

# Obsługa plików

Rozdział ten poświęcony jest omówieniu zagadnienia pracy na danych zewnętrznych zamieszczonych w plikach. Przykłady zamieszczone w rozdziale pokazują, w jaki sposób realizowana jest praca z danymi zawartymi w plikach w systemach operacyjnych MS Windows i Linux. Zamieszczone komentarze i porównania pozwolą Czytelnikowi przeanalizować, w jaki sposób można wykonać implementację obsługi plików w poszczególnych systemach operacyjnych. Wiadomości zawarte w tym rozdziale mają również ogromne znaczenie praktyczne, ponieważ trudno jest realizować zadanie programistyczne, które nie wspierałoby pracy na danych zewnętrznych zawartych w plikach. Do podstawowych operacji w każdym systemie operacyjnym należą operacje otwarcia i zamknięcia plików. Ogromne znaczenie mają również możliwość odczytania danych zawartych w pliku czy zapis wartości, jakie wyznacza program do pliku. Ten rozdział ma na celu pokazać

Czytelnikowi, w jaki sposób można realizować takie zadania w systemie MS Windows czy Linux. Informacje zostały opisane w przykładach praktycznych, które ułatwią zrozumienie zasad, jakimi należy się kierować podczas pracy z plikami. Pokazano atrybuty plików oraz różne funkcje wykorzystujące możliwości współpracy z plikami zewnętrznymi.

Rozdział ten omawia również kolejny ważny element warsztatu programistycznego. Przedstawione metody tworzenia struktur danych i pracy z nimi poprzez różnego rodzaju funkcje własne i zbudowane przez użytkownika pozwolą zrozumieć zasady, jakimi należy się kierować implementując rozwiązania problemów podejmowania decyzji

Podejście takie ma za zadanie pokazać Czytelnikowi, w jaki sposób można w praktyce implementować rozumienie informacji złożonej z kilku wartości zawartych w źródłach zewnętrznych. Przykłady praktyczne operujące na strukturach pokazują rangę powiązań pomiędzy wartościami, które złożone w odpowiednią całość pozwalają dokładniej opisać zadanie. Podejście takie ma na celu wprowadzić Czytelnika w obsługę zbiorów informacji, czyli baz danych. Przykłady praktyczne pokażą, w jaki sposób można za pomocą plików z danymi oraz wykorzystania wiedzy o strukturach i tablicach stworzyć procedurę pozwalającą w sposób efektywny przeszukiwać posiadaną bazę wiedzy. Podobne podejście do struktur danych pokazano w książkach [21, 39, 45-46, 65, 67, 102]. Natomiast w książkach [28, 73, 91, 115] można znaleźć dokładniejsze omówienie struktur informacji.

# Obsługa zbiorów – plików

Programując dowolny kod rozwiązujący zadany problem oczekujemy od komputera określonych wyników. Niezależnie do tego czy są to wyniki liczbowe czy tekstowe bardzo często zdarza się, że jest to tylko rozwiązanie przechodnie danego problemu i wyniki te będziemy przekazywać do dalszego rozpoznania. W takim przypadku, jak i każdym innym, najlepiej jest mieć wyniki zapamiętane w sposób możliwy do bezpośredniej archiwizacji. Rozwiązaniem problemu wydaje się zapis wartości do wybranego przez nas pliku. Kolejnym problemem jest również możliwość automatycznego wprowadzania danych zapamiętanych wcześniej. Najrozsądniejszym pomysłem wydaje się tutaj zastosowanie procedury odczytującej dane z plików. W komputerze najczęściej operujemy na różnego rodzaju plikach. Czytelnik zapewne najbardziej kojarzy pliki wykonywalne i tekstowe. Spróbujmy prześledzić sposób i możliwości, jakie daje język C w kwestii zapisu i odczytu danych. Jak zapewne

Czytelnik przypuszcza posłużymy się funkcjami dostępnymi w odpowiedniej bibliotece języka C. Operacja zapisu i odczytu danych pliku odbywa się poprzez funkcje, które pobierają i zwracają dane oraz zbiory (bufory), które przechowują i przekazują zawarte w plikach dane. Biblioteka *stdio.h* zawiera funkcje *getc()*, *putc()*, *read()* i *write()*, które odnoszą się bezpośrednio do urządzeń wejścia – wyjścia. W języku C są zdefiniowane trzy następujące zbiory oraz stała:

- *stdin* - standardowe wejście (klawiatura),
- *stdout* - standardowe wyjście (monitor),
- *stderr* - obsługa błędów,
- *NULL* – stała, która wskazuje na pusty ciąg znaków.

Dane, jakie zostały pobrane do odpowiednich zbiorów we (wejścia) – wy (wyjścia) są przekształcane przez funkcje, aby mogły zostać przekazane do standardowego wejścia czy wyjścia systemu. Funkcje formatujące przekształcające dane ze zbiorów *stdin* i do *stdout* mają następującą postać.

```
scanf (" Pole kontrolne " , argument1 , argument2 , ... )  
printf (" Pole kontrolne " , argument1 , argument2 , ... )
```

Pole kontrolne określające sposób wprowadzania argumentów można wypełnić w następujący sposób.

%c	Dla pojedynczego znaku.
%s	Dla ciąg znaków.
%d	Dla liczb typu int ( integer ).
%x	Dla liczb szesnastkowych.
%lf	Dla zmiennych typu double.
%le	Dla liczb zapisanych w postaci wykładniczej.
%g	Dla wydruku liczb rzeczywistych – najkrótsza postać.
%%	Znak % .
\"	Znak " .
\n	Znak nowej linii.



# Przykład 30

Napisać program wczytujący ze standardowego wejścia liczbę całkowitą, liczbę rzeczywistą w postaci wykładniczej i ciąg znaków niezawierających białego znaku.

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

```
int main()
```

```
{
```

```
    int n;
```

```
    double b;
```

```
    char z[256];
```

```
    scanf("%d,%le,%s", &n, &b, z);
```

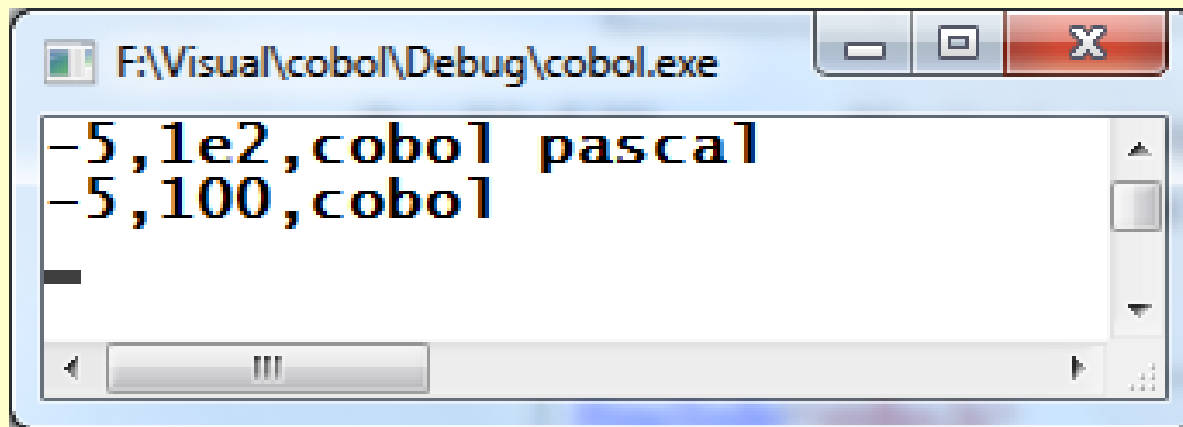
```
    printf("%d,%g,%s\n", n, b, z);
```

```
}
```

```
// Deklaracja biblioteki
```

```
// Deklaracja zmiennych
```

```
// Wczytanie i wypisanie  
wartości
```



```
F:\Visual\cobol\Debug\cobol.exe  
-5,1e2,cobol pascal  
-5,100,cobol
```

Wyniki wykonania programu na konsoli systemowej zostały pokazane na poprzedniej ilustracji. Czytelnik zechce zwrócić uwagę, że podczas czytania dowolnego ciągu znaków wystąpienie białego znaku powoduje przerwanie operacji wczytania do tablicy znaków. Faktycznie zostaje wczytane tylko pierwsze słowo *cobol* drugie pozostaje w buforze. Dlatego do pobrania całej linii tekstu należy użyć funkcji *gets()*, która znak nowej linii zamienia na znak *NULL*.

Dotychczas czytaliśmy dane ze standardowego wejścia i zapisywaliśmy wynik na standardowe wyjście. Prześledźmy teraz dokładnie procedury obsługujące zbiory zewnętrzne, które są zdefiniowane w bibliotece *stdio.h*. W deklaracji w kodzie niezbędne jest podanie wskaźnika do żadanego zbioru (pliku) mające postać.

```
FILE *plik ;
```

Następnie musimy użyć instrukcji *fopen()* otwierającej zbiór w trybie do czytania, zapisu albo dopisywania zależnie od zadanego trybu. Instrukcja otwarcia zbioru ma postać.

```
plik = fopen(nazwa_zbioru, tryb_otwarcia) ;
```

gdzie tryb\_otwarcia przyjmuje jedną z wartości:

"r" – do czytania,

"w" – do zapisu,

"a" – do dopisywania na koniec zbioru.

Podaliśmy tylko podstawowe tryby otwarcia najbardziej potrzebne do obsługi zbiorów. Analogicznie do czytania zbioru służy funkcja *getc(plik)* i do zapisu *putc(znak, plik)*. Po wykonaniu operacji na zbiorach zamykamy je funkcją *fclose(plik)*.

# Przykład 31

Napisać program przepisujący znaki ze zbioru otwartego do odczytu i zapisujący je w drugim zbiorze. Nazwy zbiorów są pobierane z linii poleceń.

```
#include<stdio.h>
int main(int argc,char *argv[])
{   char c;
    FILE *f1, *f2;
    if(argc!=3)
    {
        printf("Niepoprawna ilosc argumentow\n");
        return -1;
    }
    if((f1=fopen(argv[1],"r"))==NULL)
    {
        printf("Nie moge otworzyc pliku %s\n",
                argv[1]);
        return -1;
    }
}
```

```
// Deklaracja biblioteki
```

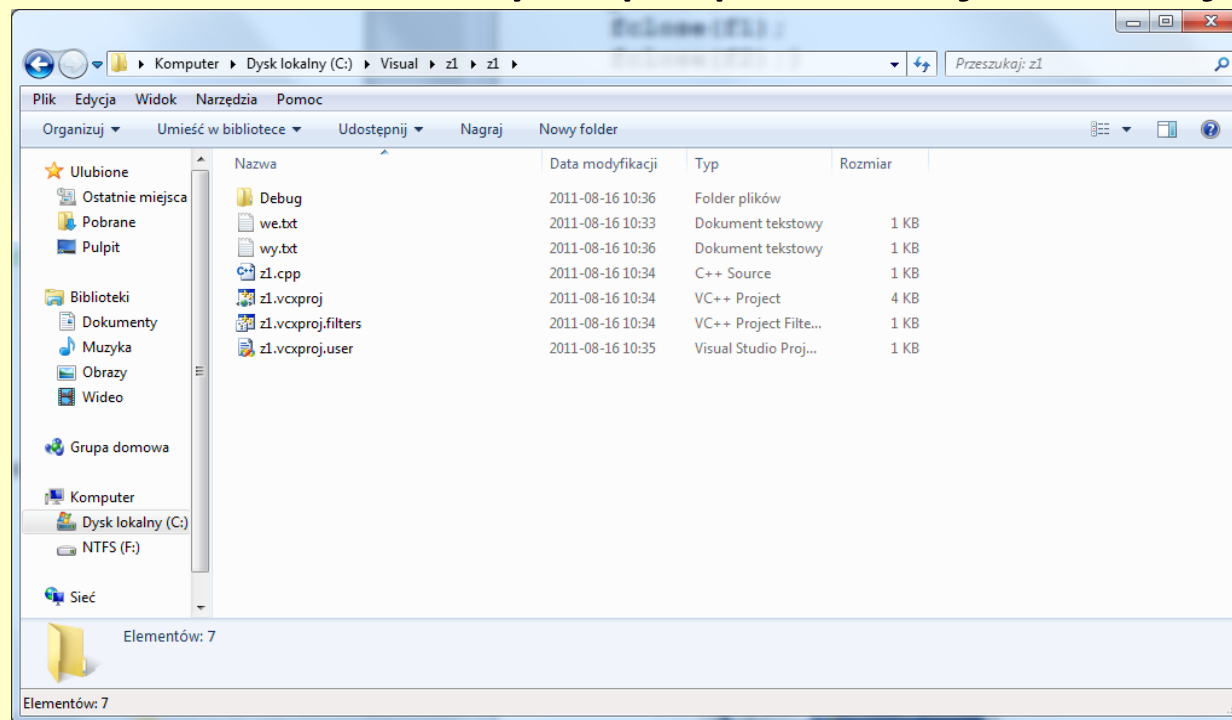
```
/*      Sprawdzenie      ilości
argumentów funkcji głównej
programu, w których zadane są
nazwy plików do odczytu i
zapisu. */
```

```
/*      Próba      otwarcia
zadeklarowanego      zbioru
pierwszego do odczytu.*/
```

<pre>if((f2=fopen(argv[2], "w"))==NULL) {     printf("Nie moge otworzyc pliku %s\n",            argv[2]);     return -1; }  while((c = getc(f1)) != EOF)     putc(c,f2);  fclose(f1); fclose(f2); }</pre>	<pre>/*      Próba      otwarcia zadeklarowanego      zbioru drugiego do zapisu.*/  /* Czytanie wartości znak po znaku ze zbioru pierwszego i zapis do zbioru drugiego. */  /* Zamknięcie zbiorów */</pre>
---	--

Uwaga: Czytelnik zechce zwrócić uwagę na kilka ważnych dla poprawności działania omawianego projektu warunków, omówionych po poniższej ilustracji.

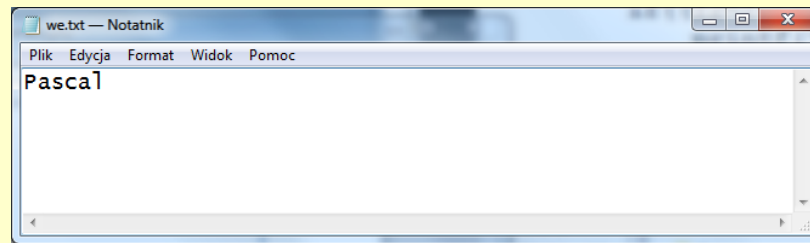
Uwaga: Czytelnik zechce zwrócić uwagę na kilka ważnych dla poprawności działania omawianego projektu warunków, omówionych po poniższej ilustracji.



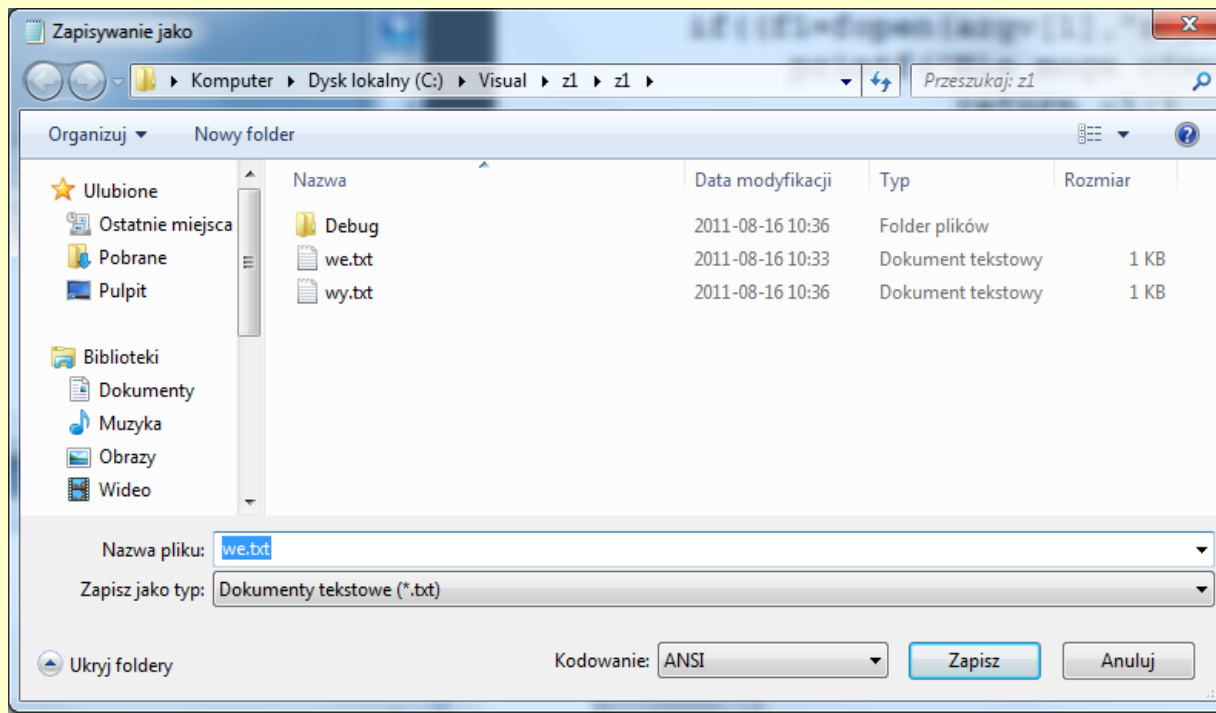
Uruchomienie programu musi się odbyć po umieszczeniu go w katalogu domyślnym naszego projektu. W naszym przypadku projekt został utworzony w katalogu *Visual*, o nazwie projektu *z1*. Stąd przechodzimy do katalogu *C:\Visual\z1\z1*, co pokazuje powyższa ilustracja.

W katalogu, pokazanym na ilustracji, znajdują się pliki tworzone przez MS Visual Studio. Natomiast nie znajdują się w nim jeszcze potrzebne pliki tekstowe. W celu zademonstrowania działania naszego programu utworzymy zbiór tekstowy w postaci pliku za pomocą np. edytora *notatnik*. W tym celu po przejściu do naszego katalogu klikamy prawym przyciskiem myszy i wybieramy prawym przyciskiem myszy opcję z menu *nowy*. Wybieramy *nowy dokument tekstowy*. Następnie zmieniamy jego nazwę na *wy* pozostawiając rozszerzenie *txt*, jest to plik tekstowy służący jako miejsce docelowe kopiowania. W podobny sposób tworzymy dokument, który ma zawierać dane do skopiowania, zmieniamy jego nazwę na *we* pozostawiając rozszerzenie *txt*. Ustawiamy kursor na ikonie pliku i otwieramy go, klikając dwa razy lewym przyciskiem myszy.

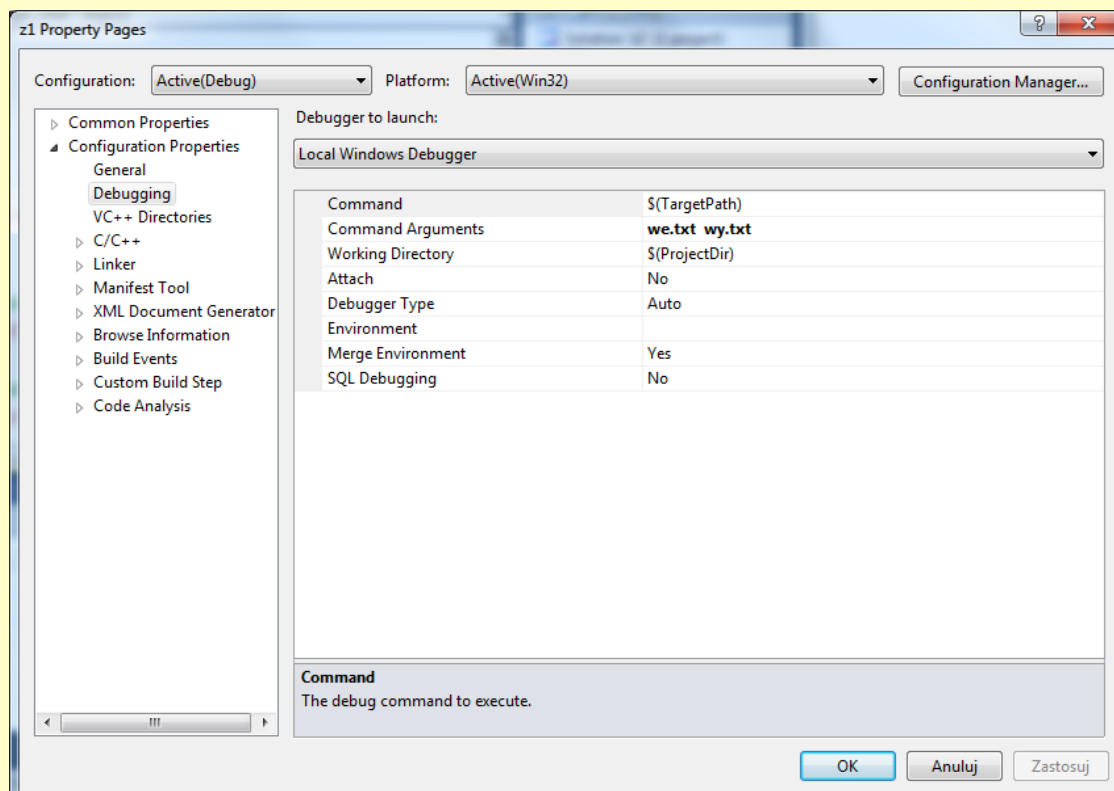




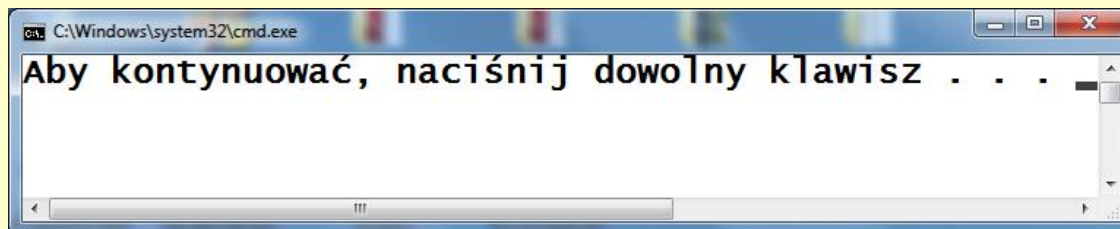
Aby zapisać tekst do pliku wybieramy z menu *zapisz jako...*, a kodowanie pliku ustawiamy na ANSI - jest to domyślne ustawienie, następnie zapisujemy plik. Operacja utworzenia nowego pliku, a następnie zapisania go ze zmienioną nazwą została pokazana na poniższej ilustracji.



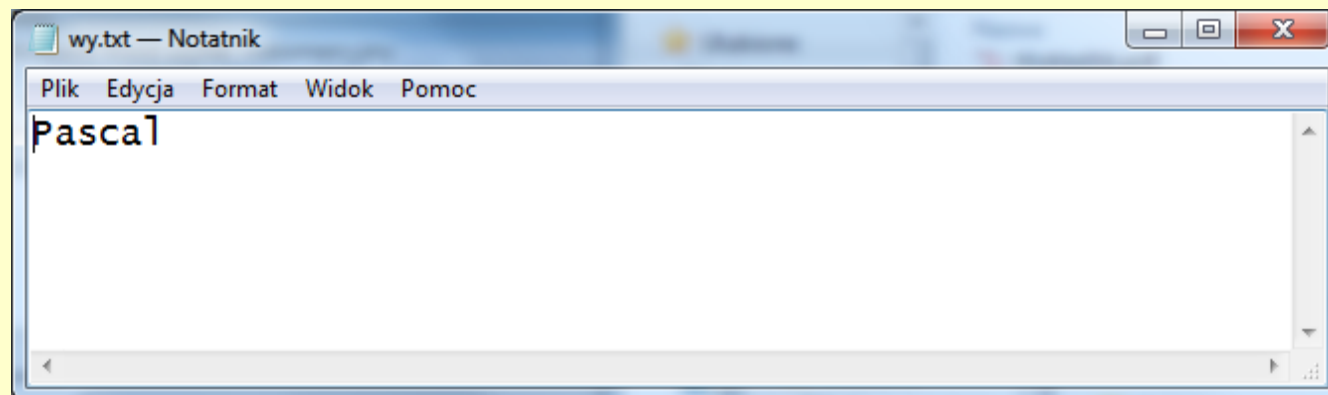
Uwaga: Trzeba pamiętać, że jeżeli nie odznaczaliśmy opcji ukrywania rozszerzenia plików w systemie MS Windows wpisujemy tylko nazwę główną pliku bez rozszerzenia. Pozostaje teraz tylko wpisać nazwy plików w argumentach wywołania programu i uruchomić go. Wykonanie dołączenia plików do argumentów funkcji głównej programu w opcjach MS Visual Studio pokazane zostało na poniższej ilustracji.



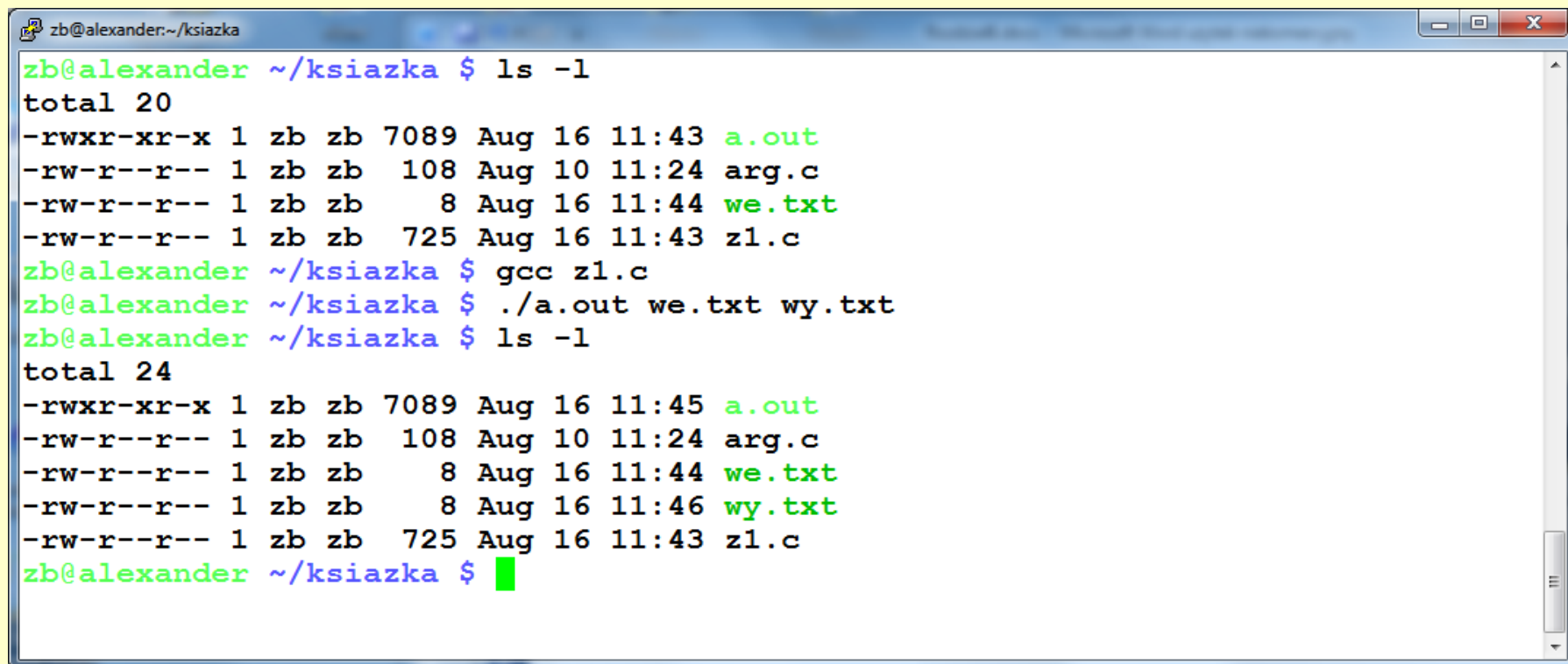
Ponieważ aplikacja nie wyświetla żadnych komunikatów o możliwych błędach na ekran pozostaje zaakceptować jej działanie przyciskając dowolny klawisz.



Następnie przejść do katalogu domyślnego aplikacji i utworzyć utworzony poprzez nasz program plik tekstowy. Jak Czytelnik zechce zauważyć na ilustracji jest on identyczny z plikiem wejściowym *we.txt*.



Kompilacja i uruchomienie programu pod systemem Linux wygląda podobnie. Po napisaniu programu oraz wpisaniu tekstu *Pascal* do pliku *we.txt* za pomocą edytora tekstowego kompilujemy program poleceniem *gcc z1.c* i uruchamiamy *./a.out we.txt wy.txt*. Wykonanie operacji kompilacji programu oraz uruchomienia go z argumentami funkcji głównej zostało pokazane na ilustracji



```
zb@alexander:~/ksiazka $ ls -l
total 20
-rwxr-xr-x 1 zb zb 7089 Aug 16 11:43 a.out
-rw-r--r-- 1 zb zb 108 Aug 10 11:24 arg.c
-rw-r--r-- 1 zb zb 8 Aug 16 11:44 we.txt
-rw-r--r-- 1 zb zb 725 Aug 16 11:43 z1.c
zb@alexander:~/ksiazka $ gcc z1.c
zb@alexander:~/ksiazka $ ./a.out we.txt wy.txt
zb@alexander:~/ksiazka $ ls -l
total 24
-rwxr-xr-x 1 zb zb 7089 Aug 16 11:45 a.out
-rw-r--r-- 1 zb zb 108 Aug 10 11:24 arg.c
-rw-r--r-- 1 zb zb 8 Aug 16 11:44 we.txt
-rw-r--r-- 1 zb zb 8 Aug 16 11:46 wy.txt
-rw-r--r-- 1 zb zb 725 Aug 16 11:43 z1.c
zb@alexander:~/ksiazka $
```

Do czytania i zapisu danych w zbiorze często używane są również odpowiednie funkcje

```
fscanf(plik,"format",argumenty)
```

```
fprintf(plik,"format",argumenty)
```

Umożliwiają one formatowanie pobieranych i zapisywanych danych. Prześledźmy teraz ich specyfikę na przykładzie praktycznym algorytmu sortowania danych w pliku.

# Obsługa plików w MS Visual Studio

W celu zapewnienia bezpieczeństwa systemu operacyjnego w MS Visual Studio dodano funkcje operujące na standardowym wejściu (klawiatura) i standardowym wyjściu (monitor).

```
int scanf_s(const char *format [,argument]... );
```

```
int printf_s(const char *format [,argument]... );
```

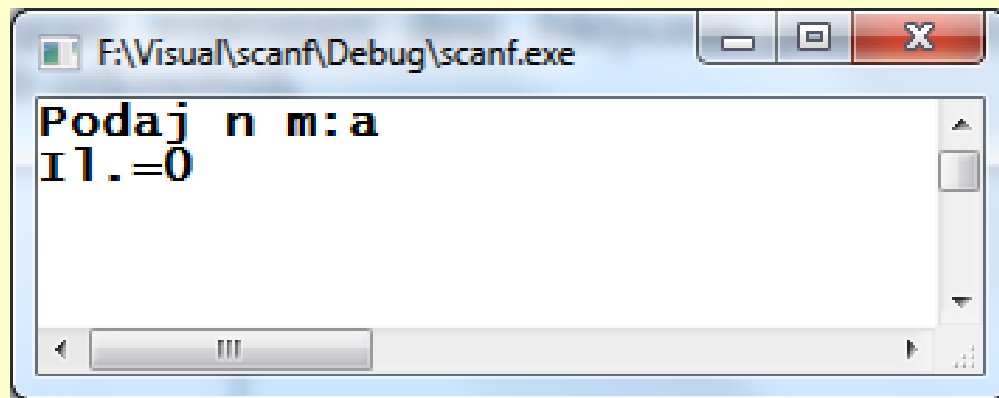
Funkcje te działają analogicznie do uprzednio przedstawionych. Użycie ich jest zalecane dla systemu Windows, ponieważ zapewniają one większe bezpieczeństwo systemu operacyjnego. Prześledźmy teraz ich działanie.

# Przykład 32

Napisać program wypisujący ilość faktycznie przeczytanych liczb podczas wczytywania dwóch liczb całkowitych.

<pre>#include&lt;stdio.h&gt; #include&lt;conio.h&gt;  int main() {     int n, m, wynik;     printf_s("Podaj n m:");     wynik = scanf_s("%d %d", &amp;n, &amp;m);     printf_s("Il.=%d\n", wynik);      _getch(); }</pre>	<pre>// Deklaracja bibliotek  // Wypisanie na ekran. // Pobranie do pamięci.</pre>
---	--

W oknie wpisujemy liczby, a program podaje nam ilość przeczytanych poprawnie liczb. Przykładowo, jeśli podamy znak jak na poniższej ilustracji możemy otrzymać pokazany wynik. Umożliwia nam to faktyczną kontrolę wczytywanych liczb.





Do otwarcia pliku używamy poniższej instrukcji.

```
errno_t fopen_s( FILE** pFile, const char *filename,  
                const char *mode);
```

Pierwszym argumentem funkcji jest wskaźnik do pliku, drugim nazwa pliku, a trzecim tryb otwarcia pliku i sposób kodowania. Domyślnym kodem otwieranego pliku jest kod ANSI (ASCII). Do czytania zbiorów używamy funkcji.

```
int fgetc( FILE *strumien );
```

```
wint_t fgetwc( FILE *strumien );
```

Pierwsza z nich czyta znaki w kodach ASCII druga znaki szerokie zdefiniowane, jako *wchar\_t* . Każdy znak zapisany jest w kodzie UNICODE i pamiętany na dwóch bajtach. Unicode zawiera wszystkie znaki międzynarodowe. Do zapisu znaków służą dwie funkcje.

```
int fputc(int c, FILE *strumien );  
wint_t fputwc(wchar_t c, FILE *strumien );
```

# Przykład 33

Napisać program przepisujący znaki ze zbioru otwartego do odczytu i zapisujący je w drugim zbiorze. Nazwy zbiorów są pobierane z linii poleceń.

```
#include<stdio.h>
#include<conio.h>
int main(int argc,char **argv)
{

    FILE *f1,*f2;
    int c, zam;

    if(argc != 3)
    {
        printf_s("Niepoprawna ilosc argumentow\n");
        _getch();
        return -1;
    }
```

```
// Deklaracja bibliotek
```

```
/*    Sprawdzenie ilości
argumentów programu. */
```

```
if( fopen_s( &f1, argv[1], "r" ) !=0 )
{
    printf( "Zbior %s nie zostal otwarty\n",argv[1]);
    _getch();
    return -1;
}
if( fopen_s( &f2, argv[2], "w" ) != 0 )
{
    printf_s( "Zbior %s nie zostal otwarty\n",argv[2]);
    _getch();
    return -1;
}

while((c=fgetc(f1))!=EOF)
    fputc(c,f2);

zam = _fcloseall( );
printf_s( "Liczba zamknietych plikow: %d\n", zam);
_getch();
return 0;
}
```

/\* Otwarcie pliku do odczytu. \*/

// Otwarcie pliku do zapisu.

/\* Przepisanie plików znak po znaku zapisanym w kodzie ANSI ( ASCII).

Zamknięcie wszystkich zbiorów.

Wydruk ilości zamkniętych zbiorów.\*/\*

Do zamykania wszystkich otwartych plików używamy funkcji:

```
int _fcloseall( void );
```

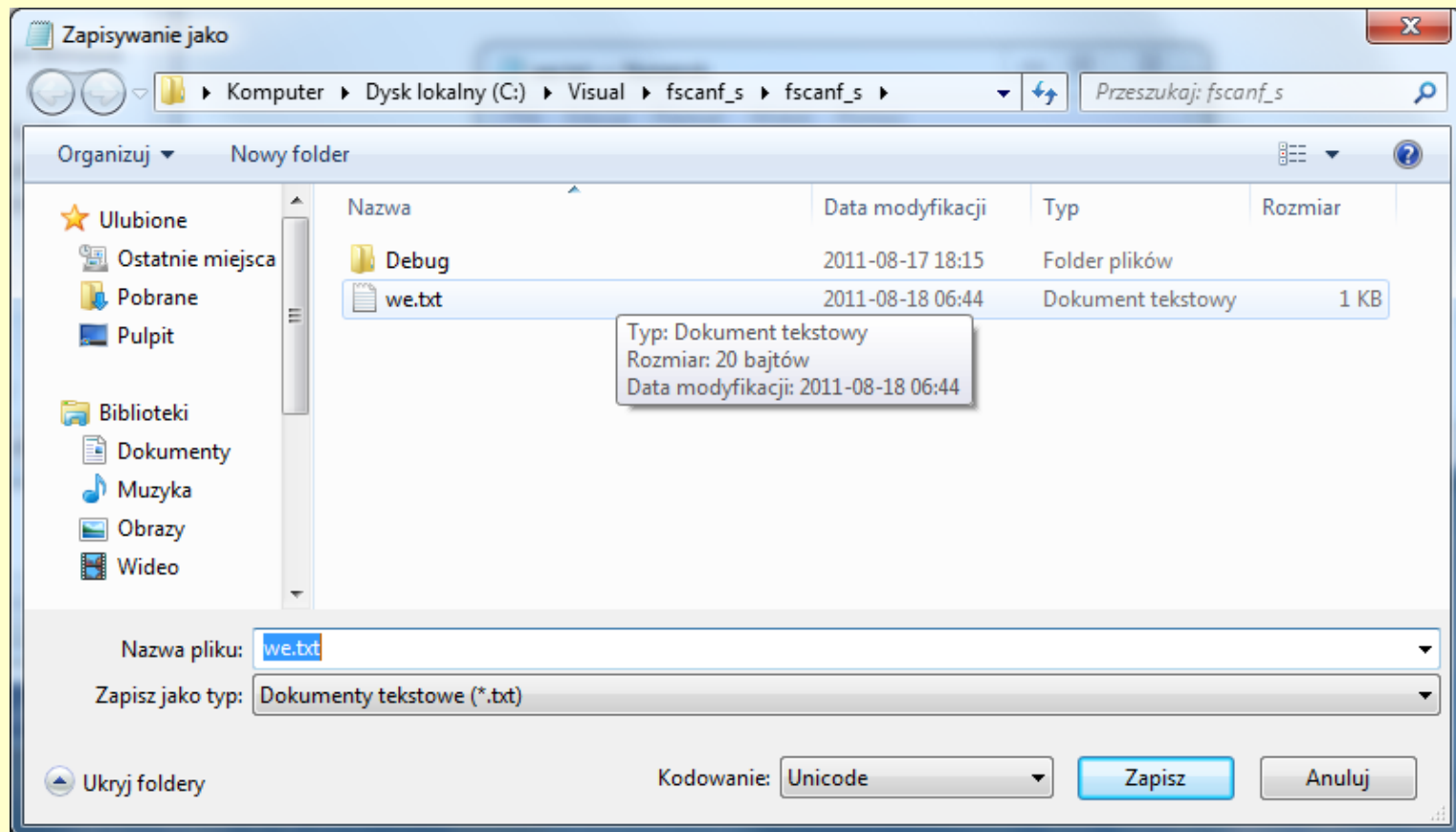
zwracającej ilość zamkniętych plików. Podobnie zdefiniowane są funkcje czytania ze zbioru z formatowaniem argumentów.

```
int fscanf_s( FILE *stream,  
              const char *format [,argument ]... );  
int fwscanf_s( FILE *stream,  
              const wchar_t *format [, argument ]... );
```

# Przykład 34

Napisać program czytający z pliku zapisanego w kodzie UNICODE liczby całkowite i obliczający średnią arytmetyczną przeczytanych liczb. Nazwa pliku zawierającego liczby jest podana z linii poleceń.

Na początku umieszczamy plik *we.txt* zapisany w UNICODE w katalogu projektu. Zwyczajowo jest to katalog projektu i podkatalog o tej samej nazwie zawierający program projektu. Za pomocą notatnika stworzymy plik *we.txt* zawierający liczby całkowite. Dla uproszczenia zapisujemy każdą liczbę w jednej linii. Ustawiamy kodowanie na UNICODE i wybieramy zapis, co pokazano na kolejnej ilustracji.



Następnie umieszczamy kod programu w naszym projekcie:

```
#include<stdio.h>
#include<wchar.h>
#include<conio.h>
int main(int argc,char **argv)
{
    FILE *f;
    int n = 0, b;
    double s=0;
    if(argc != 2)
    {
        printf("Niepoprawna ilosc argumentow\n");
        _getch();
        return -1;
    }

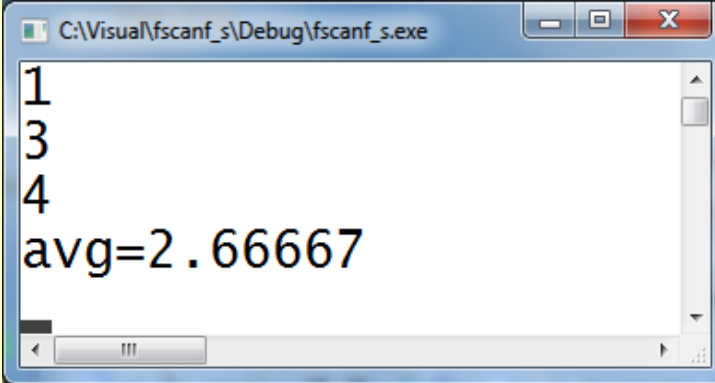
    if( fopen_s( &f, argv[1], "r,ccs=UNICODE" ) != 0)
    {
        printf( "Zbior %s nie zostal otwarty\n",argv[1] );
        _getch();
        return -1;
    }
}
```

// Deklaracja bibliotek

/\* Otwarcie pliku do  
czytania zapisanego w  
UNICODE \*/



W polu Project → Properties... → Configurations  
Properties → Debugging → Command Arguments  
wpisujemy nazwę pliku *we.txt*. Po uruchomieniu  
programu otrzymujemy następując wyniki.



A screenshot of a Windows command prompt window. The title bar shows the file path "C:\Visual\fscanf\_s\Debug\fscanf\_s.exe". The window contains the following text output:

```
1  
3  
4  
avg=2.66667
```

<pre>while(fwscanf_s(f,L"%d",&amp;b)!=EOF) {     printf_s("%d\n", b);     s+=b;     n++; }  printf_s("avg=%g\n",s/n);  _getch(); }</pre>	<pre>/*    Wyświetlenie    średniej arytmetycznej. */</pre>
--	---