задана точка;  k - номер знайденої точки;
// результат функції - мінімальна відстань
{
  int i;  double min, d;
  // приймаємо за основу найпершу точку - нумерація від 0
  *k = 0;  min = sqrt((xi[0]-x0)*(xi[0]-x0) + (yi[0]-y0)*(yi[0]-y0));
  for (i = 1; i < N; i++)
  {
    d = sqrt((xi[i]-x0)*(xi[i]-x0) + (yi[i]-y0)*(yi[i]-y0));
    if (d < min) { min = d; *k = i; }
  }
  return min;
}
```

Компілюємо DLL: Build → Build Solution.

Якщо все зроблено правильно, у вікні Output має бути приблизно таке:

```
1>------ Build started: Project: DLLCreate, Configuration: Debug Win32 ------
1>   Creating library D:\V_V\CPP Lecture\CPP_2016\CPPTest основи\
        DLLCreate\Debug\DLLCreate.lib
        and object D:\V_V\CPP Lecture\CPP_2016\CPPTest основи\
        DLLCreate\Debug\DLLCreate.exp
1>DLLCreate.vcxproj -> D:\V_V\CPP Lecture\CPP_2016\CPPTest основи\
        DLLCreate\Debug\DLLCreate.dll
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

Зауважимо важливу деталь. До проекту підключено бібліотеку `#include <cmath>` і використано стандартну функцію `sqrt()`. В результаті компіляції скомпонований файл нашої DLL буде мати функцію `sqrt()` вже як свою власну частину, тому підключати подібні стандартні функції для виконання програм користувача вже не треба.

В результаті в папці `Debug` цілого проекту мають бути файли `DLLCreate.dll`, `DLLCreate.lib` та інші. Названі два файли будуть потрібні для використання в інших програмах користувачів, тому їх варто копіювати в окрему від проекту папку, до якої будуть мати доступ прикладні програми користувачів. Необхідно, щоб ці два файли залишались в межах однієї папки.

Використання DLL. Неявне зв'язування

Неявне зв'язування з DLL виконують на етапі компіляції і будови прикладної програми користувача. Середовище програмування має самостійно виконати пошук бібліотеки і зв'язати потрібні функції DLL з прикладною програмою. В результаті компонування будуть автоматично додані засоби операційної системи для завантаження DLL під час виконання прикладної програми і виклику функцій з DLL.

Покажемо процедуру неявного зв'язування на прикладі проекту типу "Empty Project". Використаємо для тестування зв'язку з DLL ту саму функцію `main()`, яка була подана вище при розробці функцій DLL.

Крок 1. Проектування прикладної програми.

Створюємо новий проект "Empty Project", додаємо до нього файл `*.cpp`, і записуємо текст:

```
#include <iostream>
using namespace std;

// список імпортованих функцій з використанням типів C++
extern "C" __declspec(dllimport)
    int(__stdcall ReplLt) (char * STRN, char LA, char LB);
extern "C" __declspec(dllimport)
    double(__stdcall MinDset) (int N, double xi[], double yi[],
    double x0, double y0, int * k);

// тестування
int main()
{
  // тестування ReplLt()
  cout << "test ReplLt():\n";
  char one[] = "abcdabcdaarrrad";
  cout << one << endl;
```

```
  int k = ReplLt(one, 'a', '+');
  cout << one << endl << k << endl;

  // тестування MinDset()
  cout << "\ntest MinDset():\n";
  double xi[6] = { -0.15,  7.6,  2.4, 2.5, -4.4,  2.3 };
  double yi[6] = { -0.58, -0.01, 3.4, 2.9,  1.4, -6.0 };
  double x0 = 1.0;  double y0 = 1.3;
  int N;  // номер найближчої точки з масивів xi[],yi[]
  double minDist;
  minDist = MinDset(6, xi, yi, x0, y0, &N);
  cout << "Min= " << minDist << '\t' << "NumP= " << N << endl;

  system("pause");  return 0;
}
```

В прототипах функцій визначаємо їх параметром `dllimport`, тобто такими, які є в зовнішній бібліотеці. Для правильної компіляції нашої програми прототипів достатньо. Щоб переконатись, виконаємо команду `Build→Compile`, і побачимо повідомлення про успішну компіляцію.

Крок 2. Налаштування зв'язку з файлом імпорту `*.lib`.

Якщо спробувати виконати компонування `Build→Build Solution`, то побачимо повідомлення завантажувача (компонувальника) про нерозв'язані зовнішні посилання на функції. Компонувальнику потрібний файл імпорту `*.lib`.

Отже треба виконати налаштування для пошуку файла `DLLCreate.lib`.

Для цього виконуємо такі дії (для режиму Debug):

2.1. Відкрити вікно Project→Properties→Property Pages.

2.2. На лівій панелі вибрати Configuration Properties→Linker→Input.

2.3. На панелі властивостей вибрати Additional Dependencies і кнопкою вибрати <Edit...>.

2.4. У вікні Additional Dependencies у верхній частині друкуємо назву файла імпорту: DLLCreate.lib.

2.5. Натиснути OK, щоб повернутись до сторінки властивостей.

2.6. На лівій панелі вибираємо Configuration Properties→Linker→General.

2.7. На панелі властивостей вибрати Additional Library Directories, відкрити кнопкою вікно і записати шлях до розташування файла імпорту: `d:\MyDLLfun\`

2.8. Кнопками OK закриваємо вікна. Тепер прикладну програму можна успішно компілювати і компонувати, але не виконувати.

Зазначені дії стосуються системи Visual Studio. В інших системах програмування можуть бути потрібні інакші способи налаштування. В будь-якому разі необхідні дві речі: назва файла імпорту і шлях до розташування файла імпорту.

Крок 3. Налаштування зв'язку з файлом бібліотеки `*.dll`.

Якщо спробувати виконати програму, налаштовану лише на файл імпорту попереднього кроку, отримаємо подібне повідомлення:

```
The program '[1716] DLLuseImplicit.exe' has exited with code -1073741515
(0xc0000135) 'A dependent dll was not found'.
```

Під час виконання програми операційна система намагатиметься шукати саму бібліотеку `DLLCreate.dll`. За правилами пошук виконують в такому порядку:

1) пошук в каталозі, звідки запущена програма;

2) пошук в поточному каталозі;
3) пошук в системному каталозі (GetSystemDirectory);
4) пошук в каталозі Windows (GetWindowsDirectory);
5) пошук в каталогах, визначених в середовищі (PATH).

Якщо пошук виявився невдалим, програма не буде виконана.

Найпростіший спосіб уникнути цієї проблеми – копіювати DLL в каталог, що містить виконуваний EXE-файл прикладної програми. Для середовища Visual Studio копіювати DLL треба на початку в основну папку проекту – туди, де файли текстів програми.

Інші варіанти – копіювати DDL в одну з системних папок Windows, або за вказівкою змінної PATH.

Крок 4. Будова остаточного варіанта прикладної програми.

Якщо прикладна програма вважається остаточно налагодженою, будують фінальний випуск програми Release.

4.1. Project→Properties→ Property Pages→Configuration→Release.

4.2. Повторити налаштування 2.2-2.8 для режиму Release.

4.3. Після повернення до головного вікна наа панелі інструментів обираємо Solution Configuration→Release.

4.4. Повторно компілюємо `Build→Compile` і компонуємо програму `Build→Build Solution`.

4.5. Має з’явитись папка Release з EXE-файлом та іншими. Цю папку зазвичай передають користувачам нашої програми.

4.6. Програму можна виконати в середовищі програмування, командою Local Windows Debugger (кнопка ►). В цьому разі DDL підключається автоматично самою системою. Якщо ж спробувати виконати EXE-програму зовнішнім запуском з папки Release, то отримаємо повідомлення про відсутність бібліотеки `DLLCreate.dll`.

Крок 5. Копіювання файла бібліотеки `DLLCreate.dll`.

Копіюємо будь-яким способом файл `DLLCreate.dll` в папку Debug чи Release, залежно від остаточного варіанта програми.

Запускаємо програму на виконання зовні. Виконуємо за потреби додаткове тестування.

Використання DLL. Явне зв’язування

Альтернативним для неявного є явне зв'язування, коли динамічну бібліотеку завантажують в адресний простір процесу виконанням системного виклику з його коду. Після цього, використовуючи інший системний виклик, застосування отримує адресу необхідної йому функції бібліотеки і може її викликати. Після використання бібліотеку можна вилучити з пам'яті. Компонувальник при цьому нічого про неї не знає, завантажувач ОС автоматично бібліотек не завантажує, отже компіляція і запуск застосування відбуваються без клопотів щодо пошуку і підключення бібліотеки.

Зауважимо, що розглянуте вище неявне зв'язування здебільшого зводиться до автоматичного виконання тих самих викликів, які сам програміст виконує за явного.

Такий підхід вимагає від програміста додаткових зусиль, але має більшу гнучкість.

Для реалізації явного зв’язування треба виконати такі кроки.

Крок 1. Визначити місце розташування та ім'я бібліотеки DLL:

```
char AllocDLL[] = "d:\\MyDLLfun\\";
char LibName[] = "DLLCreate.dll";
```

Крок 2. Оголошення типів вказівників на зовнішні функції DLL:

```
typedef int (CALLBACK* TReplLt)(char*, char, char);
typedef double (CALLBACK* TMinDset)(int, double[], double[],
                double, double, int*);
//  #define  CALLBACK    __stdcall
```

Крок 3. Завантажити і перевірити наявність бібліотеки:

```
  HINSTANCE hDLL;  // дескриптор бібліотеки
  int buffersize = strlen(AllocDLL) + strlen(LibName) + 2;
      // розмір пам'яті для повного імені DLL
  char * FullName = new char[buffersize]; // місце для імені DLL
  strcpy_s(FullName, buffersize, AllocDLL); // копіювати шлях до файла
  strcat_s(FullName, buffersize, LibName);
          // додати ім'я файла = повне ім'я
  hDLL = LoadLibrary(FullName);  // завантаження бібліотеки
  // перевірка присутності
  if (hDLL == NULL) {  // якщо немає
    cout << "DLL not found.\n";
    system("pause");  return 0;  //  то припинити програму
  }
```

Крок 4. Побудувати вказівники на функції і перевірити наявність функцій в бібліотеці:

```
  TReplLt ReplLt = (TReplLt)GetProcAddress(hDLL, "_ReplLt@12");
                  // ім'я змінено
  if (!ReplLt) {
    cout << "Function ReplLt not found.\n";
    FreeLibrary(hDLL); system("pause"); return 0;
            // якщо немає - то вихід
  }
```

Тут потрібно зробити пояснення щодо імен функцій бібліотеки DLL. Компілятори мови C++ (а також деякі компілятори інших мов, наприклад, MASM) перетворюють імена функцій так, щоб відобразити використаний спосіб передавання параметрів. Так, до імен всіх функцій, які використовують конвенцію C, дописують спереду знак підкреслення. В кінці імені додають знак @ і розмір ділянки стеку в байтах, яку займають параметри. Отже, ім'я `"ReplLt"` буде перетворене до `"_ReplLt@12"`. Аналогічно будуть перетворені імена інших функцій.

Щоб точно знати, як виглядають перетворені імена, можна скористатись окремими засобами перегляду файлів. Переглядати можна файл `*.dll` або файл `*.lib`, секції Imports/Exports або Library Header відповідно.

Крок 5. В процесі виконання програми викликати функції як звичайно:

```
  char one[] = "abcdabcdaarrrad";
  int k = ReplLt(one, 'a', '+');
```

Звернемо увагу, що тут `ReplLt()` є не іменем функції, а оголошеним вказівником типу `TReplLt` на функцію. Проте, мова C++ не робить різниці між іменами і вказівниками на функції.

Крок 6. В кінці виконання програми звільнити бібліотеку:

```
FreeLibrary(hDLL);
```

Повний приклад програми з явним зв'язуванням

```
#include <iostream>
#include <Windows.h>
using namespace std;

// місце розташування та ім'я бібліотеки DLL
char AllocDLL[] = "d:\\MyDLLfun\\";
char LibName[] = "DLLCreate.dll";

// оголошення типів вказівників на зовнішні функції DLL
typedef int (CALLBACK* TReplLt)(char*, char, char);
typedef double (CALLBACK* TMinDset)(int, double[], double[], double,
                double, int*);
//  #define  CALLBACK   __stdcall

// тестування
int main()
{
  // завантажити і перевірити наявність бібліотеки
  HINSTANCE hDLL;  // дескриптор бібліотеки
  int buffersize = strlen(AllocDLL) + strlen(LibName) + 2;
      // розмір пам'яті для повного імені DLL
  char * FullName = new char[buffersize]; // місце для імені DLL
  strcpy_s(FullName, buffersize, AllocDLL); // копіювати шлях до файла
  strcat_s(FullName, buffersize, LibName);
           // додати ім'я файла = повне ім'я
  hDLL = LoadLibrary(FullName);  // завантаження бібліотеки
  // перевірка присутності
  if (hDLL == NULL) {  // якщо немає
    cout << "DLL not found.\n";
    system("pause");  return 0;  //  то припинити програму
  }

  // побудувати вказівники на функції
  //   і перевірити наявність функцій в бібліотеці
  TReplLt ReplLt = (TReplLt)GetProcAddress(hDLL, "_ReplLt@12");
                  // ім'я змінено
  if (!ReplLt) {
    cout << "Function ReplLt not found.\n";
    FreeLibrary(hDLL); system("pause"); return 0;
    // якщо немає - то вихід
  }

  TMinDset MinDset = (TMinDset)GetProcAddress(hDLL, "_MinDset@32");
                   // ім'я змінено
  if (!MinDset) {
    cout << "Function MinDset not found.\n";
    FreeLibrary(hDLL); system("pause"); return 0;
    // якщо немає - то вихід
  }

  // тепер можна викликати функції як звичайно
  // тестування ReplLt()
  cout << "test ReplLt():\n";
```

```
  char one[] = "abcdabcdaarrrad";
  cout << one << endl;
  int k = ReplLt(one, 'a', '+');
  cout << one << endl << k << endl;

  // тестування MinDset()
  cout << "\ntest MinDset():\n";
  double xi[6] = { -0.15,  7.6,  2.4, 2.5, -4.4,  2.3 };
  double yi[6] = { -0.58, -0.01, 3.4, 2.9,  1.4, -6.0 };
  double x0 = 1.0;  double y0 = 1.3;
  int N;  // номер найближчої точки з масивів xi[],yi[]
  double minDist;
  minDist = MinDset(6, xi, yi, x0, y0, &N);
  cout << "Min= " << minDist << '\t' << "NumP= " << N << endl;

  // в кінці звільнити бібліотеку
  FreeLibrary(hDLL);
  delete[] FullName;
  system("pause");  return 0;
}
```

Приклад отриманих результатів:

```
test ReplLt():
abcdabcdaarrrad
+bcd+bcd++rrr+d
5

test MinDset():
Min= 2.19317    NumP= 3
```

.