Laboratorium Architektury Komputerów
i Systemów Operacyjnych

Ćwiczenie 3

Interfejs programowania aplikacji (API)

Wprowadzenie

Interfejs programowania aplikacji, nazywany w skrócie API (ang. application program interface), obejmuje zbiór struktur danych i funkcji, a niekiedy stanowi listę komunikatów. API stanowi ustaloną konwencję wywoływania, za pomocą której program użytkowy (aplikacja) może uzyskać dostęp do funkcji systemu operacyjnego lub funkcji udostępnianych przez inne moduły oprogramowania, które zwykle implementowane są jako biblioteki. API definiowane jest na poziomie kodu źródłowego i stanowi pewien poziom abstrakcji między aplikacją a jądrem systemu operacyjnego (lub innego programu usługowego), co z kolei tworzy potencjalne możliwości przenośności kodu.

Rozmaite interfejsy API są szeroko wykorzystywane w praktyce programowania, chociaż skrót API nie zawsze jest stosowany. Aktualnie, do najbardziej popularnych API dla komputerów osobistych należy opracowany przez firmę Microsoft pakiet Win32 API, zawierający opisy funkcji używanych w systemie Windows. Do połowy lat dziewięćdziesiątych w oprogramowaniu komputerów osobistych szeroko stosowano zestaw funkcji DOS API, który definiowany był na poziomie asemblera. W środowisku systemu Linux dominuje POSIX API. Rozmaite pakiety API używane są w prawie każdym systemie komputerowym.

Przykładowa funkcja API

Poniżej podano fragmenty oryginalnego opisu funkcji usługowej GetDiskFreeSpaceEx udostępnianej w ramach Windows API. Funkcja ta podaje informacje o rozmiarze niezajętych obszarów pamięci dyskowej.

GetDiskFreeSpaceEx Function

Retrieves information about the amount of space that is available on a disk volume, which is the total amount of space, the total amount of free space, and the total amount of free space available to the user that is associated with the calling thread.

```
BOOL  WINAPI  GetDiskFreeSpaceEx (
  __in          LPCTSTR  lpDirectoryName,
  __out         PULARGE_INTEGER  lpFreeBytesAvailable,
  __out         PULARGE_INTEGER  lpTotalNumberOfBytes,
  __out         PULARGE_INTEGER  lpTotalNumberOfFreeBytes
) ;
```

Parameters

lpDirectoryName
A directory on the disk.
If this parameter is NULL, the function uses the root of the current disk.
If this parameter is a UNC name, it must include a trailing backslash, for example, \\MyServer\MyShare\.
This parameter does not have to specify the root directory on a disk. The function accepts any directory on a disk.
The calling application must have FILE_LIST_DIRECTORY access rights for this directory.

lpFreeBytesAvailable
A pointer to a variable that receives the total number of free bytes on a disk that are available to the user who is associated with the calling thread.
This parameter can be NULL.
If per-user quotas are being used, this value may be less than the total number of free bytes on a disk.

lpTotalNumberOfBytes
A pointer to a variable that receives the total number of bytes on a disk that are available to the user who is associated with the calling thread.
This parameter can be NULL.
If per-user quotas are being used, this value may be less than the total number of bytes on a disk.

lpTotalNumberOfFreeBytes
A pointer to a variable that receives the total number of free bytes on a disk.
This parameter can be NULL.

Return Value

If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero (0). To get extended error information, call GetLastError.

Remarks

The values obtained by this function are of the type ULARGE_INTEGER. Do not truncate these values to 32 bits.
The **GetDiskFreeSpaceEx** function returns zero (0) for *lpTotalNumberOfFreeBytes* and *lpFreeBytesAvailable* for all CD requests unless the disk is an unwritten CD in a CD-RW drive.
Symbolic link behavior—If the path points to a symbolic link, the operation is performed on the target.

Requirements

Client	Requires Windows Vista, Windows XP, or Windows 2000 Professional.
Server	Requires Windows Server 2008, Windows Server 2003, or Windows 2000 Server.
Header	Declared in WinBase.h; include Windows.h.
Library	Use Kernel32.lib.
DLL	Requires Kernel32.dll.
Unicode	Implemented as **GetDiskFreeSpaceExW** (Unicode) and **GetDiskFreeSpaceExA** (ANSI).

See Also

Disk Management Functions
GetDiskFreeSpace
GetModuleHandle
GetProcAddress
Build date: 8/15/2007

Podana tu funkcja GetDiskFreeSpaceEx może być używana w programach wykonywanych w systemie Windows, a wszystkie szczegóły dotyczące sposobu jej wywołania, znaczenia poszczególnych parametrów i ich dopuszczalnych wartości zawarte są w dokumentacji, której

fragment został przytoczony powyżej. Tak więc opis interfejsu Win API ma postać dokumentu (tekstowego), w którym podano opisy funkcji systemowych, struktur i komunikatów udostępnianych dla programów użytkowych. W dalszej części podany jest program przykładowy ilustrujący zastosowanie tej funkcji.

W Unixie (lub w Linuksie) można wymienić dla przykładu zestaw funkcji wykonujących operacje na plikach: *open, close, lseek, read* i inne. Dla każdej z tych funkcji podawany jest opis funkcjonalny wraz ze szczegółowymi informacjami odnośnie przekazywania parametrów do funkcji i interpretacji wartości zwracanych przez te funkcje. Takie opisy nie zawierają na ogół jakichkolwiek danych o sposobie wykonywania tych funkcji przez system operacyjny.

Tak więc z jednej strony interfejsy API pozwalają na wykorzystanie usług oferowanych przez system, ale ukrywają wewnętrzne działania systemu, co określane jest jako pewien poziom abstrakcji, rozumianej jako ukrycie szczegółów implementacyjnych. Pominięcie szczegółów implementacyjnych zazwyczaj ułatwia zrozumienie funkcji, co stanowi jeden z czynników przyśpieszających budowę systemu oprogramowania. Koncepcja ta koresponduje z zasadami hermetyzacji w językach obiektowych.

Warto zwrócić uwagę, że podobnie jak w przypadku katalogów podzespołów elektronicznych, korzystanie z informacji podanych w opisie API wymaga pewnego przygotowania. Wprawdzie katalogi podają niekiedy schematy przykładowych zastosowań, podobnie jak w opisach API można znaleźć fragmenty programów, to jednak zrozumienie reguł ich działania wymaga uprzedniego przestudiowania wybranej tematyki.

API a sterowanie urządzeniami zewnętrznymi komputera

W okresie kształtowania się podstawowych konstrukcji systemów operacyjnych znaczną ich część stanowiły podprogramy obsługi urządzeń zewnętrznych komputera. Wydzielenie tych podprogramów nastąpiło w naturalnym procesie rozwoju oprogramowania ze względu na złożoność procesów obsługi urządzeń, możliwość ich uszkodzenia lub przedwczesnego zużycia przy nieprawidłowej eksploatacji. Stopniowo też, wraz z wprowadzaniem mechanizmów ochrony w systemach operacyjnych, wprowadzono zasadę obowiązkowego pośrednictwa usług systemowych przy korzystaniu z urządzeń zewnętrznych. W ten sposób ukształtowała się grupa funkcji usługowych API przeznaczonych do sterowania pracą urządzeń zewnętrznych komputera.

W współczesnych systemach operacyjnych, w których stosowana jest wielozadaniowość i wynikająca z niej wirtualizacja urządzeń, bezpośrednie sterowanie urządzeniem mogłoby być stosowane jedynie w przypadku oddania urządzenia do wyłącznej dyspozycji jednego procesu. Ewentualne błędy sterowania mogłyby zachwiać pracą całego systemu.

Ponieważ API nie precyzuje sposobu realizacji funkcji, więc ta sama funkcja może być zrealizowana w różnych środowiskach za pomocą różnych urządzeń, co znacznie wspomaga ewentualną przenośność programów.

Z punktu widzenia programisty, dostępność pakietu operacji wykonujących działania na urządzeniach komputera uwalnia go od konieczności analizowania zasad sterowania poszczególnych urządzeń i tworzenia oprogramowania dla nich.

Program przykładowy

Podany niżej program wyświetla rozmiar wolnego miejsca na wskazanym dysku w bajtach. Do tego celu używana jest funkcja GetDiskFreeSpaceEx, której opis podano powyżej. Jak wynika z opisu funkcja ta ma cztery parametry, których znaczenie jest następujące:

- *lpDirectoryName* — symbol dysku przedstawiony w postaci łańcucha znaków będącego ścieżką dostępu do katalogu głównego; wartość NULL reprezentuje dysk bieżący;
- *lpFreeBytesAvailable* — wskaźnik (adres) do zmiennej typu ULARGE_INTEGER (liczba 64-bitowa bez znaku) — do zmiennej tej zostanie wpisana liczba bajtów dostępnych dla programu wywołującego;
- *lpTotalNumberOfBytes* — wskaźnik (adres) do zmiennej typu ULARGE_INTEGER — do zmiennej tej zostanie wpisana całkowita liczba bajtów na wskazanym dysku;
- *lpTotalNumberOfFreeBytes* — wskaźnik (adres) do zmiennej typu ULARGE_INTEGER — do zmiennej tej zostanie wpisana całkowita liczba wolnych bajtów na wskazanym dysku.

Ponieważ przy stosowanych aktualnie dyskach o pojemności przekraczającej 1 TB, rozmiar dysku liczony w bajtach będzie liczbą zawierającą więcej niż 32 bity, z tego powodu właściwe obliczenie wykonywane jest z użyciem wartości 64-bitowych (bez znaku) typu ULARGE_INTEGER (można też stosować omawiany wcześniej typ unsigned __int64). Można jeszcze dodać, że poprzedzenie zmiennej symbolem & oznacza adres (wskaźnik) tej zmiennej.

```
#include <windows.h>
#include <stdio.h>

int main()
{
  ULARGE_INTEGER
     l_dostepnych,  // liczba bajtów na dysku dostępnych
                    // dla programu,
     l_pojemnosc,   // liczba bajtów na dysku
     l_wolnych ;    // liczba wolnych bajtów na dysku;

  LPCSTR dysk = "d:\\";

  GetDiskFreeSpaceEx (dysk, &l_dostepnych, &l_pojemnosc,
                                           &l_wolnych);

  printf("\nLiczby bajtow na dysku %s\n", dysk);
  printf("\nLiczba bajtow dostepnych dla programu = %I64d\n",
                                           l_dostepnych);
  printf("\nCalkowita liczba bajtow = %I64d\n", l_pojemnosc);
  printf("\nCalkowita liczba wolnych bajtow = %I64d\n",
                                           l_wolnych);

  return 0;
}
```

Definiowanie poprawnego i spójnego API

W odniesieniu do API stawia się zazwyczaj pewne wymagania. Oczywiście interfejs musi być poprawny, spójny, podatny na dalsze rozszerzenia i dobrze udokumentowany. Interfejs API powinien być przyjazny dla programisty, zwłaszcza gdy przewidywane jest jego stosowanie przez programistów nie znających dokładnie danej problematyki. API powinien być w miarę możliwości niezależny od platformy. Dalej wskazane jest by kolejne wersje pewnej klasy API zachowywały zgodność z wersjami wcześniejszymi, a jeśli konieczna jest zmiana to lepszym rozwiązaniem jest zdefiniowanie nowej funkcji.

W celu zilustrowania podanych koncepcji spróbujmy zdefiniować dwie nowe funkcje, które również podają informacje o zajętości dysku, ale są lepiej dostosowane do typowych zastosowań praktycznych. W szczególności funkcje wymagają podania tylko jednego parametru, a wyznaczone wartości w postaci liczby megabajtów zwracane są przez nazwę funkcji, co ułatwia kodowanie programu. Poniżej podano opisy tych funkcji w postaci zbliżonej do stosowanej przez producentów oprogramowania — opisy te stanowią interfejs API.

PodajRozmiarObszaruNiezajetego Funkcja

Funkcja podaje informacje o rozmiarze niezajętego obszaru na dysku, wyrażone w megabajtach.

```
unsigned int PodajRozmiarObszaruNiezajetego(
            char symbol_dysku
);
```

Parametry

symbol_dysku

Symbol dysku w postaci pojedynczej litery, np. E lub e.

Zwracane wartości

W przypadku poprawnego wykonania funkcja zwraca rozmiar wolnego obszaru na wskazanym dysku, wyrażony w megabajtach.
W przypadku błędu wykonania funkcja zwraca wartość 0xFFFFFFFF (tj. –1).

Wymagania

Klient	Wymaga systemu: wersja Windows nie starsza niż Windows 2000 Professional.
Nagłówek	Deklarowany w pliku rozmiar_dysku.h.
Biblioteka	dysk_info.lib
DLL	dysk_info.dll

Data utworzenia: 21.10.2013

PodajRozmiarObszaruZajetego Funkcja

Funkcja podaje informacje o rozmiarze zajętego obszaru na dysku, wyrażone w megabajtach.

```
unsigned int PodajRozmiarObszaruZajetego(
            char symbol_dysku
);
```

Parametry

symbol_dysku

Symbol dysku w postaci pojedynczej litery, np. E lub e.

Zwracane wartości

W przypadku poprawnego wykonania funkcja zwraca rozmiar zajętego obszaru na wskazanym dysku, wyrażony w megabajtach.
W przypadku błędu wykonania funkcja zwraca wartość 0xFFFFFFFF (tj. –1).

Wymagania

Klient	Wymaga systemu: wersja Windows nie starsza niż Windows 2000 Professional.
Nagłówek	Deklarowany w pliku rozmiar_dysku.h.
Biblioteka	dysk_info.lib
DLL	dysk_info.dll

Data utworzenia: 21.10.2013

Biblioteki statyczne i dynamiczne

Określenie API dla pewnej klasy funkcji nie wystarcza do zbudowania programu, który będzie korzystał z funkcji — konieczne dołączenie kodu realizującego opisane funkcje. Zazwyczaj kod implementujący funkcje API ma postać zestawu funkcji zorganizowanych w postaci *biblioteki funkcji*.

W trakcie pisania programu sięgamy zwykle do wielu typowych funkcji, jak np. *sin* czy *cos*, nie zastanawiając się w jaki sposób zostaną one obliczone. Poza funkcjami matematycznymi, w programowaniu korzysta się z procedur wprowadzania i wyprowadzania danych, procedur operujących na plikach, komunikacji między procesami i wielu innych. Wszystkie te funkcje (procedury) zgromadzone są w specjalnych plikach zwanych bibliotekami. Każda biblioteka zawiera zestaw tematycznie związanych ze sobą podprogramów, np. biblioteka podprogramów graficznych, matematycznych, itd. Można więc powiedzieć, że biblioteka stanowi zbiór gotowych, przetestowanych i udokumentowanych podprogramów (procedur), które każdy programista może wykorzystać w swoich programach.

Najczęściej podprogramy biblioteczne przechowywane są w postaci programów półskompilowanych, co bardzo ułatwia ich dołączanie i przyśpiesza proces translacji programu. Programy biblioteczne w postaci źródłowej, np. zakodowane w języku Fortran, spotyka się znacznie rzadziej.

Dołączanie podprogramów bibliotecznych do programu wykonuje się dwoma sposobami. Pierwszy z nich polega na włączeniu podprogramów bibliotecznych do programu już w etapie konsolidacji (linkowania) programu. Wówczas wytworzony plik wynikowy (w systemie MS Windows z rozszerzeniem .EXE) zawiera nie tylko przetłumaczone instrukcje napisane przez programistę, ale także potrzebne podprogramy biblioteczne.

Dołączane w ten sposób biblioteki określa się jako *biblioteki statyczne*. Istnieje też drugi sposób dołączania, w którym podprogramy biblioteczne ładowane są do pamięci dopiero w chwili uruchomienia programu (a nawet później, gdy dany podprogram jest potrzebny do obliczeń). W takim przypadku mówimy o *bibliotekach dynamicznych*. Pliki w systemie Windows, w których umieszczone są biblioteki dynamiczne mają rozszerzenie .DLL (niekiedy .EXE).

Zarówno biblioteki statyczne jak i dynamiczne mogą być samodzielnie tworzone przez programistów, i są dostępne w wielu systemach operacyjnych, w tym w systemie MS Windows i w systemie Linux.

Tworzenie biblioteki dynamicznej w środowisku MS Visual Studio

Powróćmy do omawianego wcześniej przykładowego API, w którym zdefiniowano funkcje `PodajRozmiarObszaruNiezajetego` i `PodajRozmiarObszaruZajetego`. Poniżej podano kod tych funkcji w języku C, w wersji przystosowanej do włączenia w skład biblioteki dynamicznej (.DLL).

```
#include <windows.h>
#include <stdlib.h>

BOOL  WINAPI  DllEntryPoint (HINSTANCE  hinstDLL ,
              DWORD  fdwReason ,      LPVOID  lpvReserved)
{
  return(TRUE);
}

//=========================================================

unsigned int __declspec(dllexport) _stdcall
          PodajRozmiarObszaruNiezajetego(char symbol_dysku)
{
     unsigned __int64 l_dostepnych;
     BOOL odp;
     unsigned int wynik;
     char dysk[4] = " :\\";

     if (!isalpha(symbol_dysku)) return 0xFFFFFFFF;
     // blad, jesli symbol dysku nie jest litera

     dysk[0] = symbol_dysku;

     odp = GetDiskFreeSpaceEx((LPCSTR)dysk,
          (PULARGE_INTEGER)&l_dostepnych, NULL, NULL);
     if (!odp) return 0xFFFFFFFF;
     // blad, jesli nie mozna ustalic rozmiaru

     // wyznaczenie liczby megabajtów - dzielenie przez 2^20
     wynik = (unsigned int)(l_dostepnych >> 20);
     return (unsigned int) wynik;
}

//=========================================================

unsigned int __declspec(dllexport) _stdcall
          PodajRozmiarObszaruZajetego (char symbol_dysku)
{
  unsigned __int64 l_pojemnosc, l_wolnych, l_zajetych;
  unsigned int wynik;
```

```
  BOOL odp;
  char dysk[4] = " :\\";

  if (!isalpha(symbol_dysku)) return 0xFFFFFFFF;
  // blad, jesli symbol dysku nie jest litera

  dysk[0] = symbol_dysku;

  odp = GetDiskFreeSpaceEx((LPCTSTR)dysk, NULL,
  (PULARGE_INTEGER)&l_pojemnosc, (PULARGE_INTEGER)&l_wolnych);
  if (!odp) return 0xFFFFFFFF;
  // blad, jesli nie mozna ustalic rozmiaru

  l_zajetych = l_pojemnosc - l_wolnych;

  wynik = (unsigned int)(l_zajetych >> 20);
  // wyznaczenie liczby megabajtów - dzielenie przez 2^20

  return (unsigned int) wynik;
}
```

Podprogramy (funkcje) wchodzące w skład biblioteki DLL nie różnią się istotnie od innych funkcji bibliotecznych. Funkcje te stosują standard przekazywania parametrów `_stdcall`, a także wymagają podania atrybutu eksportowalności w postaci `__declspec(dllexport)` (uwaga: dwa znaki podkreślenia).

Ponadto, w każdej bibliotece DLL obok właściwych funkcji musi być także zdefiniowana funkcja `DllEntryPoint`. Funkcja ta umożliwia programiście przeprowadzenie inicjalizacji funkcji bibliotecznych. Funkcja `DllEntryPoint` jest wywoływana przez system Windows bezpośrednio po załadowaniu biblioteki do pamięci, przed jej usunięciem z pamięci, a także przy dołączaniu i odłączaniu procesu, jak również przy tworzeniu i kasowaniu wątku. Prototyp omawianej funkcji ma postać:

```
BOOL  WINAPI  DllEntryPoint (HINSTANCE  hinstDLL,
                  DWORD  fdwReason ,  LPVOID  lpvReserved) ;
```

gdzie

`hinstDLL`	uchwyt modułu DLL — może być używany przy wywoływaniu innych funkcji;
`fdwReason`	kod opisujący przyczynę wywołania funkcji, np. DLL_PROCESS_ATTACH oznacza, że biblioteka DLL została dołączona do przestrzeni adresowej bieżącego procesu jako rezultat uruchomienia procesu lub wskutek wykonania funkcji `LoadLibrary`; podobnie, kod DLL_PROCESS_DETACH oznacza, że biblioteka DLL zostaje odłączona od przestrzeni adresowej procesu wskutek jego zakończenia lub wykonania funkcji `FreeLibrary`; zdefiniowane są także kody związane z operacjami na wątkach;
`lpvReserved`	parametr opisuje dalsze cechy procesu dołączania i odłączania biblioteki, np. przyjmuje wartość NULL w przypadku, poprzedni parametr przyjmuje wartość DLL_PROCESS_ATTACH i biblioteka ładowana jest dynamicznie;

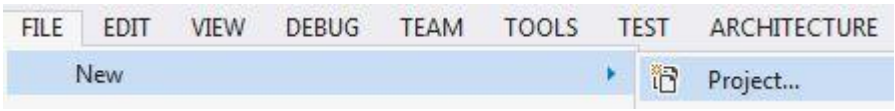
Funkcja `DllEntryPoint` nie musi udzielać odpowiedzi na wszystkie możliwe wartości parametru `fdwReason` — dla niektórych może podejmować działania, inne zaś może ignorować. W wielu przypadkach czynności realizowane przez funkcję `DllEntryPoint` ograniczają się tylko do przekazania wartości TRUE lub FALSE za pomocą instrukcji `return`, a parametry funkcji są ignorowane.

Plik biblioteczny powinien zaczynać się od wiersza `#include <windows.h>` Następnie umieszcza się funkcję `DllEntryPoint`, a za nią właściwe funkcje biblioteczne. W najprostszym przypadku funkcja `DllEntryPoint` może mieć postać:

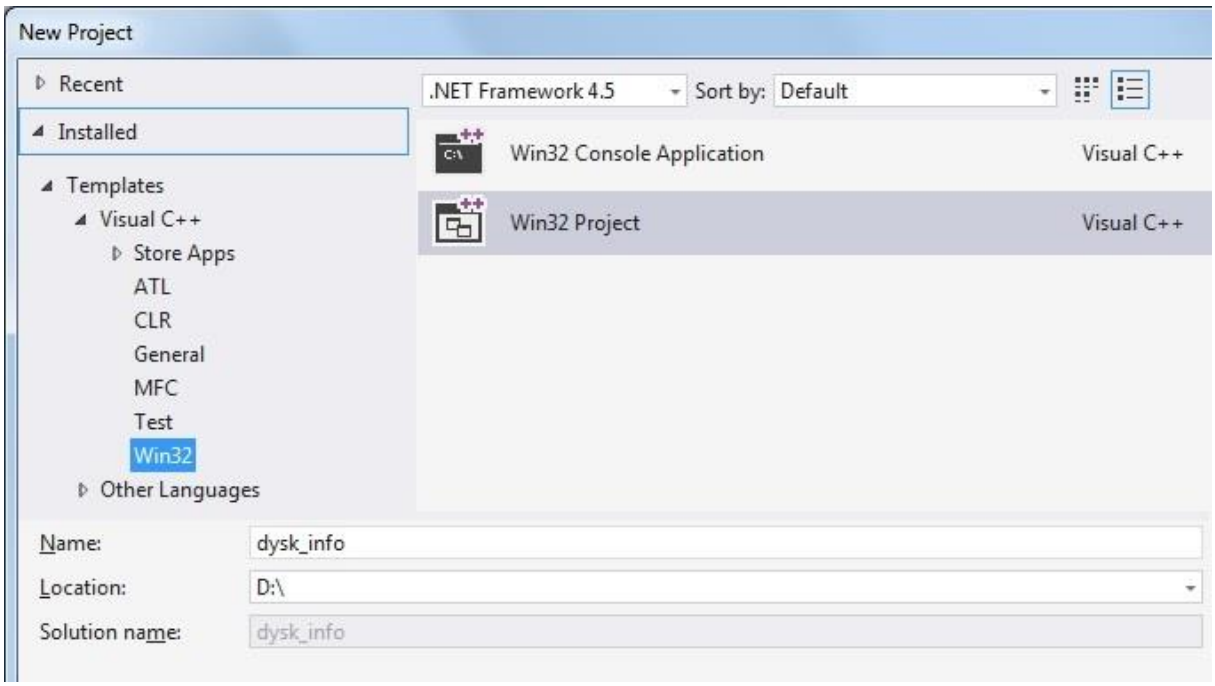
```
BOOL  WINAPI  DllEntryPoint (HINSTANCE  hinstDLL ,
                    DWORD  fdwReason, LPVOID  lpvReserved)
{
  return(TRUE);
}
```

W celu utworzenia biblioteki dynamicznej w środowisku MS Visual Studio 2013 konieczne jest wykonanie następujących czynności:

1. Po uruchomieniu MS Visual Studio należy wybrać opcje: File / New / Project

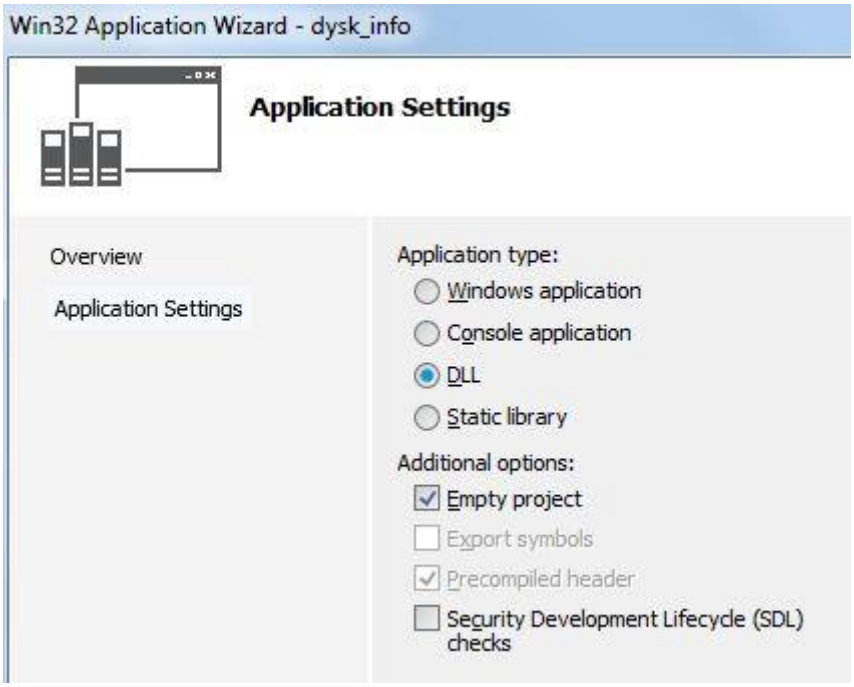


2. W oknie nowego projektu (zob. rys.) określamy najpierw typ projektu poprzez rozwinięcie opcji Visual C++. Następnie wybieramy opcje Win32 / Win32 Project. Do pola Name wpisujemy nazwę biblioteki (tu: dysk_info) i naciskamy OK. W polu Location powinna znajdować się ścieżka D:\. Znacznik Create directory for solution należy ustawić w stanie nieaktywnym.

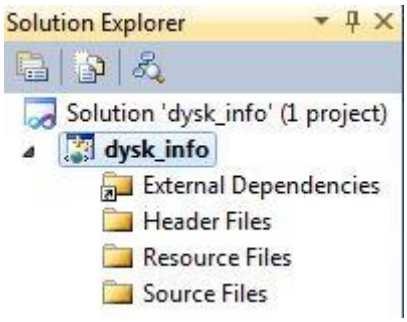


3. W rezultacie wykonania opisanych wyżej operacji pojawi się niżej pokazane okno Win32 Application Wizard

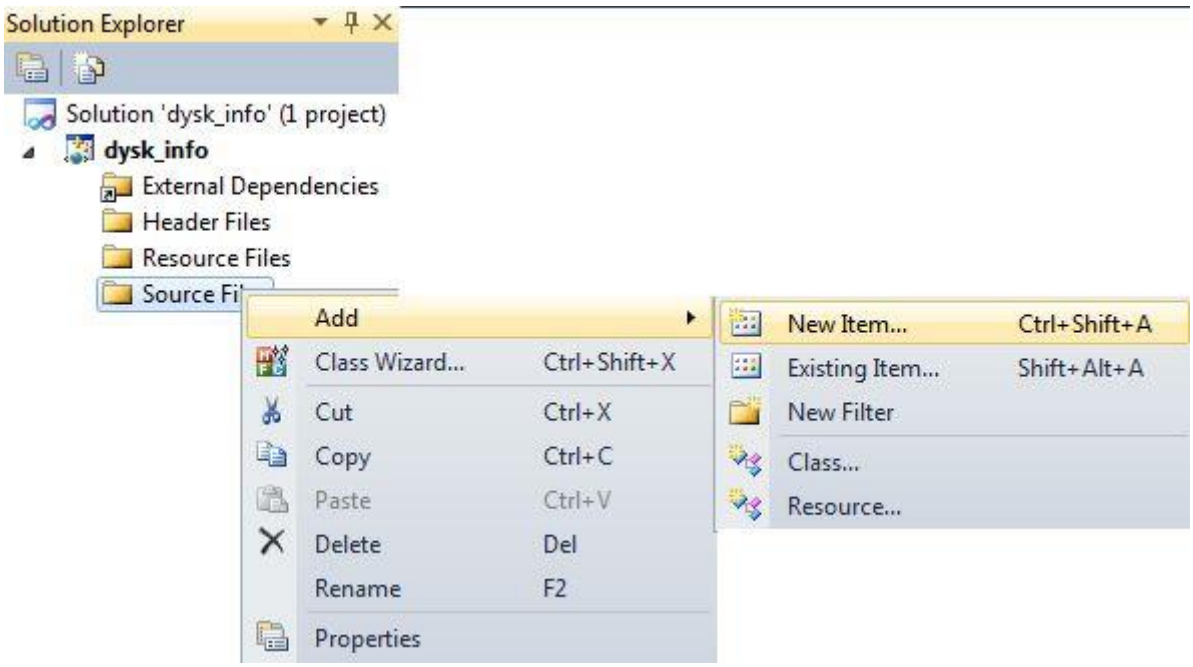
4. Następnie naciskamy przycisk Next i wybieramy typ aplikacji DLL oraz Empty Project.



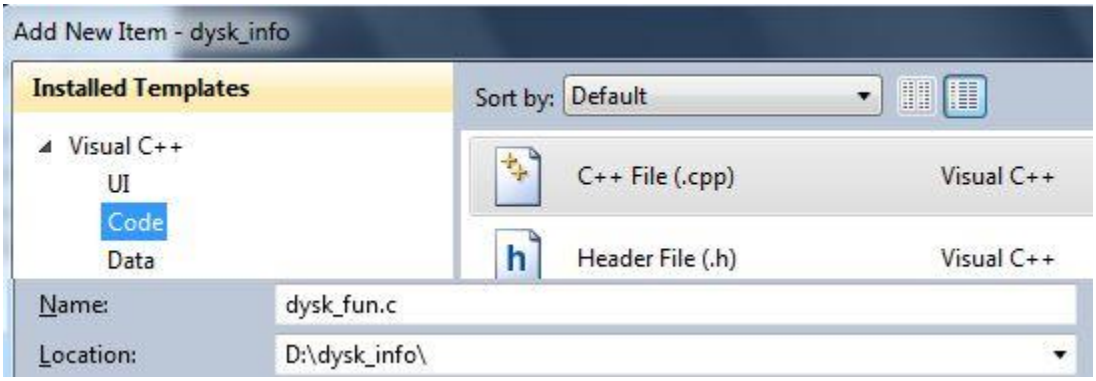
5. Po naciśnięciu przycisku Finish pojawi się okno, którego fragment pokazany jest poniżej.



6. Z kolei należy prawym klawiszem myszki nacisnąć na pozycję Source File i wybrać Add / New Item.

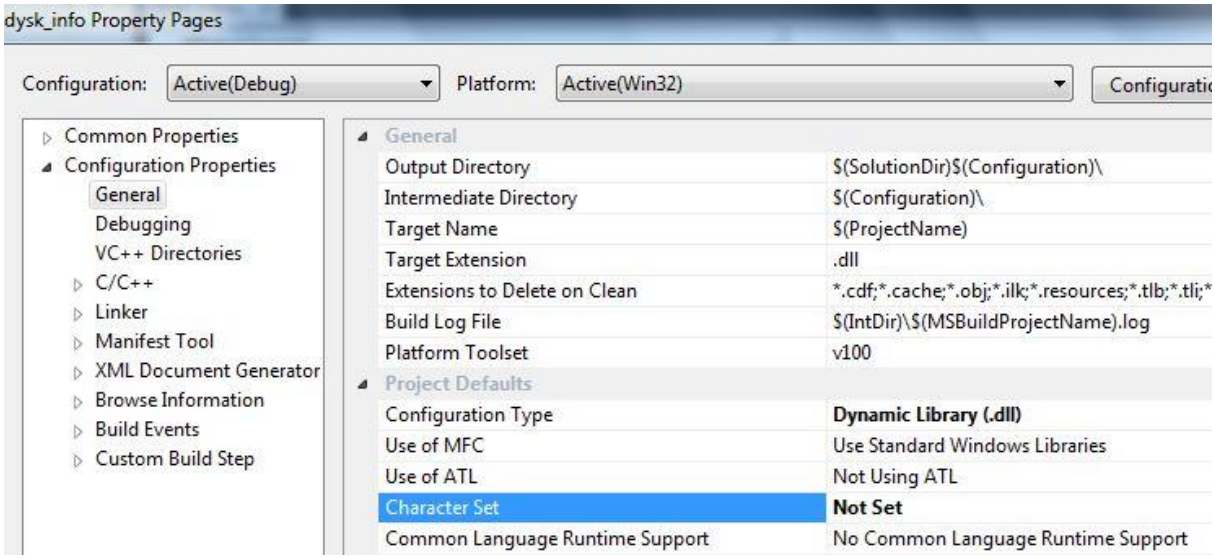


7. W rezultacie pojawi się okno pokazane na poniższym rysunku. W lewej części okna należy wybrać opcję Code, a do pola Name należy wpisać nazwę pliku zawierającego kody funkcji bibliotecznych (tu: dysk_fun.c).



8. Po naciśnięciu przycisku Add pojawi się puste okno, do którego należy wpisać kody funkcji bibliotecznych (str. 7-8) i nacisnąć kombinację klawiszy Ctrl S (Save dysk_fun.c).

9. Następnie trzeba skonfigurować opcje kompilatora. W tym celu należy kliknąć prawym klawiszem myszki w oknie Solution Explorer odszukać pozycję dysk_info i wybrać opcję Property. W tym momencie na ekranie pojawi się okno disk_info Property Pages. W lewej części okna należy należy wybrać pozycję General, a nastepnie w prawej części okna w wierszu Character Set ustawić Not Set i nacisnąć OK. Ilustruje to poniższy rysunek.



10. W celu utworzenia biblioteki DLL wystarczy wybrać opcję Build / Build Solution (klawisz F7 lub inny w zależności od konfiguracji). Biblioteka dynamiczna zostanie umieszczona w pliku `dysk_info.dll`

Wykorzystanie funkcji zawartych w bibliotece dynamicznej

Funkcje wchodzące w skład biblioteki dynamicznej mogą być łatwo wywoływane z poziomu zwykłej aplikacji w języku C lub C++. Poniżej podano program przykładowy w języku C, który korzysta z funkcji zawartych w opisanej wyżej bibliotece DLL.

```
#include <stdio.h>
#include "dysk.h"

int main()
{
  printf("\nObszar niezajety = %d [MB]\n",
                          PodajRozmiarObszaruNiezajetego('D'));
  printf("\nObszar zajety    = %d [MB]\n",
                          PodajRozmiarObszaruZajetego('D'));
  return 0;
}
```

Z punktu widzenia kompilatora funkcje zawarte w bibliotece dynamicznej są zupełnie nieznane — kompilator nie zna liczby i typów argumentów funkcji, a także nie zna typu

wartości zwracanej przez funkcji. Z tego powodu na początku pliku źródłowego umieszcza się pliki nagłówkowe (z rozszerzeniem `.h`), w których zawarte są prototypy funkcji. Zazwyczaj dla każdego API udostępniany jest przez odpowiedni plik nagłówkowy. Dla omawianego tu API utworzono plik nagłówkowy `dysk.h`, zawierający poniższe wiersze:

```
extern  unsigned  int  _stdcall
     PodajRozmiarObszaruNiezajetego (char  symbol_dysku);

extern  unsigned  int  _stdcall
     PodajRozmiarObszaruZajetego (char  symbol_dysku);
```

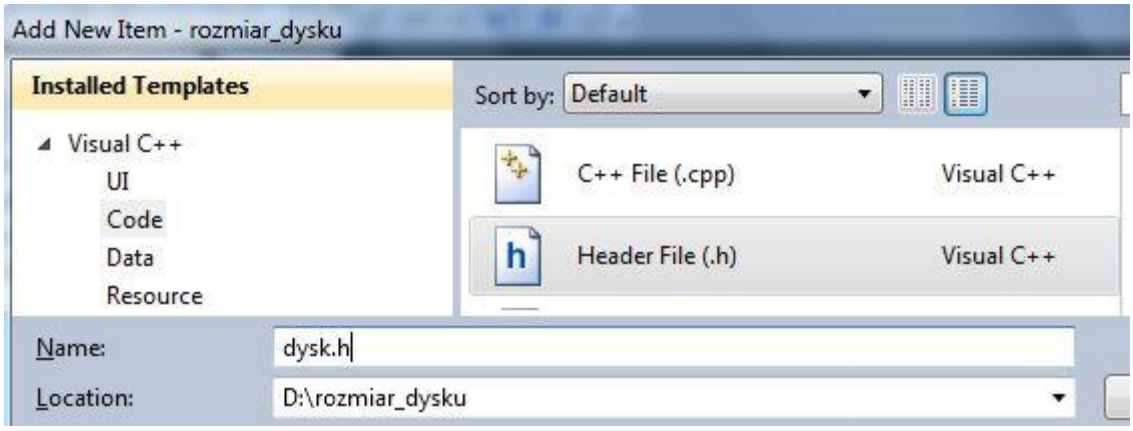
Z kolei w trakcie konsolidacji (linkowania) należy udostępnić informacje o zawartości biblioteki dynamicznej, nie przekazując jednak kodu tych funkcji. Rolę takiej zastępczej biblioteki, w której zawarte są jedynie informacje o funkcjach bibliotecznych (ale bez kodu) pełni tzw. *biblioteka importu* (ang. import library). W podanym przykładzie biblioteka ta zawarta jest w pliku `dysk_info.lib`

Istnieje też możliwość (nie używana w podanym przykładzie) wyznaczania adresu funkcji bibliotecznej dopiero w trakcie wykonywania. W tym przypadku aplikacja nie zawiera jawnych odwołań do funkcji API, ale jedynie nazwy funkcji kodowane w formie łańcuchów ASCII. Za pomocą dostępnych funkcji można przywołać potrzebną bibliotekę i wyznaczyć adresy używanych funkcji.

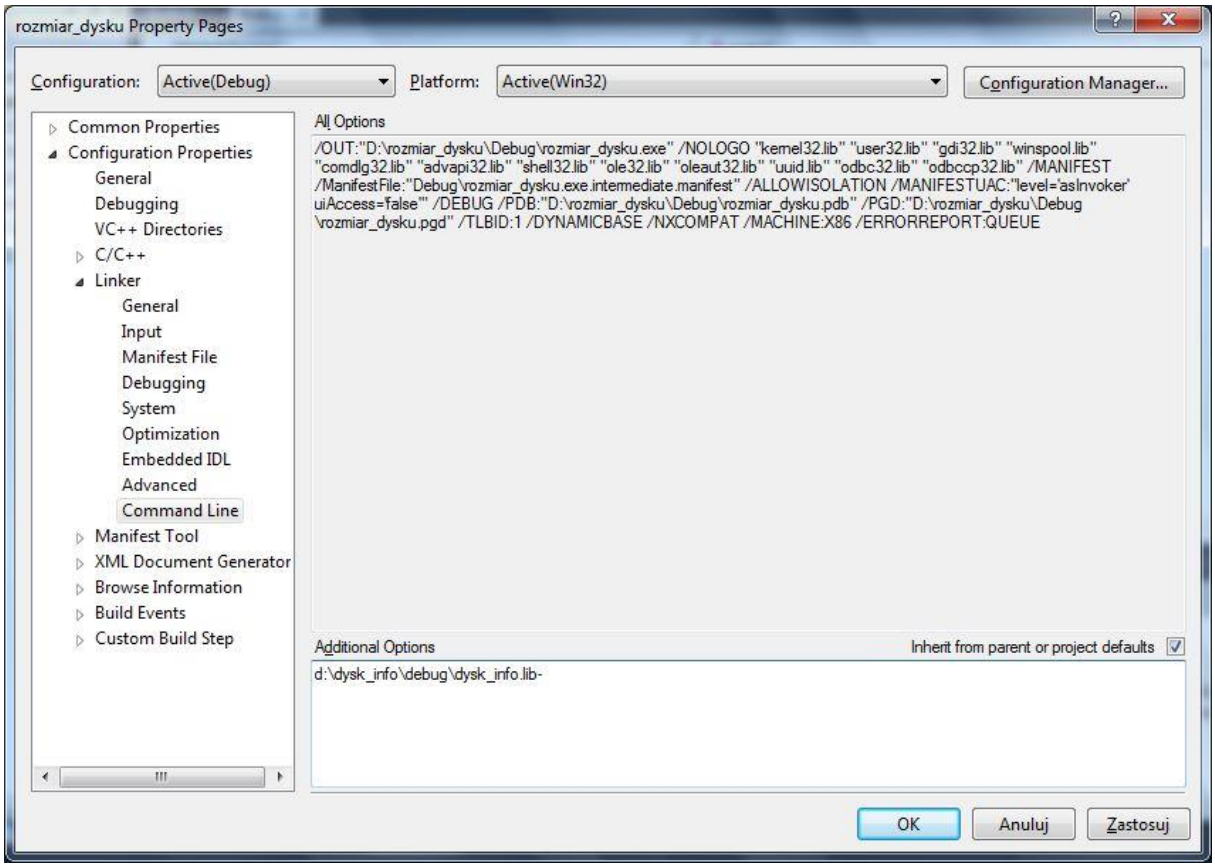
Tworzenie aplikacji przykładowej w środowisku Visual Studio 2013

Aplikację przykładową w języku C skonstruujemy w sposób konwencjonalny. Tak jak poprzednio tworzymy nowy projekt wybierając kolejno File / New /Project, a w oknie nowego projektu wybieramy Visual C++ / General / Empty Project. W polu Name wpisujemy nazwę projektu, np. rozmiar_dysku. Następnie do projektu (grupa Source Files) dołączamy plik `dysk.c`, do którego wprowadzamy kod podany na stronie 12. Naciskamy kombinację klawiszy Ctrl S.

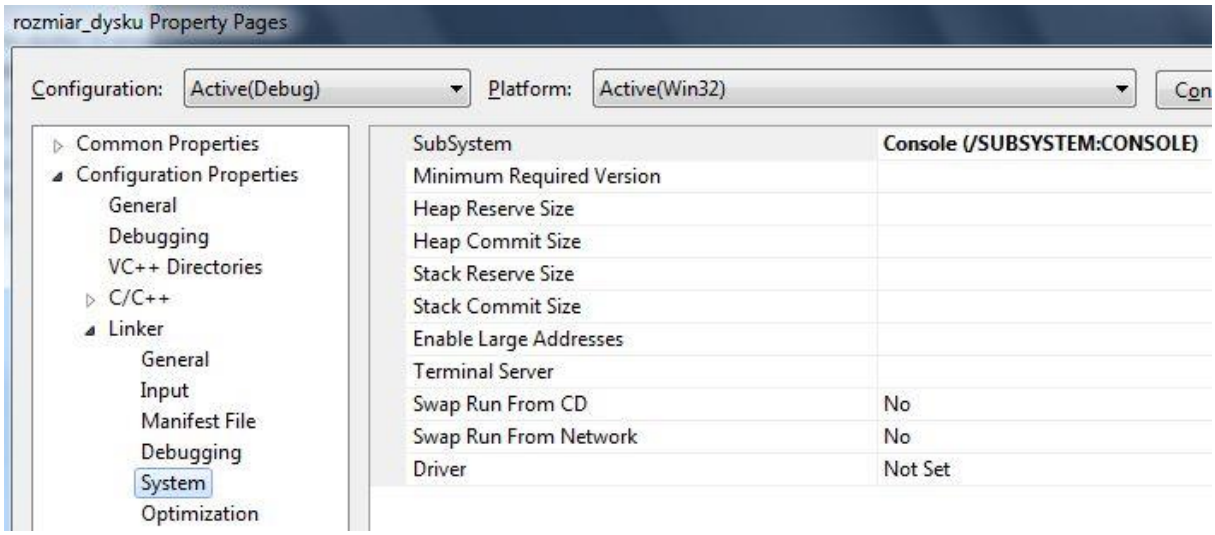
Do projektu dołączymy także plik nagłówkowy `dysk.h`, który omawiany jest na poprzedniej stronie. W tym celu należy kliknąć prawym klawiszem myszy na pozycję Header Files i wybrać Add / New Item. W rezultacie pojawi się okno pokazane na rysunku na następnej stronie. W oknie tym wybieramy Visual C++ / Code / Header File (.h). W polu Name wpisujemy nazwę pliku nagłówkowego (tu: `dysk.h`). Do okna `dysk.h` należy skopiować nagłówki funkcji podane na stronie 13 i nacisnąć Ctrl S.



Ponieważ kody funkcji wywoływanych w programie umieszczone są w bibliotece, konieczne jest przekazanie konsolidatorowi (linkerowi) odpowiednich informacji. Sposób dołączenia biblioteki importu ilustruje poniższy rysunek. W szczególności, w oknie Solution Explorer prawym klawiszem myszki wskazać nazwę projektu, następnie kliknąć na pozycję Properties, co spowoduje pojawienie się ekranie okna dialogowego. W oknie tym, po wybraniu pozycji Linker / Command Line, w polu Additional options należy wpisać nazwę biblioteki importu (tu: d:\dysk_info\debug\dysk_info.lib). Nacisnąć OK.



W ramach pozycji Linker należy ustawić także typ aplikacji. W tym celu zaznaczamy grupę System (zob. rysunek na następnej stronie) i w polu SubSystem wybieramy opcję Console (/SUBSYSTEM:CONSOLE).



Po wykonaniu kompilacji i konsolidacji (linkowania) (klawisz F7) trzeba skopiować bibliotekę dynamiczną `dysk_info.dll` (znajdującą się w katalogu `d:\dysk_info\debug`) do katalogu `d:\rozmiar_dysku`.

Zadania do wykonania

1. Utworzyć bibliotekę dynamiczną (DLL) zawierającą opisane wcześniej funkcje biblioteczne `DllEntryPoint`, `PodajRozmiarObszaruNiezajetego` i `PodajRozmiarObszaruZajetego`.

2. Sprawdzić działanie funkcji bibliotecznych za pomocą programu przykładowego, który korzysta z tych funkcji.

3. Opracować API dla zestawu funkcji obliczających pola i objętości w formie dokumentu tekstowego (za pomocą edytora *Word* lub *notatnika*). API powinno być zredagowane w języku polskim w formie zbliżonej do podanej w początkowej części instrukcji i zawierać opisy kilku funkcji. Należy wybrać jeden zestaw funkcji spośród niżej wymienionych:
   a. funkcje obliczające pola powierzchni figur geometrycznych
   b. funkcje obliczające pola powierzchni bocznej brył
   c. funkcje obliczające objętości brył

   Argumenty i wartości funkcji powinny być typu *float* lub *double*.

4. Opracować i uruchomić bibliotekę dynamiczną (DLL), w której zostaną umieszczone funkcje implementujące ww. API. Kod funkcji napisać w języku C (ewentualnie w asemblerze). Opracować także odpowiedni plik nagłówkowy (z rozszerzeniem .h), w którym umieszczone będą prototypy opracowanych funkcji.

5. Opracować i uruchomić program przykładowy w języku C ilustrujący sposób wywoływania funkcji z opracowanego API.

6. W programie przykładowym (pkt. 5) może pojawić się konieczność wprowadzania danych z klawiatury. W programie w języku C używa się do tego celu funkcji

scanf_s, która zastąpiła używaną dawniej funkcję scanf. Przykładowo, jeśli wprowadza się ciąg znaków, którego długość nie przekracza 15, wywołanie funkcji scanf_s może mieć postać:

```
char napis[16] ;
- - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - -
scanf_s ("%15s", napis, 16);
```

7. Niekiedy przy wywołaniu funkcji scanf_s stosuje się bardziej złożony format "%[^\n]s" , który powoduje, że funkcja scanf_s kończy czytanie wiersza po napotkaniu znaku nowego wiersza \n (typowy format "%s" powoduje, że scanf_s traktuje spację jako koniec wiersza).

8. Zarówno w funkcji scanf_s, jak też w funkcji printf (która używana jest do wyświetlania wyników), stosowane są specyfikatory formatu — poniżej podano najczęściej używane:

%s wczytanie lub wyświetlanie łańcucha znaków,
%d wczytanie lub wyświetlanie wartości typu int (liczba całkowita ze znakiem),
%u wczytanie lub wyświetlanie wartości typu unsigned int (liczba całkowita bez znaku),

W przypadku wczytywania liczb typu float stosuje się format %f, a w przypadku liczb typu double — format %lf.

W przypadku wyświetlania wyników za pomocą funkcji printf format %f stosuje się dla liczb typu float i double (format %lf stosuje się do wartości typu long double).