

# Struktury danych

# Struktury

W informatyce, a zwłaszcza w programowaniu bardzo często operujemy na danych mających złożoną budowę. Złożoność budowy opiera się na powiązaniu kilku informacji ze sobą poprzez jedną ogólną dla nich nazwę. Oznacza to, że mówiąc np. o samochodzie będziemy mieli na myśli nie tylko jego model czy markę, ale również kolor, pojemność silnika, max. prędkość czy spalanie. Wszystkie te cechy będą ze sobą powiązane poprzez nazwę modelu samochodu. W ten sposób będziemy mieli utworzoną strukturę nazwaną za poszczególnym modelem, która będzie w sobie zawierała inne informacje określające interesujący nas model. Czytelnik za pewne zauważył analogię takiego sposobu opisu do omówionych już tablic. Warto również zapoznać się z podejściem przedstawionym w książkach [21, 28, 73, 91, 115]. W istocie struktura jest tablicą, której elementami są obiekty różnego typu. W bazach danych struktury noszą nazwę rekordów. Elementy lub zmienne struktury będziemy nazywali składowymi struktury. Formalny opis struktury stosowany w programowaniu ma poniższą postać.

```
struct nazwa_struktury  
{  
    lista elementów struktury  
};
```

Zauważmy, że tak jak wszystkie dane zapamiętywane poprzez komputer, struktura potrzebuje zadeklarowanego miejsca w pamięci komputera. Sam opis struktury nie rezerwuje miejsca w pamięci komputera. Dopiero deklaracja struktur wraz z odpowiednimi nazwami przydziela miejsce w programie. Strukturom podobnie jak tablicom możemy nadawać wartości początkowe. Zastanówmy się nad przykładem praktycznego zastosowania struktury. Dobrym tematem wydają się liczby zespolone.

Czytelnik z kursu algebry zapewne pamięta budowę liczb zespolonych. Każda z liczb zespolonych opisana jest poprzez wartość rzeczywistą  $Re(z)$  oraz część urojoną  $Im(z)$ . Widać zatem charakter strukturalny liczb zespolonych, gdzie poszczególne części liczby możemy przypisać składowym struktury. Przedstawimy teraz sposób wykorzystania struktury do napisania funkcji operującej na zmiennych zespolonych.

## Przykład 35

Wykonywać dodawanie liczb zespolonych za pomocą struktur.

W naszym programie dodawanie liczb zespolonych zdefiniowane zostanie następująco

$$(z.a, z.b) = (x.a + y.a, x.b + y.b) = (x.a, x.b) + (y.a, y.b)$$

Widzimy, że w wyrażeniach strukturalnych dostęp do składowej struktury uzyskujemy stosując postać:

`nazwa_struktury.składowa`

Przykładowo w opisie struktury zespolonej liczba składa się z dwóch zmiennych typu `double` *a* i `double` *b*. Nazwa struktury dość często jest nazywana etykietą struktury.

```
struct liczba{double a,b;}
```

W funkcjach struktury przekazywane są przez kopiowanie wartości. Tablice są przekazywane przez kopiowanie wskaźnika, nie ma kosztownego powielania wszystkich elementów tablicy. Przedstawmy sposób przekazywania struktury do funkcji i zwracania wartości za pomocą funkcji dodawania liczb zespolonych.

```
#include<stdio.h>
#include<conio.h>
struct liczba{ double a,b;};

struct liczba dodaj(struct liczba x, struct liczba y)
{
    struct liczba z;
    z.a = x.a + y.a;
    z.b = x.b + y.b;
    return z;
}
int main()
{
    struct liczba e={ 1.0,1.0}, f={ 2.0,2.0}, g;

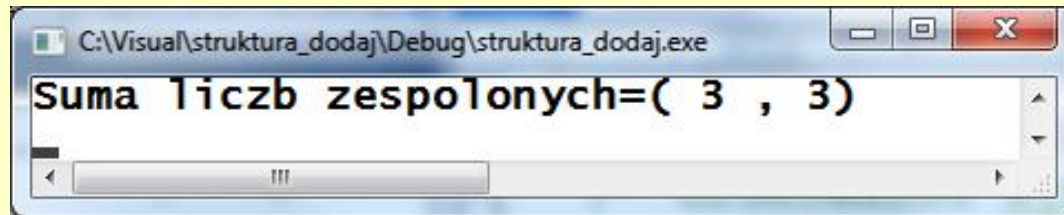
    g = dodaj(e,f);
    printf("Suma liczb zespolonych=( %g , %g)\n",
                                                g.a,g.b );

    _getch();
    return 0;
}
```

```
// Deklaracja bibliotek
// Definicja liczby zespolonej.

/* Deklaracja funkcji dodawania
liczb zespolonych w postaci
struktury. */
/* Operacja dodawania liczb
zespolonych. Zwracana wartość
jest w postaci odpowiedniej
pary liczb, czyli struktury.*/
// Funkcja główna programu.
/* Deklaracja struktur e oraz f w
postaci tablicy elementów.
Następuje przydział pamięci
komputera. */
/* Wywołanie funkcji i
podstawienie wyniku pod
strukturę g */
/* Drukowanie wyniku, jako
odpowiedniej pary liczb. */
```

Zobaczmy jak wygląda wynik napisanego programu. Ponieważ zadeklarowaliśmy dwie liczby zespolone o postaci (1,1) i (2,2), a następnie wykonaliśmy operację dodawania otrzymujemy wynik pokazany na poniższej ilustracji.



Czasami wygodnie jest przekazać strukturę przez wskaźnik. Deklaracja *struct liczba \*x;* mówi, że *x* jest wskaźnikiem do struktury *liczba*. Wskaźników do struktur używa się tak często, że zastosowano specjalną notację, aby odwoływać się do konkretnej składowej poszczególnej struktury za pomocą wskaźnika

**x -> składowa\_struktury**

Oczywiście odwołanie to można zrealizować w sposób tradycyjny znany już Czytelnikowi.

`(*x).składowa_struktury`

W wyrażeniu  $(*x).a$  nawiasy są konieczne, gdyż operator składowej struktury ma wyższy priorytet niż operator adresowania za pomocą wskaźnika  $*$ . Tak więc  $*x.a$  oznacza tyle samo co  $*(x.a)$ , a takie odwołanie jest niepoprawne, gdyż  $x$  jest wskaźnikiem. Zastanówmy się teraz jak zrealizować przedstawiony problem dodawania liczb zespolonych za pomocą adresowania pośredniego.



# Przykład 36

Program dodający liczby zespolone za pomocą adresowania pośredniego poprzez przekazanie danych przez adres.

```
#include<stdio.h>
#include<conio.h>
struct liczba{ double a,b;};

void dodaj(struct liczba *x,
  struct liczba *y,struct liczba *z)
{
  z->a = x->a + y->a;
  z->b = x->b + y->b;
}
```

```
// Deklaracja bibliotek
// Definicja liczby zespolonej.

/*  Funkcja  nie  zwraca
obliczonych wartości, gdyż
wykonuje operacje jedynie na
adresach zadeklarowanych
struktur. */
```

```
int main()
{
    struct liczba e, f, g;

    printf("Podaj pare liczb:");
    scanf("%lf %lf",&e.a,&e.b);
    printf("Podaj pare liczb:");
    scanf("%lf %lf",&f.a,&f.b);

    dodaj(&e,&f,&g);

    printf("Suma liczb zespolonych=( %g ,%g )\n",
                                                g.a, g.b);

    _getch();
    return 0;
}
```

// Główna funkcja programu.

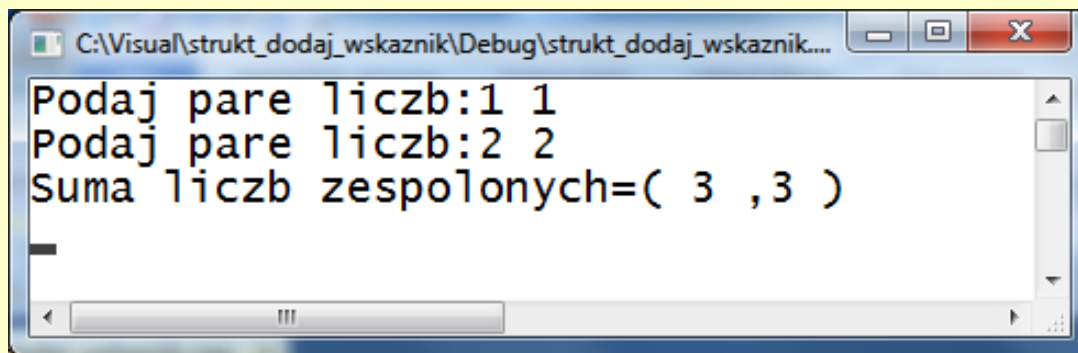
// Deklaracja struktury e, f, g.

// Wczytanie liczb zespolonych.

/\* Wykonanie funkcji dodaj i zapisanie wyniku w strukturze g. \*/

// Wydruk wyznaczonej wartości.

Wykonując program musimy wprowadzić dwie liczby zespolone, czyli cztery wartości.

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Visual\strukt\_dodaj\_wskaznik\Debug\strukt\_dodaj\_wskaznik....'. The window contains three lines of text: 'Podaj pare liczb:1 1', 'Podaj pare liczb:2 2', and 'Suma liczb zespolonych=( 3 ,3 )'.

```
C:\Visual\strukt_dodaj_wskaznik\Debug\strukt_dodaj_wskaznik....  
Podaj pare liczb:1 1  
Podaj pare liczb:2 2  
Suma liczb zespolonych=( 3 ,3 )
```

Wykorzystajmy teraz możliwości nadbudowy języka C w celu przeładowania operatora dodawania (+), tak abyśmy mogli dodać dwie liczby zespolone. Definicja struktury liczby zostaje uzupełniona o definicję operatora dodawania określonego na liczbach zespolonych. Struktura w omawianych przykładach odnosi się do deklaracji liczby samej w sobie.

# Przykład 37

Program dodający liczby zespolone poprzez przeładowanie operatora dodawania.

<pre>#include&lt;stdio.h&gt; #include&lt;conio.h&gt;  struct liczba {     double a,b;      liczba operator + (liczba y)     {         liczba temp;         temp.a=a + y.a;         temp.b=b + y.b;         return temp;     } };</pre>	<pre>// Deklaracja bibliotek  // Definicja struktury liczby.  // Definicja budowanego operatora dodawania.</pre>
--	--

```
int main(int argc, char* argv[])
{
    liczba x,y,z;
    printf("Podaj 1-liczbe:");
    scanf("%lf %lf", &x.a, &x.b);
    printf("Podaj 2-liczbe:");
    scanf("%lf %lf",&y.a,&y.b);

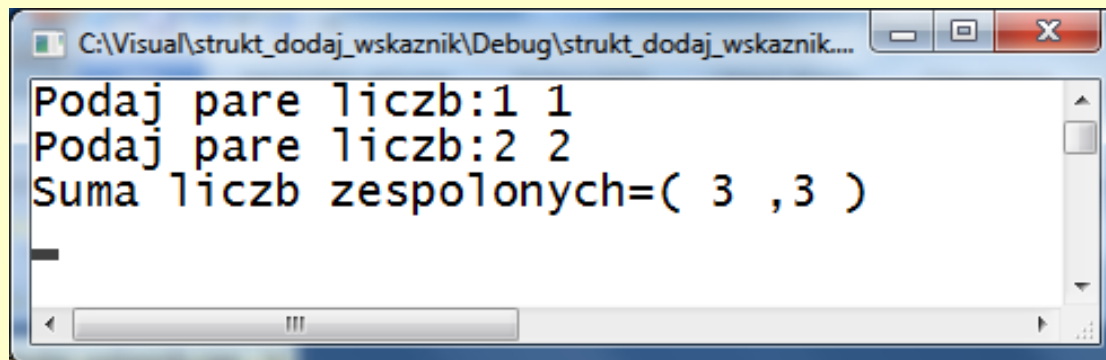
    z=x+y;
    printf("( %g , %g )\n",z.a,z.b);

    _getch();
    return 0;
}
```

// Funkcja główna programu.

// Wczytanie liczb do struktury.

/\* Wykonanie dodawania i wydruk wartości.\*/



```
C:\Visual\strukt_dodaj_wskaznik\Debug\strukt_dodaj_wskaznik...
Podaj pare liczb:1 1
Podaj pare liczb:2 2
Suma liczb zespolonych=( 3 ,3 )
```

Wykorzystajmy teraz możliwości nadbudowy języka C w celu przeładowania operatora dodawania (+), tak abyśmy mogli dodać dwie liczby zespolone. Definicja struktury liczby zostaje uzupełniona o definicję operatora dodawania określonego na liczbach zespolonych. Struktura w omawianych przykładach odnosi się do deklaracji liczby samej w sobie.

## Przykład 37

Program dodający liczby zespolone poprzez przeładowanie operatora dodawania.

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

```
struct liczba
```

```
{
```

```
    double a, b;
```

```
    liczba operator + (liczba y)
```

```
{
```

```
    liczba temp;
```

```
    temp.a=a + y.a;
```

```
    temp.b=b + y.b;
```

```
    return temp;
```

```
}
```

```
};
```

```
// Deklaracja bibliotek
```

```
// Definicja struktury liczby.
```

```
// Definicja budowanego operatora  
dodawania.
```

```
int main(int argc, char* argv[])
{
    liczba x,y,z;
    printf("Podaj 1-liczbe:");
    scanf("%lf %lf", &x.a, &x.b);
    printf("Podaj 2-liczbe:");
    scanf("%lf %lf",&y.a,&y.b);

    z=x+y;
    printf("( %g , %g )\n",z.a,z.b);
    _getch();
    return 0;
}
```

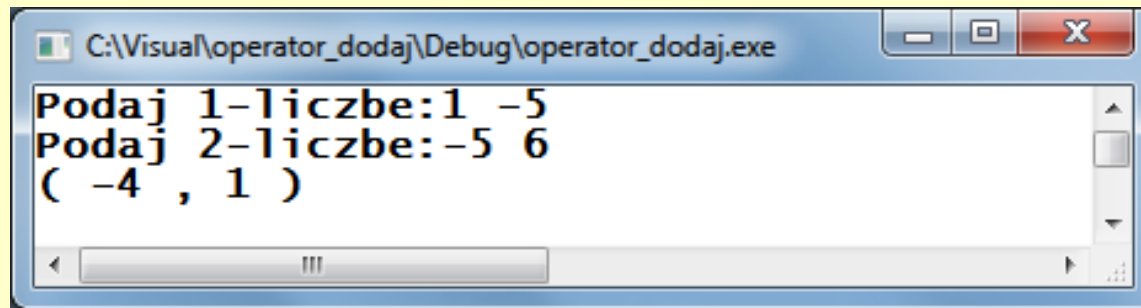
// Funkcja główna programu.

// Wczytanie liczb do struktury.

// Wykonanie dodawania i wydruk wartości.

Program możemy kompilować za pomocą kompilatora *g++* pod systemem operacyjnym Linux lub bezpośrednio w Visual Studio. Zwyczajowo programy napisane w C++ noszą rozszerzenie \*.cpp, chociaż nie jest to konieczne w przypadku użycia kompilatora *g++*. Działanie programu Czytelnik widzi na poniższej ilustracji.





```
C:\Visual\operator_dodaj\Debug\operator_dodaj.exe
Podaj 1-liczbe:1 -5
Podaj 2-liczbe:-5 6
(-4 , 1)
```

Operator (+) został przeładowany w celu określenia operacji dodawania na liczbach zespolonych. Pozostałe operacje na liczbach zespolonych są matematycznie zdefiniowane następująco.

Odejmowanie:

$$(a,b)-(c,d) = (a-c,b-d)$$

Mnożenie:

$$(a,b)*(c,d) = (a*c-b*d,a*d+b*c)$$

Dzielenie:

$$(a,b)/(c,d) = ((a*c+b*d)/(c*c+d*d),(b*c-a*d)/(c*c+d*d))$$

# Przykład 38

Programując przeładowanie pozostałych operatorów wykonać działania na liczbach zespolonych.

```
#include<stdio.h>
#include<conio.h>
struct liczba
{
    double a,b;
    liczba operator +(liczba y)
    {
        liczba temp;
        temp.a = a + y.a;
        temp.b = b + y.b;
        return temp;
    }
}
```

```
// Deklaracja bibliotek.
```

```
// Definicja struktury liczby.
```

```
/*    Definicja    budowanego
operatora dodawania.*/
```

```
    liczba operator -(liczba y){  
        liczba temp;  
        temp.a = a - y.a;  
        temp.b = b - y.b;  
        return temp;  
    }
```

```
    liczba operator *(liczba y){  
        liczba temp;  
        temp.a = a * y.a - b * y.b;  
        temp.b = a * y.b + b * y.a;  
        return temp;  
    }
```

```
    liczba operator /(liczba y){  
        liczba temp;  
        double t = y.a * y.a + y.b * y.b;  
        temp=y;  
        temp.b = -temp.b;  
        temp = temp * (*this);  
        temp.a /= t;  
        temp.b /= t;  
        return temp;  
    }
```

```
};
```

```
/* Definicja budowanego operatora  
odejmowania. */
```

```
/* Definicja budowanego operatora  
mnożenia.*/
```

```
/* Definicja budowanego operatora  
dzielenia.*/
```

```
int main(int argc, char* argv[])
{
    liczba x, y, z;
    printf("Podaj 1-liczbe:");
    scanf("%lf %lf", &x.a, &x.b);
    printf("Podaj 2-liczbe:");
    scanf("%lf %lf", &y.a, &y.b);

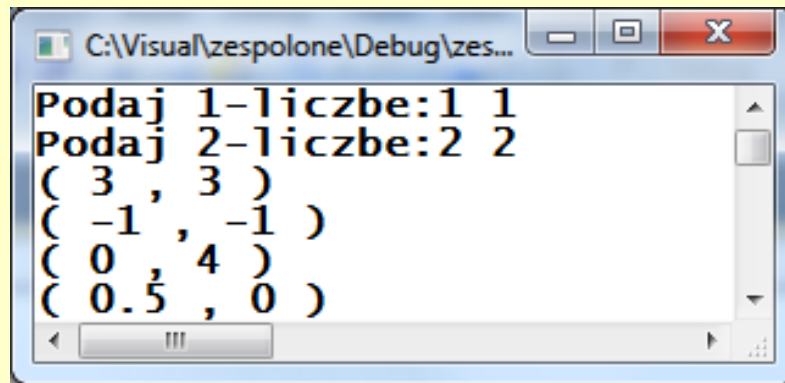
    z=x+y;
    printf("( %g , %g )\n", z.a, z.b);
    z = x - y;
    printf("( %g , %g )\n", z.a, z.b);
    z = x*y;
    printf("( %g , %g )\n", z.a, z.b);
    z = x/y;
    printf("( %g , %g )\n", z.a, z.b);

    _getch();
    return 0;
}
```

// Funkcja główna programu.

// Wczytanie liczb zespolonych.

// Wykonanie działań i wydruk wartości.



```
C:\Visual\zespolone\Debug\zes...
Podaj 1-liczbe:1 1
Podaj 2-liczbe:2 2
( 3 , 3 )
( -1 , -1 )
( 0 , 4 )
( 0.5 , 0 )
```

Pod systemem operacyjnym udostępniono bibliotekę *complex* umożliwiającą dokonywanie operacji na liczbach zespolonych. Parę liczb rzeczywistych  $(a,b)$  możemy interpretować jako punkt na płaszczyźnie o zadanych współrzędnych na osi  $X$  i na osi  $Y$ . Często zapisujemy liczbę zespoloną również w postaci  $(r, \varphi)$ , gdzie  $r$  jest odległością punktu  $(a,b)$  od początku układu współrzędnych  $O$ ,  $\varphi$  jest kątem pomiędzy osią  $OX$  oraz prostą przechodzącą przez początek układu współrzędnych i punkt  $(a,b)$ . Możemy zapisać liczbę zespoloną w postaci wzoru algebraicznego

$$z = a + b \cdot i, \text{ gdzie } i \cdot i = -1$$

lub w postaci trygonometrycznej

$$z = r \cdot e^{i \cdot \varphi} = r \cdot (\cos \varphi + i \cdot \sin \varphi)$$

Czytelnik porówna oba zapisy z przedstawioną na kolejnej ilustracji graficzną reprezentacją liczby zespolonej na płaszczyźnie. Liczba, którą pokazano na ilustracji ma zapis:

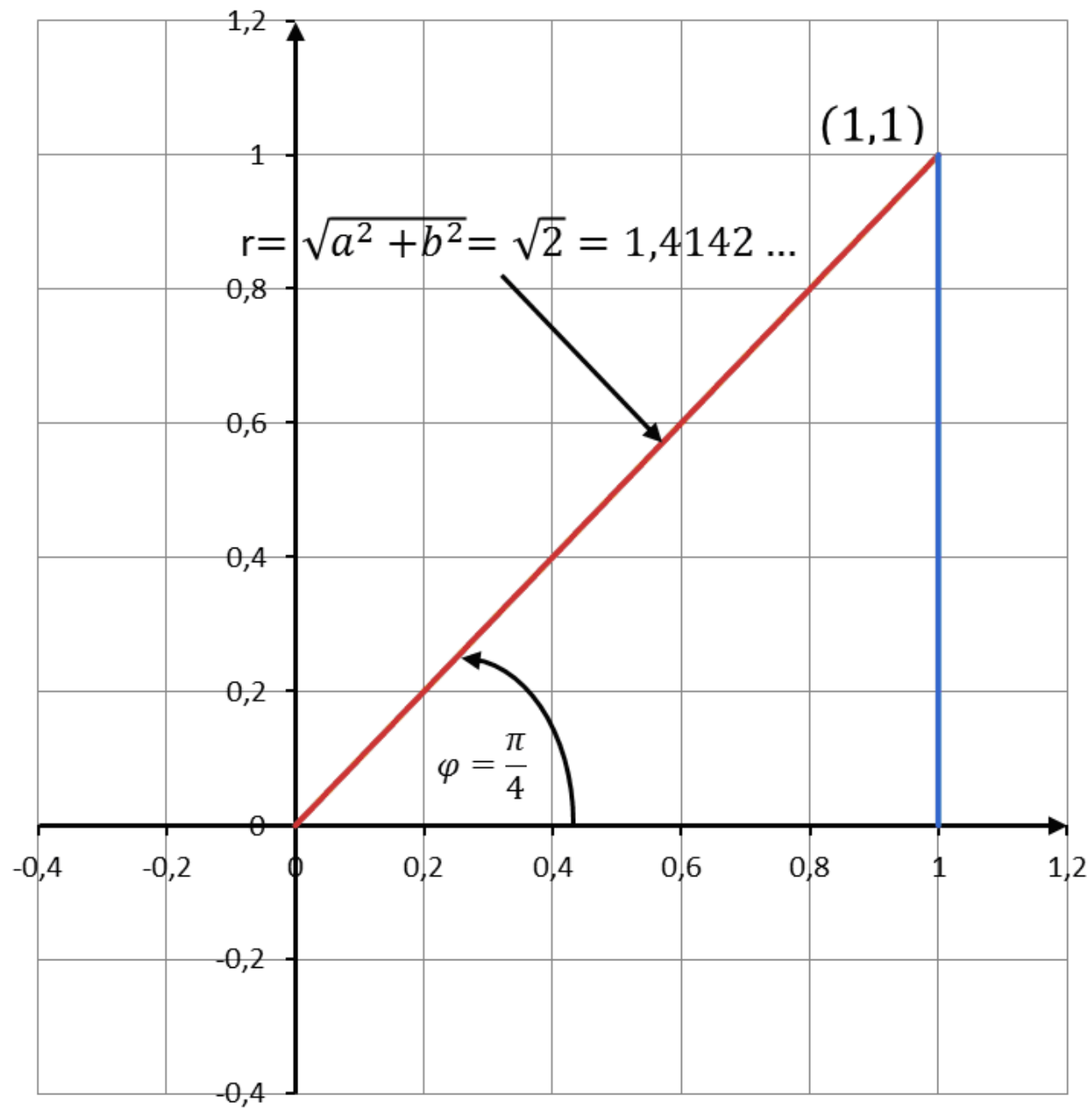
- W postaci algebraicznej korzystając z podanego wzoru liczba będzie mieć postać:  $z = 1 + i$ , gdzie współczynniki  $a$  oraz  $b$  obrazują współrzędne punktu A na płaszczyźnie,
- W postaci trygonometrycznej korzystając z podanego wzoru liczba będzie mieć postać:  $z = \sqrt{2} \cdot e^{i \cdot \frac{\pi}{4}} = \sqrt{2} \cdot \left( \cos \frac{\pi}{4} + i \cdot \sin \frac{\pi}{4} \right)$ , gdzie  $\varphi$  obrazuje kąt położenia względem osi poziomej natomiast  $r$  jest długością kreślonego odcinka.

Oba zapisy liczby zespolonej są równoważne, co pokazuje następujące równanie.

$$z = 1 + i =$$

$$= \sqrt{2} \cdot e^{i \cdot \frac{\pi}{4}} = \sqrt{2} \cdot \left( \cos \frac{\pi}{4} + i \cdot \sin \frac{\pi}{4} \right) =$$

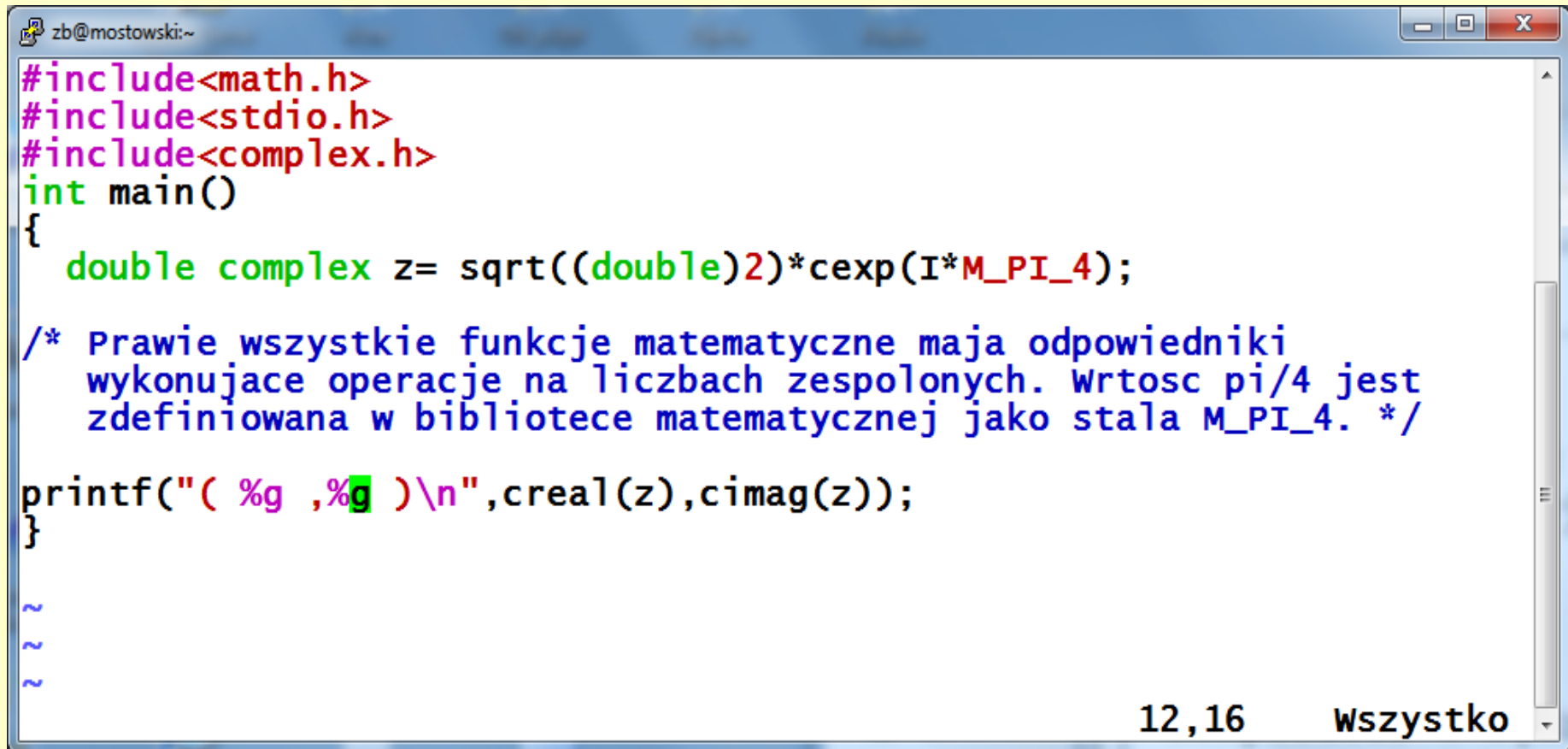
$$= \sqrt{2} \cdot \left( \frac{\sqrt{2}}{2} + i \cdot \frac{\sqrt{2}}{2} \right) = 1 + i$$





# Przykład 39

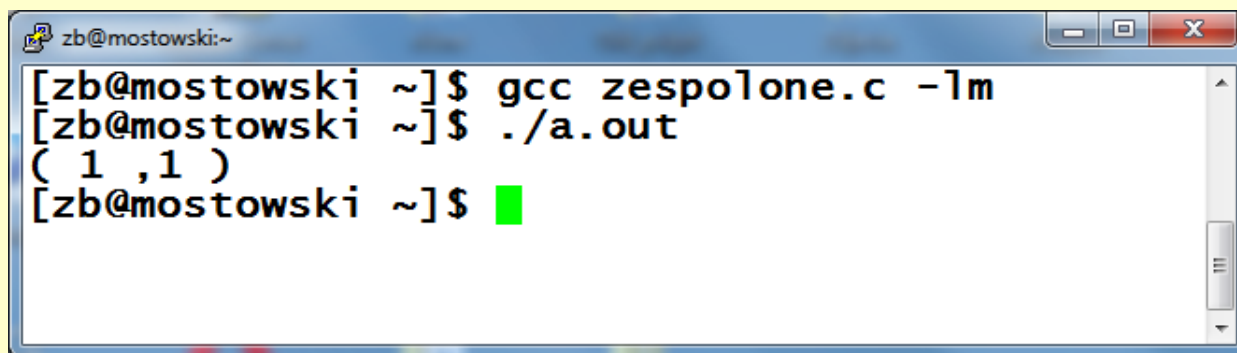
Napisać program obliczający  $\sqrt{2} \cdot e^{i \cdot \frac{\pi}{4}}$ , tak aby program był kompilowany pod systemem operacyjnym Linux.



```
zb@mostowski:~  
#include<math.h>  
#include<stdio.h>  
#include<complex.h>  
int main()  
{  
    double complex z= sqrt((double)2)*cexp(I*M_PI_4);  
  
    /* Prawie wszystkie funkcje matematyczne maja odpowiedniki  
       wykonujace operacje na liczbach zespolonych. Wrtosc pi/4 jest  
       zdefiniowana w bibliotece matematycznej jako stala M_PI_4. */  
  
    printf("( %g ,%g )\n",creal(z),cimag(z));  
}  
  
~  
~  
~
```

12,16      wszystko

Czytelnik zapewne zwrócił uwagę, że na powyższej ilustracji w edytorze linuxowym program zajmuje jedynie kilka linijek. Możliwe jest to dzięki wykorzystaniu możliwości zaawansowanych bibliotek matematycznych. Dla potrzeb zastosowanej biblioteki *complex* zdefiniowano stałą  $i$  odpowiadającą w zapisie liczbie  $i$ , gdzie  $i^2 = -1$ . Kompilator języka C będzie działał poprawnie, jeśli jest wykonany zgodnie ze standardem C99. Kompilację wykonamy z opcją linkowania  $-lm$ , co Czytelnik może prześledzić na poniższej ilustracji.



```
zb@mostowski:~$ gcc zespolone.c -lm
zb@mostowski:~$ ./a.out
( 1 ,1 )
zb@mostowski:~$
```

Obliczona wartość odpowiada liczbie zespolonej. Podobnie wygląda obsługa biblioteki *complex* w Visual Studio. Jednak w tym przypadku dodatkowo trzeba zadeklarować domyślną przestrzeń *std* (jest ona wymagana).

## Przykład 40

Napisać program obliczający  $\sqrt{2} \cdot e^{i \cdot \frac{\pi}{4}}$ , tak aby program był kompilowany pod systemem operacyjnym Windows w MS Visual Studio.

```

#include<complex>
#include<stdio.h>
#include<conio.h>
int main( )
{
    using namespace std;

    double pi_4 = atan(1.0);
    complex <double> co (polar(sqrt((double)2 ), pi_4));

    printf("( %g , %g)\n", co);
    double absco = abs ( co );
    double argco = arg ( co );

    printf("Modol liczby co uzyskany przy uzyciu:");
    printf("abs ( co ) = %g\n", absco );
    printf("Argument liczby co uzyskany przy uzyciu:");
    printf(" arg (co) = %g radianow\n", argco );
    printf(", to jest %g stopni.\n", argco*180/(pi_4*4));

    printf("Czesc rzeczywista = %g\n",real( co ));
    _getch();
}

```

```
// Deklaracja bibliotek
```

```

/*      Użycie      domyślnej
przestrzeni nazw std.*/
/* Deklaracja liczby w postaci
wykładniczej polar ( sqrt(
(double)2 ) , pi_4 ). */

```

```
// Obliczenie modułu liczby co.
```

```

/* Obliczenie argumentu liczby
co.*/

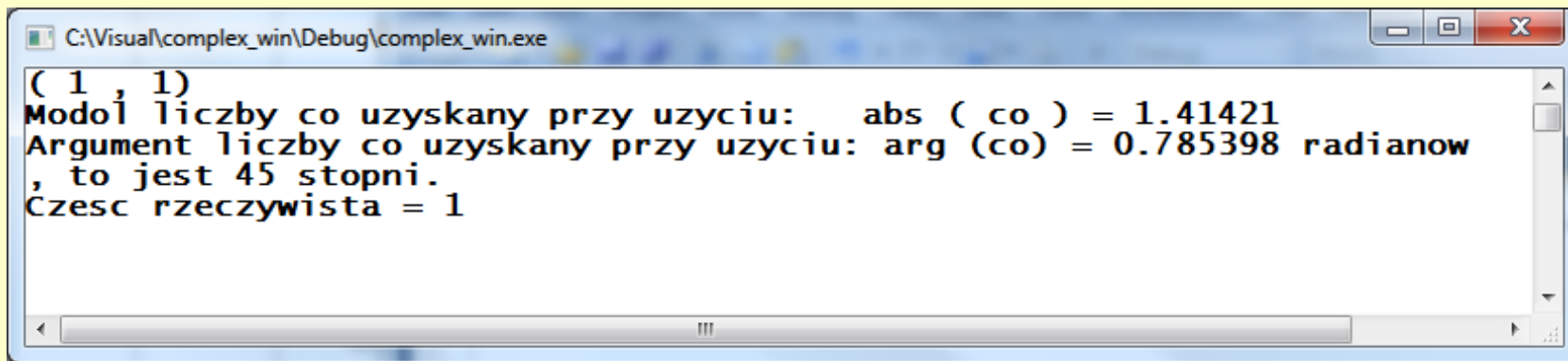
```

```

/* Wydruk części rzeczywistej
liczby co.*/

```

Czytelnik zechce prześledzić działanie operacji zapisanych w kodzie programu po jego skompilowaniu na poniższej ilustracji.



```
( 1 , 1)
Moduł liczby co uzyskany przy uzyciu: abs ( co ) = 1.41421
Argument liczby co uzyskany przy uzyciu: arg (co) = 0.785398 radianow
, to jest 45 stopni.
Czesc rzeczywista = 1
```

# Tablica w zapisie struktur

Programując bardzo często tworzymy swoistego rodzaju systemy połączonych ze sobą programów, które komunikują się między sobą. Komunikacja ta polega na przekazywaniu wyznaczonych wartości czy danych pomiędzy programami. Dzieje się tak, kiedy nasz program musi przeprowadzić skomplikowane obliczenia złożone z wielu etapów. W takim przypadku bardzo ważna jest umiejętność przekazywania danych pomiędzy programami w odpowiednim formacie. Jednym ze sposobów zwracania przez podprogram wskaźnika oraz tablicy jest zapisanie go w postaci struktury za pomocą tablicy.

```
struct tablica{ int n;double *a;};
```

# Przykład 41

Napisać funkcję czytającą ciąg liczb rzeczywistych z pliku *we.txt* oraz program wypisujący wartość największą i najmniejszą z ciągu przeczytanych liczb wraz ze średnią arytmetyczną ciągu przeczytanych liczb.

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

```
// Deklaracja bibliotek.
```

```
struct tablica
{
    int n;
    double *a;
};
```

```
// Deklaracja struktury.
```

<pre> tablica czytaj(char *s) {     struct tablica w = {0, NULL};     double e;     FILE *f;      if((f = fopen(s, "r")) == NULL) return w;      w.a=(double *)malloc(0);     while(fscanf(f, "%lf", &amp;e) != EOF)     {         w.n++;         w.a=(double*)realloc(w.a, sizeof(double)*w.n);         w.a[w.n-1] = e;     }     return w; }  int main(int argc, char **argv) {     double min, max, avg;     struct tablica w;     int i; </pre>	<pre> /* Funkcja wczytująca liczby do tablicy.*/ /* Deklaracja struktury oraz nadanie jej wartości początkowych. */  // Próba otwarcia pliku.  /* Próba przeczytania elementu z pliku.*/  /* Dodanie elementu do ciągu liczbowego. */  // Funkcja główna programu.  /* Deklaracja zmiennych i struktury.*/ </pre>
---	---



```
w = czytaj("we.txt");
if(w.n == 0)
{
    printf("Nie moge otworzyc pliku wejsciowego\n");
    _getch();
    return 0;
}
```

```
min = max = avg = w.a[0];
for(i=1;i<w.n;i++)
{
    if(w.a[i] > max) max = w.a[i];
    if(w.a[i] < min) min = w.a[i];
    avg += w.a[i];
}
```

```
printf("max=%g min=%g avg=%g\n",
        max, min, avg/w.n);
```

```
_getch();
return 0;
```

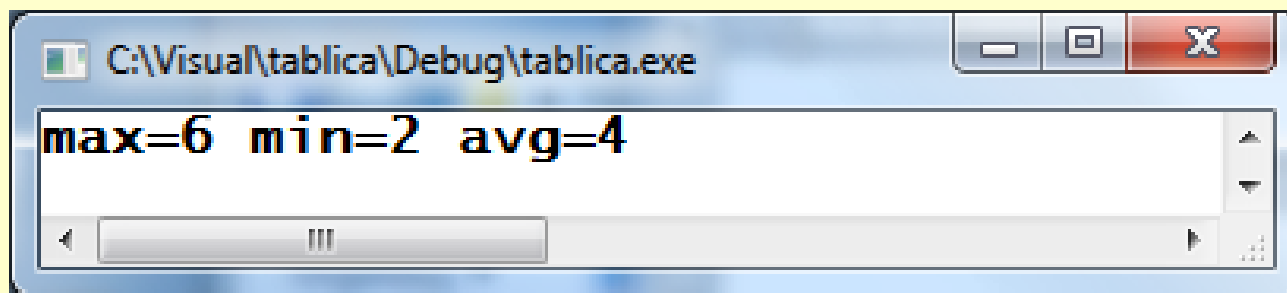
```
}
```

```
/* Wywołanie funkcji wczytującej.*/
/* Sprawdzenie czy funkcja
przeczytała ciąg liczb. */
```

```
/* Algorytm obliczania wartości
największej, najmniejszej i średniej
arytmetycznej. */
```

```
/* Wypisanie obliczonych wartości.
*/
```

Program działa poprawnie po utworzeniu pliku *we.txt* w katalogu domyślnym i wyświetla wyniki w zależności od wpisanych liczb, co pokazano na poniższej ilustracji.



Drugim sposobem działania tablicy jest przekazanie adresu wskaźnika tablicy do funkcji.

# Przykład 42

Napisać program spełniający te same zadania, co poprzedni za pomocą przekazania adresu wskaźnika tablicy do funkcji.

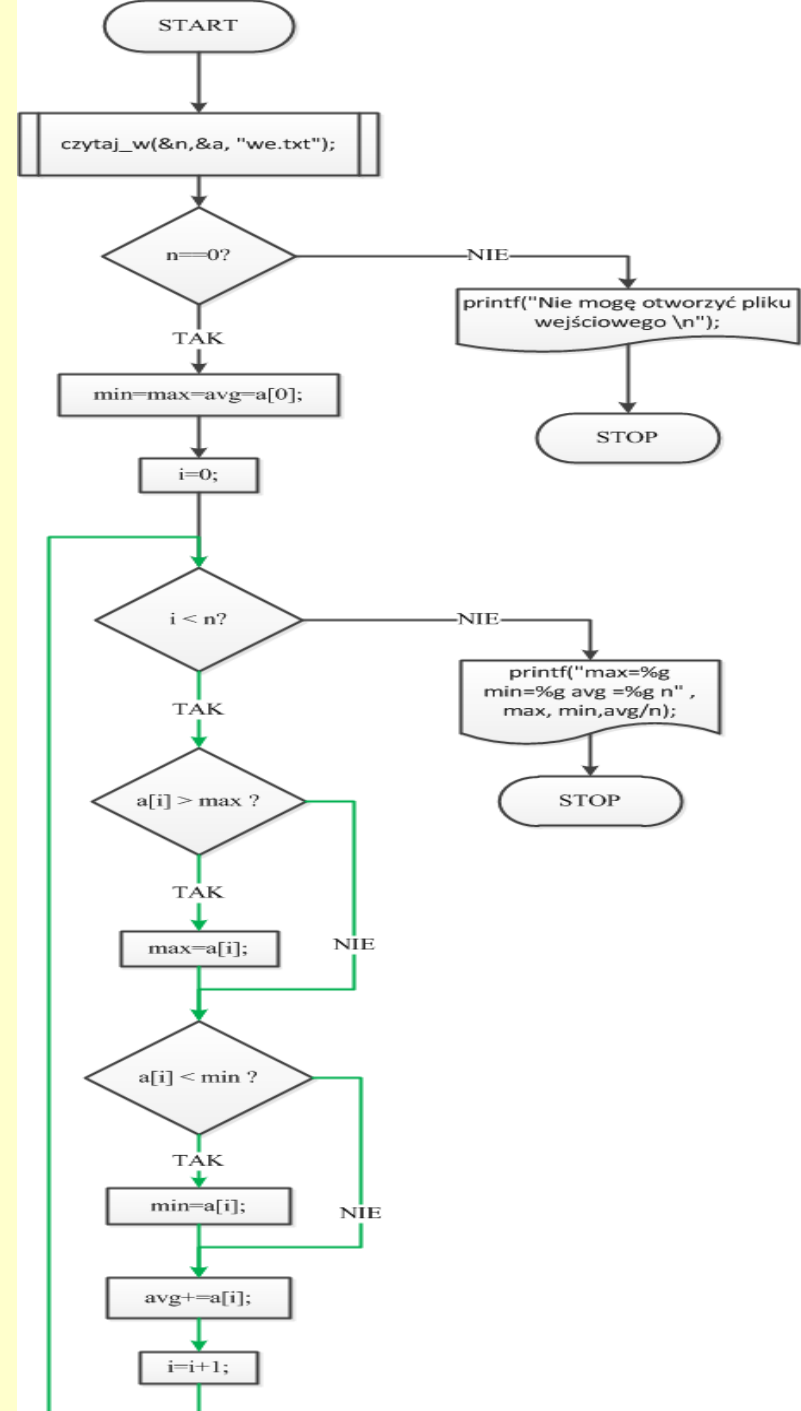
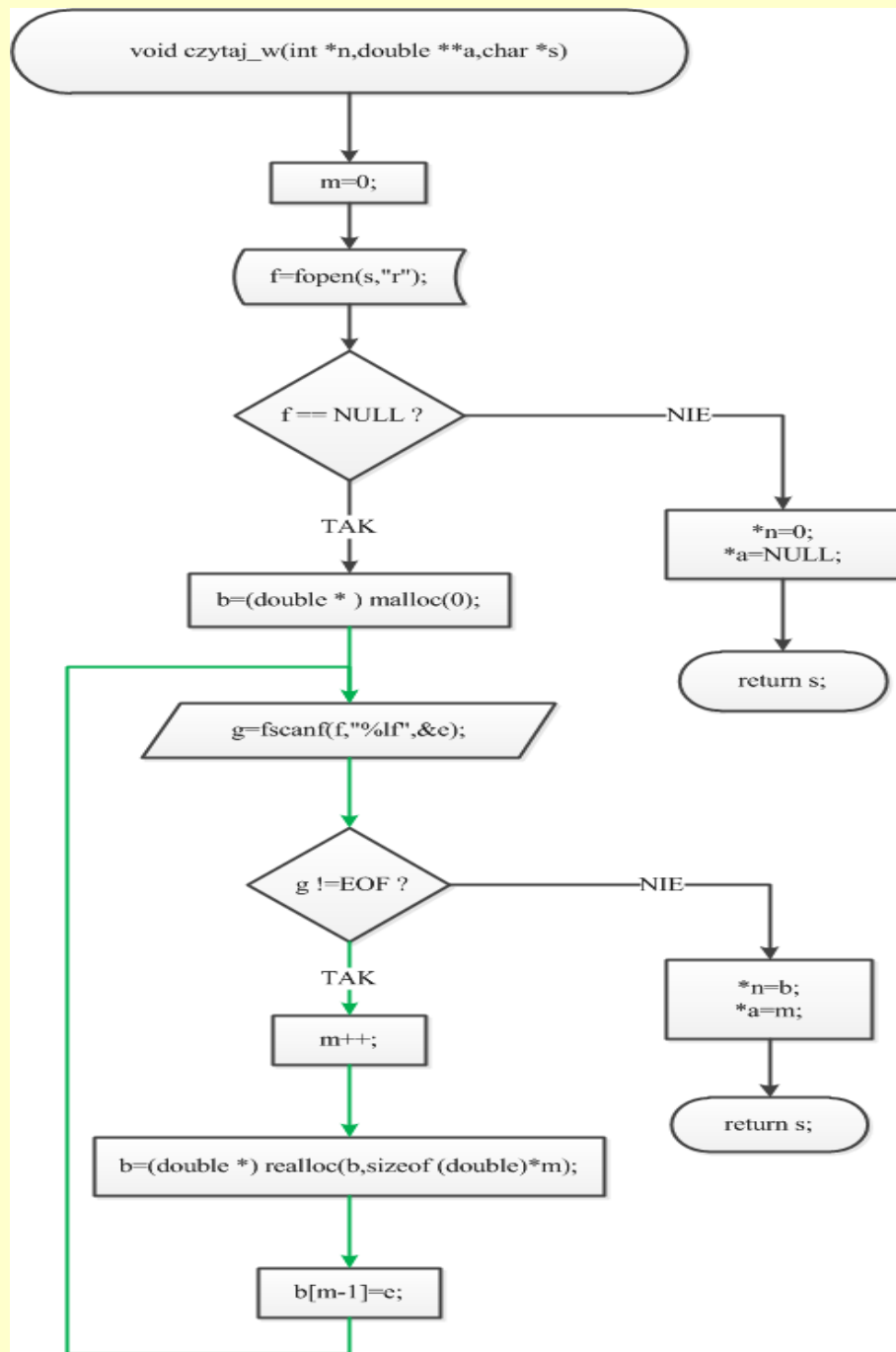
<pre>#include&lt;stdio.h&gt; #include&lt;stdlib.h&gt; #include&lt;conio.h&gt;  void czytaj_w(int *n,double **a,char *s) {     double e, *b;     int m=0;     FILE *f;      if((f = fopen(s,"r")) == NULL)     {         *n =0;         *a=NULL;         return;     } }</pre>	<pre>// Deklaracja bibliotek.  /*  Funkcja  wczytująca  dane  z pliku.*/  // Próba otwarcia pliku.</pre>
---	--

<pre>b=(double *)malloc(0); while(fscanf(f, "%lf", &amp;e) != EOF) {     m++;     b = (double *)realloc(b,sizeof(double)*m);     b[m-1]=e; } *a=b; *n=m; return; } int main(int argc,int **argv) {     int n,i;     double max,min,avg,*a;     czytaj_w(&amp;n, &amp;a, "we.txt");     if(n==0)     {         printf("Nie moge otworzyc pliku wejsciowego\n");         _getch();         return 0;     }</pre>	<pre>/* Próba przeczytania elementu z pliku.*/  /* Dodanie elementu do ciągu liczbowego.*/ /* Zwrócenie wartości do programu.*/  // Główna funkcja programu.  /* Wywołanie funkcji wczytującej.*/ /* Sprawdzenie czy wywołana funkcja przeczytała ciąg liczb. */</pre>
--	--

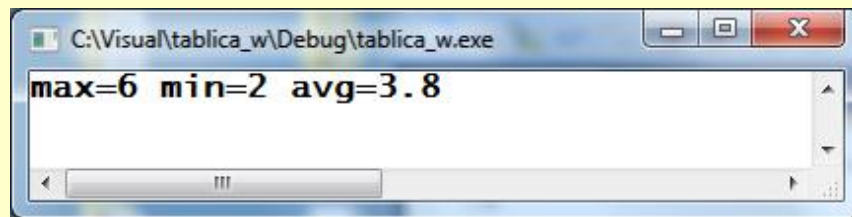
```
min = max = avg = a[0];
for(i = 1; i < n; i++)
{
    if(a[i] > max) max = a[i];
    if(a[i] < min) min = a[i];
    avg += a[i];
}
printf("max=%g min=%g avg=%g\n",
        max, min, avg/n);
_getch();
return 0;
}
```

```
/* Algorytm obliczania
wartości największej,
najmniejszej i średniej
arytmetycznej. */
```

```
/* Wypisanie obliczonych
wartości.*/
```

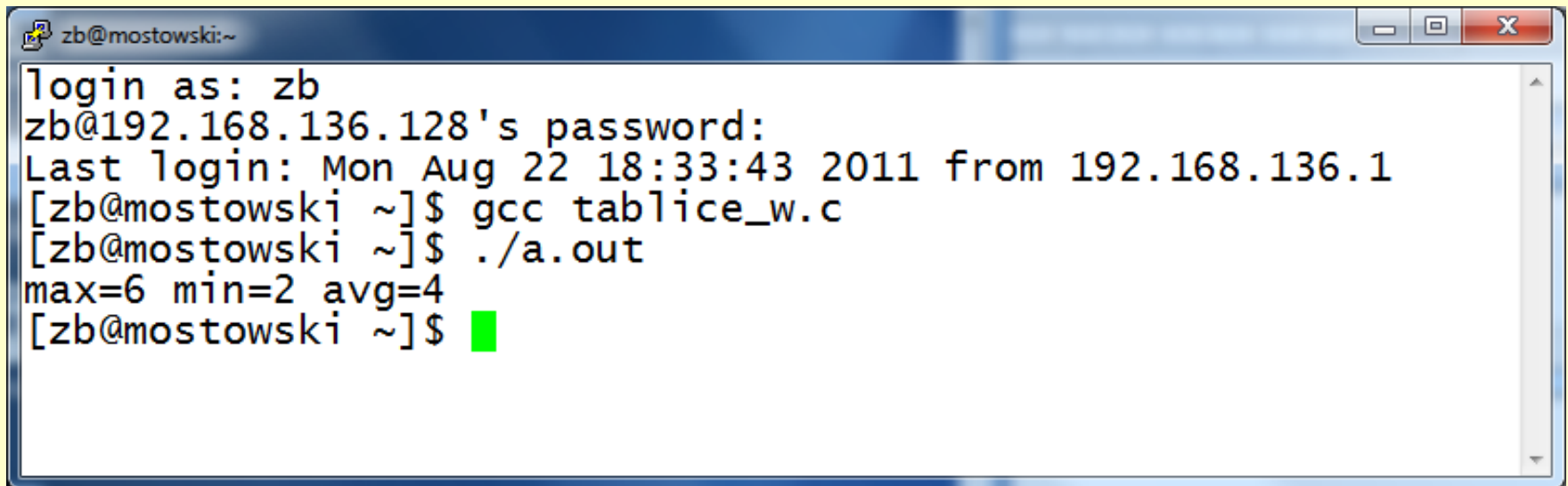


Powyższe schematy obrazują realizacje procedur wczytania danych oraz całego kodu wyznaczającego wartości ekstremalne dla danych wczytanych z pliku. Efekt działania stworzonego oprogramowania widzimy na poniższej ilustracji.



Uwaga: Przedstawiony program po wykreśleniu z kodu biblioteki *conio* oraz funkcji *\_getch()* działa poprawnie również pod system operacyjnym Linux. Pozostaje jednak otwarta sprawa kontekstowego znaczenia znaku \*. W wielu nowoczesnych językach zrezygnowano z arytmetyki na wskaźnikach. Nie udało się tego uniknąć w ostatnim programie przy przekazywaniu wskaźników pomiędzy funkcjami. Pozostaje się zgodzić z opinią, że język C/C++ powinien uzyskać nowy standard, choć ciągle jest to jeden z najczęściej stosowanych języków programowania. Działanie programu po skompilowaniu skorygowanego kodu programu pod systemem Linux zostało przedstawione na poniższej ilustracji.

# Okno kompilacji w systemie Linux kodu realizującego Przykład 42.



```
zb@mostowski:~  
login as: zb  
zb@192.168.136.128's password:  
Last login: Mon Aug 22 18:33:43 2011 from 192.168.136.1  
[zb@mostowski ~]$ gcc tablice_w.c  
[zb@mostowski ~]$ ./a.out  
max=6 min=2 avg=4  
[zb@mostowski ~]$
```



# Tablice struktur

Zastanówmy się teraz nad praktycznym zastosowaniem poznanych struktur. Najczęściej jednak w praktyce nie stosujemy jedynie rozwiązania podstawowego, ale oparte o kombinację kilku schematów podstawowych. Jednym z tradycyjnych zastosowań odpowiednio zbudowanej struktury jest lista płac. Do sporządzenia listy płac potrzebne są dane personalne pracownika. Strukturę danych personalnych pracownika dla prezentacji użycia struktury możemy zadeklarować w następujący sposób.

```
struct dane{ int id; char nazwisko[24], praca[24];};
```

Czytelnik zapewne zauważył, że zostało zastosowane połączenie omówionej struktury danych z poznanymi już tablicami. Jest to często wykorzystywana w literaturze [21, 28, 112, 115] forma. Zaprezentowana struktura jest swoistego rodzaju bazą danych, która zawiera interesujące nas rekordy. Ważnym jest podkreślenie zastosowania swoistego numerycznego identyfikatora każdej osoby w bazie. Identyfikator pracownika jest tworzony na potrzeby tabeli pracownicy i jest unikatowy. W bazach globalnych nie można tworzyć podstawowego klucza wyszukiwania na podstawie nazwiska, imienia czy daty urodzenia. Przy wzięciu pod uwagę tylko tych trzech składowych istnieje możliwość duplikatu klucza, z czym często można spotkać się w praktyce. Inaczej mówiąc, jeśli nie w naszej grupie czy roku to w kraju są osoby mające to samo nazwisko, imię czy urodzone tego samego dnia. Dla naszych potrzeb rekord (struktura) w naszej bazie jest bardzo prosta. Zakładamy, że nie ma osób o tym samym nazwisku. Do porównania ciągów znakowych użyjemy funkcji z literatury [39, 40].

```
int porownaj(char *s,char *t)
{
    while(*s == *t)
    {if(*s=='\0')return 0;
      s++;
      t++;}
    return *s-*t;
}
```

Funkcja ta porównuje dwa ciągi tekstowe i zwraca odpowiednio wartość ujemną, zero lub wartość dodatnią w zależności, czy pierwszy tekst jest leksykograficznie mniejszy od drugiego, równy lub większy.

# Przykład 43

Napisać program czytający rekordy z pliku dane.txt. W pliku dane zapisane są liniami. W każdej linii zapisany jest jeden rekord zakończony znakiem nowej linii. W linii znajdują się trzy pola oddzielone spacją id, nazwisko i praca (stanowisko). Po wczytaniu rekordów z pliku program poszukuje osoby o danym nazwisku i wypisuje wszystkie informacje o szukanej osobie albo wyświetla komunikat o nie znalezieniu tej osoby.

```
#include<stdio.h>
#include<conio.h>
struct dane{ int id;char nazwisko[24], praca[24];};
int porownaj(char *s,char *t)
{
    while(*s==*t)
    {
        if(*s=='\0')return 0;
        s++;
        t++;
    }
    return *s - *t;
}
```

```
// Deklaracja bibliotek.
// Deklaracja struktury.
/* Funkcja porownaj() zwraca:
<0,jeśli s<t, 0,jeśli s==t, >0,jeśli
s>t. */
/* Porównuj do momentu, gdy
znaki w obu tekstach są równe. */
/*Teksty są różne, bo doszliśmy
do ostatnich znaków obu tekstów.
*/
/* Zwracamy różnicę na pierwszej
pozycji, na której znaki są różne.
*/
```

```
int main()
```

```
{
```

```
    struct dane pracownik[1024];
```

```
    int i, n = 0;
```

```
    char x[1024];
```

```
    FILE *f;
```

```
    if((f=fopen("dane.txt","r"))==NULL)
```

```
    {
```

```
        printf("Nie mogę otworzyć pliku dane.txt\n");
```

```
        _getch();
```

```
        return 0;
```

```
    }
```

```
    while(fscanf(f, "%d %s %s",
```

```
        &(pracownik[n].id),pracownik[n].nazwisko,
```

```
        pracownik[n].praca) != EOF)
```

```
        n++;
```

```
    for(i = 0; i < n; i++)
```

```
        printf("%d %s %s\n",
```

```
            pracownik[i].id,pracownik[i].nazwisko,
```

```
            pracownik[i].praca);
```

```
    printf("Podaj szukane nazwisko:");
```

```
    scanf("%s", x);
```

```
/* Funkcja główna programu.*/
```

```
// Otwarcie pliku danych.
```

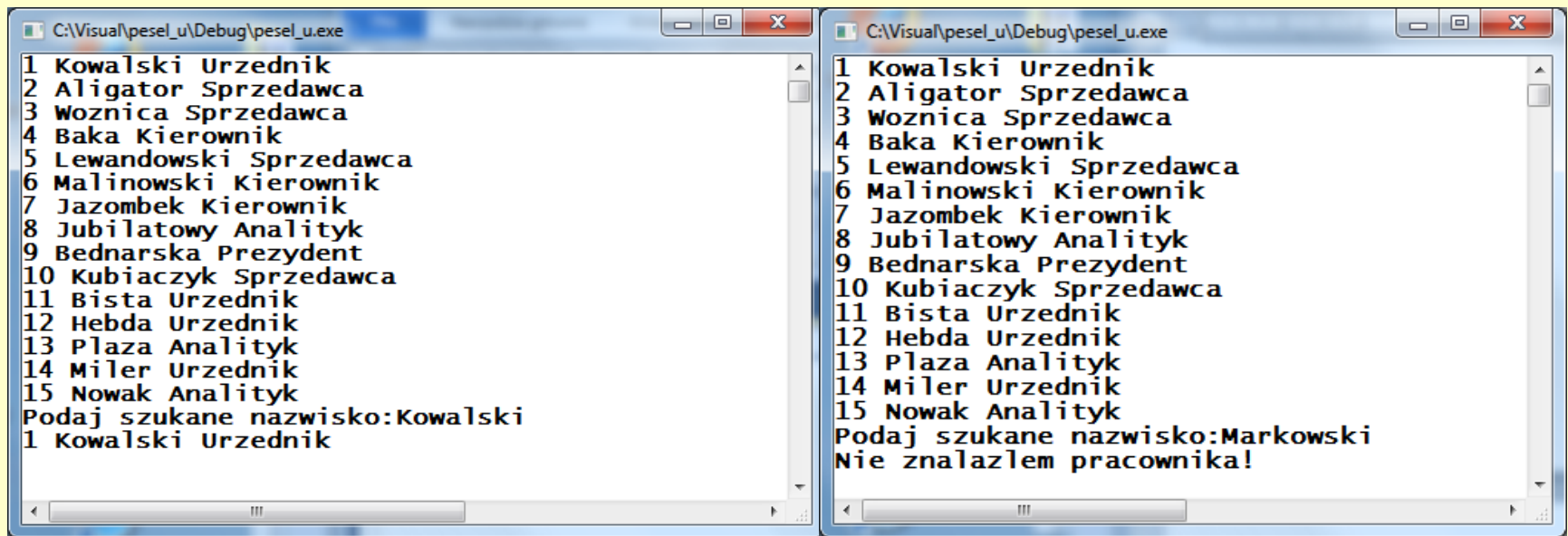
```
// Czytanie całej tabeli.
```

```
/*    Wydruk    przeczytanych  
rekordów.*/
```

```
/*          Wczytanie          osoby  
poszukiwanej w tabeli. */
```

<pre>for(i=0;i&lt;n;i++) {     if(porownaj(pracownik[i].nazwisko,x) == 0)     {         printf("%d %s %s\n",             pracownik[i].id,pracownik[i].nazwisko,             pracownik[i].praca);          break;     } } if(i == n) printf("Nie znalazlem pracownika!\n"); _getch(); }</pre>	<pre>/* Przeszukanie wczytanej tabeli w sposób liniowy. */  /* Sprawdzenie, czy doszliśmy do końca tabeli. */</pre>
--	---

Na przedstawionych ilustracjach Czytelnik zechce prześledzić działanie stworzonego oprogramowania w przypadku odnalezienia poszukiwanego rekordu lub, jeśli takowy w bazie nie istnieje.



Przedstawiony sposób wyszukiwania jest mało efektywny dla uporządkowanego leksykograficznie ciągu nazwisk. Zastanówmy się jak przyspieszyć działanie programu. Jeżeli najpierw posortujemy dane w tabeli według drugiego pola to możemy w celu znalezienia osoby zastosować algorytm binarnego poszukiwania zaprezentowany w kolejnym rozdziale.