Podstawy programowania
Programowanie w języku C

© dr hab. inż. Lidia Jackowska-Strumiłło, prof. PŁ

Instytut Informatyki Stosowanej
Politechnika Łódzka

Cechy języka C

- Język C jest niezwykle efektywnym narzędziem programowania.
- Jest elastyczny, przenośny i powszechny.
- C umożliwia wydajne programowanie na niskim poziomie niezależnie od systemu, co pozwala mu konkurować z asemblerami. Jednocześnie wykazuje przenośność charakterystyczną dla języków wysokiego poziomu.

Historia

Język C został zdefiniowany na początku lat siedemdziesiątych przez dwóch pracowników AT&T Bell Laboratories:

Briana Kernighan'a i Dennisa Ritchie'go.

Został on wykorzystany do przepisania kodu systemu operacyjnego UNIX.

1978 - opublikowanie książki *„Język C”,*

1989 – wprowadzenie standardu ANSI C,

1990 – ISO 9899:1990 C standard.

3

Symbole podstawowe

Alfabet języka C, tj. zbiór znaków, za pomocą których zapisuje się programy w tym języku zawiera:

- małe i duże litery alfabetu łacińskiego oraz “_” (znak podkreślenia zaliczany do liter),
- cyfry arabskie,
- znaki specjalne:

  + - * / = < > ( ) [ ] { } . , : ; ‘ ‘ " " ^ ! # & % | ~ ?

4

Konwencje leksykalne

Program składa się z jednej lub więcej *jednostek tłumaczenia* zapisanych w plikach. Kompilacja programu odbywa się w kilku fazach.

W fazie pierwszej - preprocesora, dokonuje się wstępnej leksykalnej transformacji programu, tzn. interpretuje się wiersze rozpoczynające się znakiem # oraz wykonuje makrodefinicje i makrorozwinięcia. Po zakończeniu fazy preprocesora program jest zredukowany do ciągu jednostek leksykalnych.

Jednostki leksykalne

Istnieje sześć klas jednostek leksykalnych:

- identyfikatory,
- słowa kluczowe,
- stałe,
- napisy,
- operatory,
- separatory.

Identyfikatory (nazwy)

Identyfikatorem nazywamy ciąg liter i cyfr zaczynający się od litery. Rozróżnia się małe i wielkie litery alfabetu.

Liczba znaków identyfikatora może być dowolna.

Dla **identyfikatorów wewnętrznych** znaczenie ma co najmniej pierwszych 31 znaków, ale w niektórych implementacjach może być ich więcej.

Identyfikatory o zewnętrznej łączności są bardziej ograniczone, np. w konkretnej implementacji ich znacząca długość może wynosić tylko sześć początkowych znaków bez rozróżniania małych i wielkich liter.

Słowa kluczowe

auto	break	case	char	const	continue	default	do
double	else	enum	extern	float	for	goto	if
int	long	register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void	volatile	while

Słowa kluczowe pisane są małymi literami.

Stałe

Wyróżnia się kilka rodzajów stałych:

- całkowite, -48 0X2A
- znakowe, 'a'
- zmiennopozycyjne, 123.5 -2.5E+6
- wyliczeniowe.

Wszystkie stałe maja jakiś typ.

Typy danych będą omówione w dalszej części wykładu.

9

Stałe całkowite

Stałą całkowitą składającą się z ciągu cyfr uważa się za:

- **dziesiętną**, jeśli rozpoczyna się cyfrą różną od 0;
- **ósemkową**, jeśli rozpoczyna się cyfrą 0 - stałe ósemkowe nie zawierają cyfr 8 i 9;
- **szesnastkową**, jeśli rozpoczyna się znakami 0x lub 0X – do cyfr szesnastkowych zalicza się litery od a do f – dla 0x, lub od A do F - dla 0X.

Stała całkowita może być opatrzona przyrostkiem, np. u lub U – *ang. unsigned*, l lub L – *ang. long*.

Typ stałej całkowitej zależy od jej postaci, wartości i przyrostka.

10

Stałe znakowe

- Stała znakowa jest ciągiem jednego lub wielu znaków zawartym w apostrofach, np. ‘x’.
- Wartością stałej znakowej zawierającej tylko jeden znak jest numeryczna wartość tego znaku w zbiorze maszyny wykonującej program.
- Wartość stałej wieloznakowej zależy od implementacji.
- Dla wyrażenia znaków, których nie zawierają stałe znakowe można stosować sekwencje specjalne.

Sekwencje specjalne

\n - nowy wiersz, NL (LF)
\t - tabulacja pozioma, HT
\v - tabulacja pionowa, VT
\b - cofanie, BS
\r - powrót karetki, CR
\f - nowa strona, FF
\a – alarm, BEL
\\ - kreska ukośna w lewo, \
\? - znak zapytania, ?
\’ - apostrof, ‘
\” - cudzysłów, ”
\ooo – liczba ósemkowa, ooo
\xhh – liczba szesnastkowa, hh

Stałe zmiennopozycyjne

Stała zmiennopozycyjna składa się z:

- części całkowitej,
- kropki dziesiętnej,
- części ułamkowej,
- litery e lub E;
- wykładnika potęgi – może być ze znakiem,
- opcjonalnego przyrostka typu (f,F,l lub L).

Niektóre z tych elementów można pominąć.

Przykłady:

1.2e5f *(float)* 3.85 (double) 1e-2L *(long double)*

13

Stałe wyliczeń

Identyfikatory zadeklarowane jako wyliczniki są stałymi typu *int*.

14

Napisy

- Napis (string) jest ciągiem znaków ujętym w znaki cudzysłowu, np. "abc".
- W stałych napisowych można stosować te same sekwencje specjalne, co w stałych znakowych.
- Sąsiadujące napisy mogą być sklejane podczas kompilacji programu.
- Stała napisowa jest tablicą, której elementami są znaki. Ostatnim elementem tablicy jest znak '\0', który nie należy do napisu.

15

Separatory

- **Separatorami** są:
  - **komentarz** – ciąg dowolnych znaków, pomiędzy znakami /*, a */ lub występujący po znakach // do końca wiersza - komentarze oznaczone znakami /*… */ nie mogą być zagnieżdżone i nie mogą występować wewnątrz napisu lub stałej znakowej,
  - odstępy (spacje),
  - znaki poziomej i pionowej tabulacji,
  - przejście do nowego wiersza, strony, itp.

Separatory są nazywane *„białymi plamami”.*

16

Pierwszy program – tekst.c

```
# include <stdio.h> // dolaczenie biblioteki stdio
/* To jest pierwszy program pisany w C */
/*****************************************************/
int main (void) //glowna funkcja programu
{
 printf("I'm Bond, James Bond!\n"); // drukowanie
        // tekstu "I'm Bond,…" na ekranie
 return 0;
}
```


Uruchomienie programu napisanego w języku C

Program źródłowy jest kompilowany i linkowany.

Polecenie tworzące kod wynikowy ma postać:

- tcc tekst.c – kompilator Turbo C++ firmy Borland – tekst.exe,
- bcc tekst.c – kompilator Borland C++ (tekst.exe),
- cl tekst.c – kompilatory C firmy Microsoft (tekst.exe),
- cc tekst.c – kompilacja w środowisku Unix (a.out).

Obecnie częściej niż z wiersza poleceń korzysta się ze ***zintegrowanych środowisk programistycznych*** firm Microsoft, Borland, IBM lub innych, np. MS Visual C++.

Podstawowe typy danych

W języku C występują cztery podstawowe proste typy danych:

- **char** – pojedynczy bajt przechowujący jeden znak,
- **int** - liczba całkowita, rozmiar zależy od systemu,
- **float** – liczba zmiennopozycyjna (rzeczywista) pojedynczej dokładności,
- **double** - liczba zmiennopozycyjna (rzeczywista) podwójnej dokładności.

19

Zmienne

- ***Zmienną*** nazywamy daną, która może przyjmować różne wartości w ramach typu przypisanego lub określonego dla tej zmiennej.
- Zmienne służą do przechowywania wartości i informacji, które są przetwarzane podczas wykonywania programu.
- W zmiennych mogą być zapisane liczby, nazwy, tekst itp.
- Wszystkie ***zmienne*** muszą być zadeklarowane przed pierwszym użyciem w programie.

20

Definicja zmiennych w języku C

Definicja i deklaracja

typ nazwa1, nazwa2;

Definicja, deklaracja i inicjalizacja:

typ nazwa = stała;

Przykłady:

```
int samochody;    // przechowuje liczby całkowite
char litera;      // wartość znakowa, np. ‘b’
float saldo;      // saldo bankowe
double pi = 3.14; // zmienna o wysokiej dokładności
char tytul [20] = "Pan Tadeusz"; //napis
```


Wyrażenia

- Wyrażenie jest zapisem algorytmu określającego sposób wyznaczania pewnej wartości.
- Wyrażenie to kombinacja nazw funkcji, zmiennych, stałych, operatorów i podwyrażeń, która zapisana jest w konwencji języka C.

Przykłady

```
a = b + c
printf("Hello")
mojafunkcja ()
```

Instrukcje

- Instrukcje są to konstrukcje języka opisujące czynności wykonywane na danych.
- W języku C istnieją dwa rodzaje instrukcji:
  - proste, które nie zawierają jako składowych innych instrukcji,
  - strukturalne, zbudowane na podstawie schematu strukturalizacji ciągu instrukcji.
- Instrukcja prosta to wyrażenie zakończone średnikiem.

Wyrażenia i instrukcje

Przykłady

wyrażenia:

```
a = b + 0.5*c
printf("Hello")
mojafunkcja ()
```

instrukcje proste:

```
a = b + 0.5*c; // przypisanie
 printf("Hello");
 mojafunkcja (); //wywołanie
                 //funkcji
```

Instrukcje proste

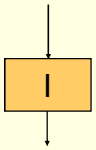
- ***Instrukcja przypisania:***

  z = w;

- ***Instrukcja wywołania funkcji:***

  nazwa_funkcji (*lista parametrów*);

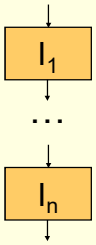
Symbol instrukcji w sieci działań algorytmu:



Instrukcja złożona

```
{
   instrukcja 1
   instrukcja 2
   …
   instrukcja n
}
```

Sekwencja w sieci działań algorytmu:



Instrukcja złożona nazywana jest też **blokiem instrukcji**.

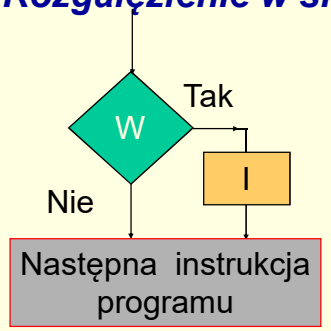
Instrukcje strukturalne

♦***Instrukcja warunkowa „if”:***

if (*warunek logiczny*) *instrukcja*

if (*warunek logiczny*) *instrukcja1* **else** *instrukcja2*

Rozgałęzienie w sieci działań algorytmu:



27

Instrukcja if

if (wyrażenie**)** instrukcja 1 **[else** instrukcja 2**]**

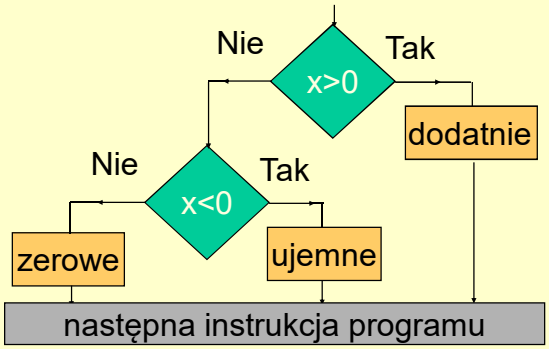
Przykłady:

```
if (s == 2) printf("Dwa pierwiastki");
if (x > y)  a = 1;   else a = 0;
if (w == z)
    {
        a=z-32.5;
        printf("Wartość wyrażenia: %f\n", a);
    }
```

28

Instrukcje warunkowe zagnieżdżone:

```
if (x > 0) printf("Dodatnie");
    else if  (x<0) printf("Ujemne");
                else printf("Zerowe");
```



29

Instrukcja for

warunek początkowy

powtarzaj pętlę dopóki warunek2 jest spełniony

wykonaj w każdym obiegu pętli

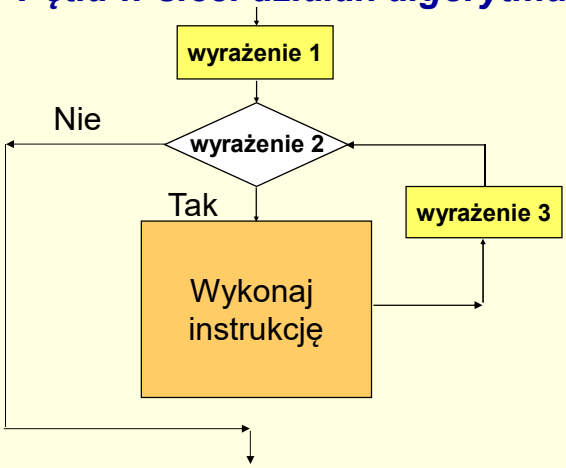
for (wyrażenie1**;** wyrażenie2**;** wyrażenie3**)** instrukcja

30

Instrukcja for

for (wyrażenie1; wyrażenie2; wyrażenie3) instrukcja

Pętla w sieci działań algorytmu:



Przykłady:

```
for (i=1; i<10; i++) printf("i =%d", i);

for (i=10; i>1; i--)
{
        x=5+10*i;
        printf("i=%d, x=%d", i, x);
}
```

Przykład 1

```
# include <stdio.h>
/************************************************/
int main (void)
{
   int x;
   for (x =1; x<100; x = x + 1)
   {
        printf("Liczba %d\n", x);
   }
   return 0;
}
```


Funkcja *printf* – szablon wydruku

printf to nazwa funkcji bibliotecznej, której deklaracja znajduje się w pliku nagłówkowym stdio.h. Jej pierwszym argumentem jest zawsze łańcuch znaków.

printf("łańcuch formatujacy", zmienne, wyrażenia);

Łańcuch formatujący składa się ze zwykłych znaków i sekwencji formatujących, które rozpoczynają się znakiem % i określają sposób wyświetlania następnych argumentów.

Sekwencje formatujące

- %i, %d, – liczba całkowita, dziesiętna,
- %u - liczba całkowita dziesiętna, bez znaku,
- %o – liczba całkowita ósemkowa, bez znaku, bez wiodącego zera
- %x, %X - liczba całkowita szesnastkowa, bez znaku, bez wiodącego 0x lub 0X,
- %c – znak,
- %s – łańcuch znaków (tekst),
- %f – liczba zmiennopozycyjna z kropką dziesiętną,
- %e - liczba zmiennopozycyjna w postaci wykładniczej,

35

Sekwencje formatujące, cd.

- %g - liczba zmiennopozycyjna o formacie %f lub %e w zależności od wartości liczby,
- %6d – liczba dziesiętna, zajmująca co najmniej 6 znaków,
- %5f – liczba zmiennopozycyjna, zajmująca co najmniej 5 znaków,
- %.2f – liczba zmiennopozycyjna z 2 miejscami po kropce dziesiętnej,
- %6.2f – liczba zmiennopozycyjna z 2 miejscami po kropce dziesiętnej, zajmująca co najmniej 6 znaków,
- %% - znak %.

36

Przykład 2

```
# include <stdio.h>
/*************************************************/
int main (void)
{
   int liczba = 6;
   float e = 2.718282;
   printf("Liczba całkowita to %d, a rzeczywista
                                  to %f \n", liczba, e);
   return 0;
}
/*************************************************/
```

37

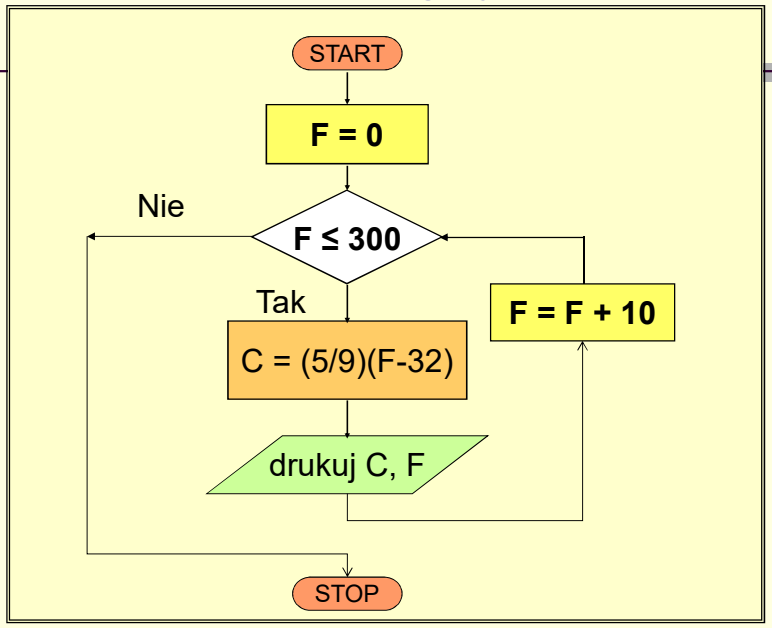
Przykład 3

Napisz program, który wypisuje zestawienie temperatur w skali Fahrenheita w zakresie od 0 do 300 stopni i ich odpowiedników w skali Celsjusza, co 10 stopni F, wyliczonych ze wzoru:

$$C = (5/9)(F-32)$$

38

Sieć działań algorytmu:



39

Przykład 3, cd.

```
# include <stdio.h>
/* zestawienie temperatur w skali Fahrenheita-
   Celsjusza */
int main (void)
{
   int fahr;
   for (fahr =0; fahr<=300; fahr = fahr + 10)
        printf("%3d %6.1f\n", fahr, 5.0/9*(fahr-32));
   return (0);
}
```

40

Przykład 4

Napisz program, który oblicza pole koła dla dowolnego promienia wpisanego przez użytkownika.

Dyskusja:

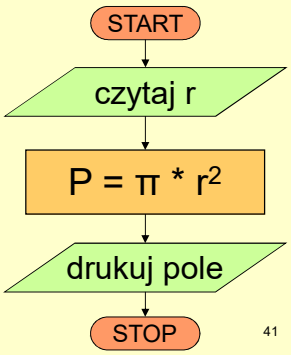
Dane wejściowe: promień

Dane wyjściowe: pole

Algorytm:

$$P = \pi r^2$$

Sieć działań algorytmu



41

Przykładowy program

```
/*Plik: Pole1.c        program do obliczania pola koła*/

#include <stdio.h>
#define PI 3.14159                  /*część deklaracyjna*/

int main()                          /*część wykonawcza*/
{
   float r,pole;                    /*to na promień i wynik*/
   printf("Podaj promien: ");
   scanf("%f", &r);                 /*czytaj promień*/
   pole=PI*r*r;                          /*oblicz pole*/
   printf("Pole kola o promieniu r = %.4f wynosi %.4f", r, pole);
                                    /*wypisz wynik*/
   return 0;
}
```

42

Czytelność programu

```
#include <stdio.h>
#define PI 3.14159
int main()
{
    float r,pole;
    scanf("%f", &r);
    pole=PI*r*r;
    printf("Pole kola o promieniu r = %.4f wynosi %.4f", r, pole);
    return 0;
}
```

```
#include <stdio.h> #define PI 3.14159
int main() { float r,pole; scanf("%f", &r);
pole=PI*r*r; printf("Pole kola o promieniu r = %.4f wynosi %.4f", r,
    pole);return 0;}
```


Przykładowy program 2

```
/*Plik pole2.c            program do obliczania pola koła*/

#include <stdio.h>
#include <math.h>

int main ()                          /*program główny*/
{
    float promien, pole;             /*deklaracje zmiennych*/
    printf("Program do obliczania pola koła o promieniu r\n");
    printf ("Podaj promień koła r=" );
    scanf("%f",&promien);                    /*czytaj promień*/
    pole=M_PI*pow(promien,2);                    /*policz pole*/
    printf("Pole koła o promieniu r=%.4f wynosi %.4f",promien, pole);
                                             /*wypisz wynik*/
    return 0;
}
```

Wprowadzanie danych

Funkcje biblioteki stdio.h:

getchar – pobiera znak z klawiatury,

gets(tablica_znakowa) – wczytuje łańcuch znaków do tablicy,

scanf("sekwencja formatująca", &zmienna_1, …, &zmiena_n)

- pobiera z klawiatury dane tekstowe, przekształca je zgodnie z kodem sekwencji formatującej i umieszcza je pod adresami wskazywanymi przez odpowiednie argumenty na liście zmiennych.

45

Funkcja scanf

- Sekwencje formatujące funkcji scanf są analogiczne do sekwencji formatujących funkcji printf. Rozpoczynają się znakiem %, po którym następuje kod formatujący.
- Każdemu argumentowi na liście zmiennych odpowiada pole znaków wejściowych. Jest ono zdefiniowane jako ciąg czarnych znaków rozciągających się do najbliższej „białej plamy” lub jeśli zdefiniowano rozmiar pola, na długość zdefiniowaną tym rozmiarem.

46

Przykłady

```
# include <stdio.h>
   …
int c, intwar, dzien, miesiac, rok;
c = getchar ();  /* Program czeka, aż użytkownik
   wpisze z klawiatury znak i naciśnie klawisz
   [Enter]. Znak zostaje przypisany zmiennej c,
   a [Enter] ignorowany */
scanf("%d", &intwar); // wczytanie liczby całkowitej
/* wczytanie daty w postaci dd/mm/yy : */
scanf("%u/%u/%u", &dzien, &miesiac, &rok);
```

47

Kody formatujące funkcji scanf

d,i,o,u,x – wczytuje odpowiednio liczbę dziesiętną, całkowitą, ósemkową, bez znaku (nieujemną) lub szesnastkową i umieszcza ją pod adresem wskazywanym przez odpowiedni argument na liście zmiennych, będący wskaźnikiem do liczby całkowitej;

i – liczba całkowita może być dziesiętna, ósemkowa (z wiodącym zerem) lub szesnastkowa (z wiodącym 0x lub 0X);

e,f,g – wczytuje liczbę zmiennopozycyjną i zapisuje ją jako zmienną typu float;

c, s – wczytuje odpowiednio znak lub łańcuch znaków.

48

Modyfikatory kodów formatujących funkcji scanf

%*s – gwiazdka przed kodem powoduje odrzucenie odpowiadających mu danych wejściowych, które nie są zapisywane do pamięci;

%10s – liczba całkowita po znaku % określa maksymalną długość pola wczytywanych znaków;

Opcjonalne znaczniki długości: h, l i L - umieszczone przed kodem modyfikują typ zmiennej, w której jest zapisywana dana, np.:

%hd – short int; %ld – long int; %lf – double;

%Lf – long double.

Przykład 5

Problem:

Należy napisać program do rozwiązywania równań kwadratowych:

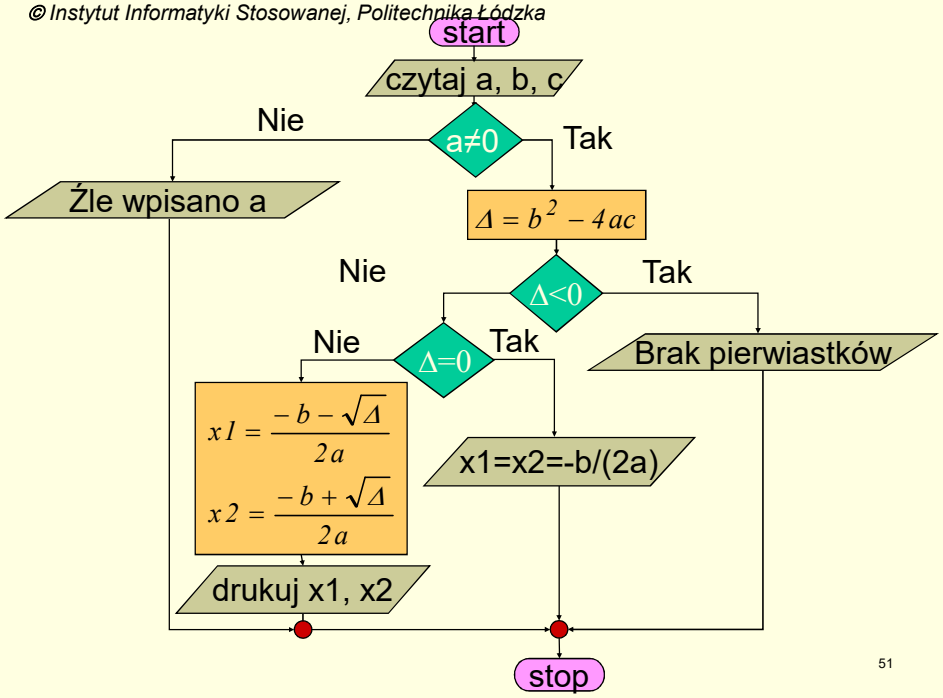
$$ax^2 + bx + c = 0, \qquad \text{gdzie:} \quad a \neq 0$$

Dyskusja:

Algorytm tego programu wynika bezpośrednio z matematyki, więc nie trzeba go układać. Polega on na obliczeniu wyróżnika, sprawdzeniu jego znaku dla określenia liczby i rodzaju pierwiastków oraz obliczeniu pierwiastków.

Dane wejściowe: współczynniki równania

Dane wyjściowe: obliczone pierwiastki, tj. ich wartość i rodzaj (rzeczywiste lub zespolone).



51

```
/*Plik rownanie.c          program obliczający pierwiastki równania kwadratowego*/
#include <stdio.h>
#include <conio.h>
#include <math.h>

int main()                                                    /*program główny*/
{
  float  a,b,c,delta,x1,x2;                          /*współczynniki, delta, pierwiastki*/
  printf("Program oblicza pierwiastki równania kwadratowego");
  printf("Podaj kolejno współczynniki równania a,b,c dla a różnego od zera");
  scanf("%f %f %f",&a,&b,&c);
  if (a!=0)
  {
            delta=b*b - 4*a*c;
            if (delta< 0)  printf("Brak pierwiastków rzeczywistych");
            else if (delta == 0) printf ("x1=x2=%.4f", -b/(2*a));
                 else
                 {
                       x1= (-b - sqrt(delta))/(2*a);
                       x2= (-b + sqrt(delta))/(2*a);
                       printf("x1 = %.4f, x2=%.4f",x1,x2);
                 }
  }
  else printf("Źle wpisano a, powtórz jeszcze raz\n");
  getch();
  return 0;
}                                                             /*koniec programu*/
```

52

Przykład 6

Napisz program, który wyznacza współczynnik **BMI** ze wzoru:

$$BMI = m/h^2$$

gdzie: m – masa w kg, h – wzrost w metrach.

Wypisz na końcu informację, czy waga danej osoby jest w normie ($20 \le BMI \le 25$), czy też ma ona nadwagę lub niedowagę.

Obliczanie BMI

Dyskusja:

Dane wejściowe: masa i wzrost

Dane wyjściowe: **BMI**, informacja czy waga danej osoby jest w normie czy też ma ona nadwagę lub niedowagę

Algorytm:

1. Wczytaj masę w kg ***m*** i wzrost w metrach ***h***.
2. Oblicz BMI ze wzoru: $BMI = m/h^2$
3. Drukuj BMI
4. Jeśli BMI < 20 wypisz: "Masz niedowage",
   w przecinym wypadku:
   jeśli BMI ≤ 25 wypisz: "To jest w normie",
   w przecinym wypadku: "Masz nadwage".

```c
# include <stdio.h>
   /* obliczanie BMI*/

int main (void)
{
   char  tekst [20];
   float  m,h,bmi;
   printf("Wpisz imie: ");
   scanf("%s", tekst);
   printf("Podaj wage w kg i wzrost w metrach:\t");
   scanf("%f %f", &m, &h);
   bmi = m/(h*h);
   printf("\n\nHello %s! \n\nTwoj BMI wynosi: %5.2f\n", tekst, bmi);
   if (bmi<20) printf("\aMasz niedowage\n\n");
         else if (bmi<=25) printf("To jest w normie\n\n");
                  else printf("\aMasz nadwage\n\n");
   getchar();
   return 0;
}
```


Przykład 7

Problem:

Należy obliczyć średnią z **n** liczb.

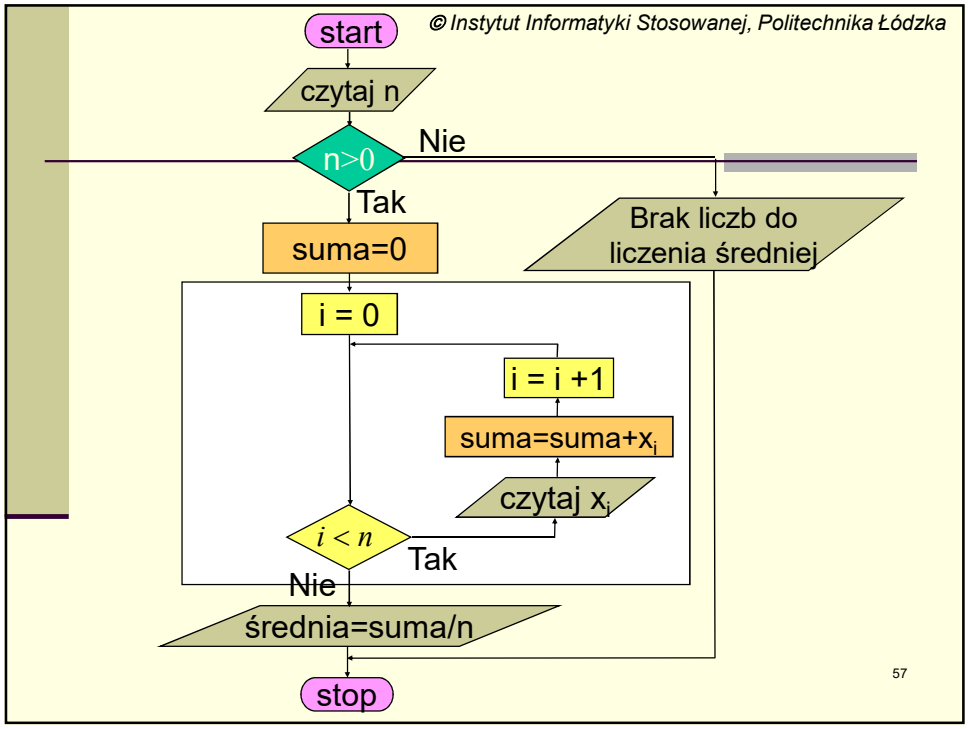
Dyskusja:

Dane wejściowe: ilość liczb n oraz ich wartości

Dane wyjściowe: średnia z n liczb

Algorytm:

1. Wczytać ilość liczb ***n*** i zbadać, czy $n>0$.
2. Jeśli tak, wczytywać i sumować w pętli ***n*** liczb, a potem obliczyć i wypisać średnią.
3. Jeśli nie wypisać odpowiednią informację dla użytkownika.




```c
/*Plik srednia.c          Program obliczający średnią arytmetyczną*/
#include <stdio.h>
int main()                                  //program główny
{
    int i, n;                               // zmiena pętli, ilość liczb
    float x, suma;                  // liczba sumowana, wynik
    printf("Program do obliczania średniej arytmetycznej\n");
    printf ("Podaj ilość liczb n = ");
    scanf("%d", &n);
    if(n>0)                                 // instrukcja if, warunek logiczny
    {                                       // początek instrukcji 1
        suma=0.0;
        for(i=0; i<n; i++)              //początek pętli for
        {
            printf("Podaj liczbę %d: ", i+1);
            scanf("%f",&x);
            suma=suma+x;
        }                                   //koniec pętli for
        printf("Średnia wynosi %8.3f", suma/n);
    }
    else printf("Brak liczb do policzenia średniej\n");     // instrukcja2
    return 0;
}
```

58

Przykład 8

Problem:

Należy wyznaczyć numery porządkowe (kody) liter i cyfr.

Dyskusja:

Dane: litery i cyfry
Dane wejściowe: brak
Dane wyjściowe: litery, cyfry i ich kody ASCII

Algorytm:

Należy wypisać w pętli kolejne litery i ich kody ASCII, a w następnej pętli cyfry i ich kody ASCII.


```
/*Plik kody_znakow.c
Program znajduje kody ASCII liter i cyfr*/

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

int main()
{
    char znak;
    printf("Program wypisuje kody ASCII znaków\n");
    for(znak='a'; znak<='z'; znak++)
        printf("Litera %c ma znak %d\n", znak, znak);
    for(znak='0'; znak<='9'; znak++)
        printf("Cyfra %c ma znak %d\n", znak, znak);

   getch();        // conio.h
   return 0;
}
```

Konwersja danych

Funkcje biblioteki stdlib.h:

atoi(tablica_znakowa) – przekształca tekst na liczbę całkowitą, generując zero, jeśli wystąpi błąd;

atof(tablica_znakowa) - przekształca tekst na liczbę zmiennopozycyjną podwójnej precyzji, generując zero, jeśli wystąpi błąd;

atol(tablica_znakowa) – przekształca tekst na liczbę całkowitą typu **long int**, generując zero, jeśli wystąpi błąd;

Przykłady

```
# include <stdio.h>
# include <stdlib.h>
   …
char  tekst [20];
int     intwart;
double  floatwart;

gets(tekst);
intwart = atoi(tekst);
floatwart = atof(tekst);
```

Przykład 9

Napisz program, który oblicza sumę kolejnych liczb od 1 do n, wykorzystując wzór Abela:

$$s = n\,(n+1)\,/\,2,$$

gdzie:

s – suma liczb,

n – ostatnia liczba w szeregu dodawanych liczb.

63


```
# include <stdio.h>
# include <stdlib.h>
   /* sumowanie kolejnych liczb w szeregu od 1 do n*/
int main (void)
{
   char tekst [20];
   int    n,s;
   printf("Podaj liczbe n: ");
   gets(tekst);
   n = atoi(tekst);
   if (n!=0)
   {
   s = n * (n+1) /2;
   printf("Suma liczb calkowitych od 1 do %d wynosi: %d\n", n,s);
   } else printf("Wpisano bledne dane\n");
   getchar();
   return 0;
}
```

64

Wyrażenia arytmetyczne

- Tak jak w matematyce kolejność wykonywania działań wskazują nawiasy, później priorytet operatorów oraz łączność operatorów.
- Rozróżniamy następujące kategorie operatorów arytmetycznych, wymienione w kolejności malejącego priorytetu:
  - Jednoargumentowe
  - Multiplikatywne
  - Addytywne

  Operatory arytmetyczne dwuargumentowe są lewostronnie łączne.

65

Operatory arytmetyczne

C	Pascal	Opis
+	+	dodawanie
-	-	odejmowanie
/	/ lub div	dzielenie
*	*	mnożenie
%	mod	reszta z dzielenia dwóch liczb całkowitych

66

Instrukcja *do ... while*

do instrukcja while (wyrażenie);

W pętli *do…while* wykonywana jest instrukcja, a następnie sprawdzana jest wartość logiczna wyrażenia. Jeśli wyrażenie jest prawdziwe, pętla wykonywana jest jeszcze raz.

Przykład:

```
int x =0;
do {
        x=x+1;
        printf("Liczba %d\n", x);
   } while (x<100);
```


Przykład 10

Problem:

Dla dowolnych wczytanych liczb całkowitych należy sprawdzić, czy są one parzyste, czy nieparzyste. Sprawdzanie należy zakończyć po pojawieniu się liczby równej *zero*.

Dyskusja:

Dane wejściowe: liczby całkowite

Dane wyjściowe: komentarz – *parzysta* lub *nieparzysta*

Algorytm:

1.Należy powtarzać wczytywanie w pętli kolejnych liczb, aż do czasu napotkania *zera*.

2.Jednocześnie dla każdej liczby należy sprawdzać jej parzystość (sprawdzając resztę dzielenia przez 2 - operator %) i wypisywać odpowiednią informację na ekranie.

```
/*Plik parzystosc.c
        program sprawdzający parzystość liczb całkowitych*/

#include <stdio.h>
 int main()
{
   int n;
   printf("Program czyta liczby i sprawdza ich parzystosc\n");
   printf("az do znalezienia liczby rownej zero\n");
   do
   {
     printf("Podaj liczbe: ");
     scanf("%d", &n);
     if((n%2)==0)
              printf("Liczba %d jest parzysta\n", n);
     else printf("Liczba %d jest nieparzysta\n", n);
   } while (n!=0);
   getchar();
   return 0;
}
```


Przykład 11

Problem:

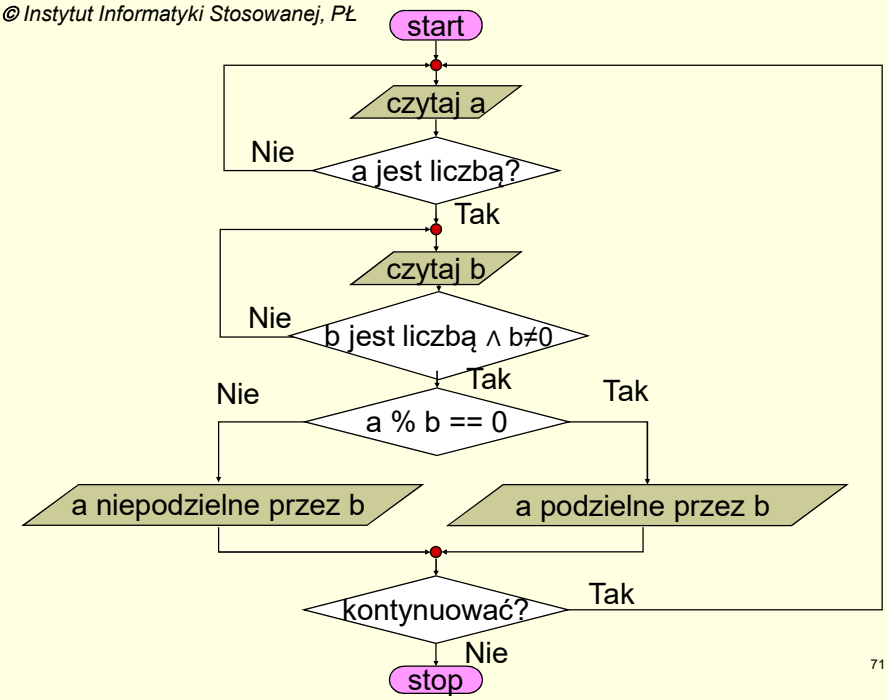
Napisz program, który dla każdej dowolnej pary liczb całkowitych **a** i **b** sprawdza, czy liczba **a** jest podzielna przez **b**. Po zakończeniu sprawdzania każdej pary liczb użytkownik powinien mieć możliwość wyboru, czy wczytać kolejną parę liczb, czy zakończyć program. Program powinien być odporny na błędne wprowadzenie danych.

Dyskusja:

W programie należy wykorzystać funkcję ***atoi.***

Algorytm:

- Wczytywać w pętli kolejne pary liczb, sprawdzać, czy **a** jest podzielne przez **b** i wypisywać odpowiednią informację na ekranie. Na końcu sprawdzić warunek wyboru.
- Jeśli użytkownik wciśnie klawisz *„n”* lub *„N”,* zakończyć program, a dla każdego innego klawisza kontynuować sprawdzanie.



71


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

int main() {
   int a, b, e;
   char z, tmp[10];
   printf("Program sprawdza podzielnosc liczby a przez liczbe b\n");
   printf("Podaj kolejno dwie liczby calkowite\n");
   do {
    do { fpurge(stdin);      // czyszczenie bufora klawiatury (GCC, UNIX)
      printf("podaj a: ");
      e = scanf("%d",&a);
    }while(e==0);
    do {fpurge(stdin);
      printf("podaj b: ");
      scanf("%s", tmp);
    }while(atoi(tmp)==0);
    b = atoi(tmp);
    if((a%b)==0) printf("Liczba a=%d jest podzielna przez b=%d", a, b);
    else printf("Liczba a=%d nie jest podzielna przez b=%d", a, b);
    fpurge(stdin);
    printf("\nKontynuowac ? (n-koniec)\n");
    z=getchar();
   }while((z!='n')&&(z!='N'));
   return 0;
}
```

72

Typ wyrażenia arytmetycznego

- Jeżeli wyrażenie zawiera instrukcję przypisania, to jego typ jest określony przez typ zmiennej, której przypisuje się wartość.
- W innych przypadkach typ wyrażenia wynika z jego składników.

Przykłady:

```
int a;
double b,c,d;
a = b * c / d        // wyrażenie typu int,
b * c / d            // wyrażenie typu double.
```


Konwersja typów i rzutowanie

- Rzutowanie jest to wymuszona konwersja typów, która odbywa się za pomocą operatora rzutowania.
- Operator rzutowania składa się z nazwy typu ujętej w nawiasy klamrowe i poprzedza on rzutowane wyrażenie:

  (nazwa_typu) wyrazenie_rzutowane

Przykład:

```
5 / 7   // wynik_wyrazenia = 0 – dzielenie całkowite
(float)5 / 7 // wynik_wyrazenia = 0.71428
5 / 7.0 // wynik_wyrazenia = 0.71428
```

Operatory zwiększania i zmniejszania

++	Zwiększenie o 1
- -	Zmniejszenie o 1

Operatory te mogą być zarówno **przedrostkowe** (przed zmienną: ++n), jak i **przyrostkowe** (po zmiennej: n++). Są one różne. W obu przypadkach zmieniana jest wartość zmiennej n, ale:

++n – zwiększa n przed użyciem jej wartości,

n++ - zwiększa n po użyciu jej wartości.

Przykłady:

```
n=5;   x = n++; // to x=5, n=6
       x = ++n; // to x=6, n=6
```

76

Wartość logiczna wyrażenia

Każde wyrażenie w języku C ma przypisaną pewną wartość:

- wartość zerowa to „fałsz”,
- wartość niezerowa to „prawda”.

Jako zmienną logiczną można wykorzystać np. zmienną typu int.

Przykład:

```
int poprawna_data = 0; // nadaje wartość „fałsz”
if (poprawna_data == 0)
   printf("Wprowadzono niepoprawną datę.\n");
```

77

Operatory porównania

Operatory relacji:

>	większe niż
<	mniejsze niż
>=	większe lub równe
<=	mniejsze lub równe

Operatory przyrównania:

==	równe
!=	nie równe (różne)

Operatory logiczne

Operator	Funkcja
!	negacja
&&	koniunkcja
||	alternatywa

Operatory porównania i logiczne zwracają zawsze wartość 1 („prawda”) albo 0 („fałsz”).

Operatory i wyrażenia przypisania

Wyrażenie typu:

wyrazenie1 op= wyrazenie2

/* gdzie: $op \in \{+ - * / \%\}$ */

jest równoważne wyrażeniu:

wyrazenie1 = (wyrazenie1) op (wyrazenie2)

Przykłady:

i +=2 lub i = i + 2

x *= y + 1 lub x = x * (y + 1)

xval % s [a+b] += 4

lub xval % s [a+b] = xval % s [a+b] + 4

Przykład 12

- Znaleźć najmniejszą liczbę naturalną n taką, że:

$$1+\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{n}>\varepsilon \quad ,$$

gdzie ε jest dowolną liczbą dodatnią.

```c
# include <stdio.h>
# include <stdlib.h>
/* program szukający n, dla ktorego spelniona jest
   nierownosc 1+1/2+1/3+…+1/n>e */

int main (void)
{
char t[10];
int n = 1;
float e,s=0;
do {
    printf("Podaj liczbe dodatnia e: ");
    gets(t);  e = atof(t);
} while (e <= 0);
    do   {
          s += 1/(float)n;  n++;
    } while (s <= e);
    printf("%d\n", n-1);
    getchar();
return (0);
}
```

84

Priorytety i łączność operatorów

Operatory	Łączność
( ) [ ] -> .	lewostronna
! ~ ++ -- + - * & (typ) sizeof	prawostronna
* / %	lewostronna
+ -	lewostronna
<< >>	lewostronna
< <= > >=	lewostronna
== !=	lewostronna
&	lewostronna
^	lewostronna
|	lewostronna
&&	lewostronna
||	lewostronna
?:	prawostronna
= += -= *= /= %= ^= |= <<= >>=	prawostronna
,	lewostronna

85

Efekty uboczne

W języku C nie określa się kolejności obliczania argumentów operatora. Wywołania funkcji, zagnieżdżone instrukcje przypisania oraz operatory zwiększania i zmniejszania powodują efekty uboczne.

Przykłady:

```
x = f( ) + g( );
printf("%d %f\n", ++n, pow(2,n));
a[ i ] = i++;
```


Instrukcja *while*

while (wyrażenie) instrukcja

W pętli *while* najpierw obliczana jest wartość wyrażenia - jeśli jest ona różna od zera, to wykonywana jest instrukcja.

Przykład:

```
while ((c = getchar( )) == ' ' || c == '\n' || c == '\t')
  ; /* instrukcja pusta – przeskocz białe znaki */
```

Pętle *while* i *for*

Pętla

for (wyrażenie1; wyrażenie2; wyrażenie3)

instrukcja

jest równoważna konstrukcji:

```
wyrażenie1;
while (wyrażenie2) {
                    instrukcja
                     wyrażenie3;
}
```


Przykład 1

```
# include <stdio.h>
/* program drukuje kolejne liczby od 1 do 100
   oraz ich kwadraty */

int main (void)
{
int i;
   for (i =1; i <=100; i++)
                printf("Liczba %d\t%d\n", i, i*i);
return 0;
}
```

Przykład 2

```
# include <stdio.h>
/* program drukuje kolejne liczby od 1 do 100
   oraz ich kwadraty */
void main ()
{
int i =1;
   while (i <=100)
   {
        printf("Liczba %d\t%d\n", i, i*i);
         i++;
   }
}
```


Przykład 13

Problem: Dla dowolnej zadanej liczby ***n*** obliczyć najmniejszą potęgę dwójki większą od ***n***.

Dyskusja:

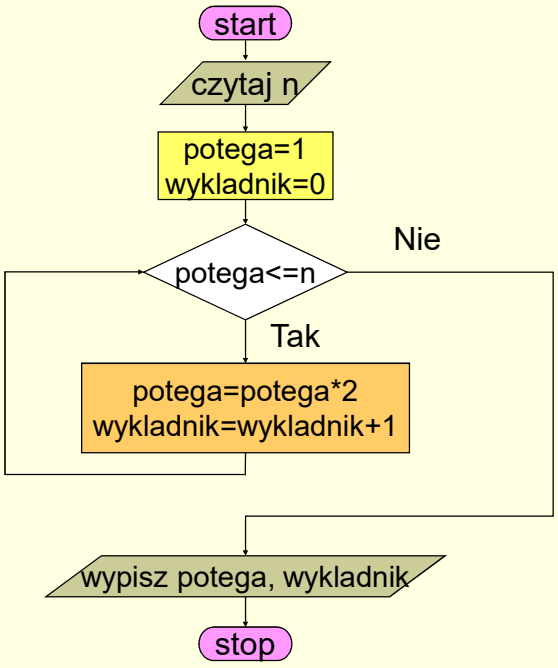
Dane wejściowe: n

Dane wyjściowe: potęga dwójki, wykładnik

Należy sprawdzać kolejne potęgi ***2*** poczynając od zerowej. Wartości te będą przechowywane w zmiennej ***potęga***, natomiast aktualny wykładnik w zmiennej ***wykładnik***.

Algorytm:

- wczytaj ***n***;
- ustaw wartości początkowe (*wykładnik=0; potęga=1*;);
- sprawdź, czy ***potęga*** <= ***n***;
- jeśli ***potęga*** <= ***n***, mnóż ją przez 2 i zwiększaj ***wykładnik***;
- wypisz wynik.



93


```c
/*Plik  potega_dwojki.c       Program znajduje najmniejszą liczbę będącą
potęgą dwójki, która jest większa od dowolnej liczby n*/
#include <stdio.h>
int main()
{
   int n, potega, wykladnik;
   printf("Program znajduje najmniejszą potege dwojki\n");
   printf("wieksza od zadanej liczby\n");
   printf("Podaj liczbe n: ");
   scanf("%d", &n);
   wykladnik=0;
   potega=1;
   while(potega<=n)                    //początek pętli
   {
     potega=potega*2;
     wykladnik=wykladnik+1;
   }                                   //koniec pętli
  printf("Najmniejsza potęga 2 większa od %d  jest %d", n, potega);
  getchar();
  return 0;
}
```

94

Funkcje matematyczne: nagłówek <math.h>

sin(x) - sinus x (kąt podany w radianach),
cos(x) - cosinus x (kąt podany w radianach),
tan(x) - tangens x (kąt w radianach),
asin(x) - arcus sinus x (wynik w przedziale [-π /2, π /2]),
acos(x) - arcus cosinus x (wynik w przedziale [0 , π]),
atan(x) - arcus tangens x (wynik w przedziale [-π /2, π /2]),
atan2(x,y) - arcus tangens x/y (wynik w przedziale [-π, π]),
sinh(x) - sinus hiperboliczny x,
cosh(x) - cosinus hiperboliczny x,
tanh(x) - tangens hiperboliczny x,

95

Funkcje matematyczne, cd. nagłówek <math.h>

exp(x) - funkcja wykładnicza: e^x,
log(x) - logarytm naturalny: ln(x), x > 0,
log10(x) - logarytm o podstawie 10: $\log_{10}(x)$, x > 0,
pow(x,y) – potęgowanie: x^y, błąd zakresu wystąpi gdy:
x=0 i y≤0 lub x<0 i y nie jest całkowite,
sqrt(x) - pierwiastek kwadratowy: $\sqrt{x}, gdzie: x \geq 0$
fabs(x) - wartość bezwzględna |x|
ceil(x) - najmniejsza liczba całkowita nie mniejsza niż x,
wynik typu double,
floor(x) - największa liczba całkowita nie większa niż x,
wynik typu double,

96

Podstawowe typy danych, cd.

- W języku C występują cztery podstawowe typy danych: char, int, float, double.
- Podstawowe typy danych można uzupełnić kwalifikatorami:

  signed, unsigned, short, long, const, volatile.

  Przykłady:

signed int x;	lub	int x;
short int y;	lub	short y;
long int z;	lub	long z;
unsigned int b;	lub	unsigned b;
unsigned short int b;	lub	unsigned short b;

97

Podstawowe typy danych - kwalifikatory

Możliwe kombinacje typów podstawowych i kwalifikatorów

Kwalifikator\typ	char	int	float	double
signed	x	x		
unsigned	x	x		
short		x		
long		x		x
const	x	x	x	x
volatile	x	x	x	x

98

Rozmiar i zakres danych dla maszyn 32-bitowych (16-bitowych)

Nazwa typu	Zakres	Rozmiar w bajtach
char	-128..127	1
unsigned char	0..255	1
short	-32768..32767	2
unsigned short	0..65535	2
int	-2^{31} ...2^{31} -1 (-2^{15} ...2^{15} -1)	4 (2)
unsigned	0...2^{32} -1 (0...2^{16} -1)	4 (2)
long	-2^{31} ...2^{31} -1	4
unsigned long	0...2^{32} -1	4
float	$3.403*10^{38}$...$1.175*10^{-38}$	4
double	$1.798*10^{308}$...$2.225*10^{-308}$	8

Kwalifikatory const i volatile

- Kwalifikatory const i volatile mogą wystąpić w deklaracji przed każdym z typów.
- Kwalifikator const (stały) oznacza, że wartość zadeklarowanego obiektu nie może się zmieniać – deklaracja stałej.
- Kwalifikator volatile (ulotny) oznacza, że wartość zadeklarowanego obiektu może się często zmieniać i przy każdym odwołaniu do niego kompilator powinien sprawdzić zawartość przydzielonych obiektowi komórek pamięci.

Przykłady

```
const float pi = 3.14; // definicja i inicjalizacja

volatile int temperatura;
volatile float stan_miernika;
printf("Bieżąca temperatura = %d oC, a ciśnienie =
                %f Pa \n", temperatura, stan_miernika)
```


Stałe symboliczne

Definicja stałej symbolicznej:

#define nazwa tekst_zastępujący

Dyrektywa **#define** przyporządkowuje nazwie symbolicznej określony ciąg znaków.

Przykłady:

```
#define PI  3.14
#define LICZBA_MASZYN 4
#define LICZBA_SILNIKOW 4
```

Typ void

Typ **void** (*ang. próżny*) może pojawić się w deklaracjach typów pochodnych.

Przykłady:

```
int funkcja (void); // funkcja nie ma parametrów

void funkcja (int); // funkcja nie zwraca
                    // żadnej wartości
```


Typ tablicowy

Typ tablicowy jest **typem pochodnym**. Tablice można tworzyć z:

- obiektów typu podstawowego,
- obiektów typu wyliczeniowego (enum),
- innych tablic,
- obiektów typu zdefiniowanego przez użytkownika (klasy),
- wskaźników,
- ze wskaźników do pokazywania na składniki klasy

Typ tablicowy

Definicja tablicy

typ_elementow nazwa_tablicy [wymiar];

Indeksy tablic w języku C
zawsze zaczynają się od zera !

Przykłady:

int a [10];
char tab_z[20];
float Ftab[15];

Przykłady odwołania do elementów tablicy:

a[0] a[i] a[k +1] a[9] tab_z[19] Ftab[14]

Przykład 3

```
# include <stdio.h>
/* program ilustrujacy wpisywanie danych do tablicy
    i wypisywanie ich z tablicy */

int main (void)
{
int i;
int tab [5];
   for (i =0; i < 5; i++)   tab [i] = 10*i;  // wpisanie danych
   for (i =0; i <5; i++)
        printf("Elemet %d ma wartosc %d\n", i, tab[i]);
return 0;
}
```

Inicjalizacja tablic

W przypadku tablic stosowana jest **inicjalizacja zbiorcza**.

Przykłady:

long t[4] = { 72896, 15, 6, 70800};

float ta[4] = {1.2, 5.0}; //dwa ostatnie elementy = 0.0

int tt[ ] = {3, 6, 19}; // definicja tablicy z 3 elementów

Tablice znakowe

char tekst [20]; // definicja tablicy 20 elementowej

Inicjalizacja tablicy znakowej

char tekst [20] = {"Alpy" };

char tekst [20] = { ‘A’, ‘l’, ‘p’, ‘y’ };

char tekst [ ] = {"Tatry" }; // rozmiar tablicy: 6

char tekst [ ] = { ‘T’, ‘a’, ‘t’, ‘r’, ‘y’ }; // rozmiar tablicy 5

Przykład 14

Problem:

Napisz program, który umożliwia wpisanie 10 liczb do tablicy, a następnie wyznacza największą z nich i wypisuje na ekranie.

Dyskusja:

Dane wejściowe: 10 liczb

Dane wyjściowe: maksimum

Algorytm:

- Wczytać w pętli *„for”* 10 liczb do tablicy.
- Wyznaczyć największą liczbę:
  - pierwszą liczbę z tablicy wpisać do zmiennej maksimum,
  - stosując pętlę for przeglądać kolejne elementy tablicy i jeśli któryś będzie większy do zmiennej maksimum, to należy go wpisać do zmiennej maksimum.
- Wypisać wynik na ekranie.


```
#include <stdio.h>   /*Program wyznacza  największą liczbę*/

int main() {
   int i;
   float max, tablica[10];
   printf("Program do wyznaczania  największej liczby w tablicy\n");
   for(i=0; i<10; i++) {
         printf("Podaj liczbe %d: ", i+1);
         scanf("%f", &tablica[i]);
   }
   max=tablica[0];
   for(i=1; i<10; i++)
         if(tablica[i]>max) max=tablica[i];
   printf("Największa liczba jest równa %10.5f\n", max);
   getchar();
   return 0;
}
```

Przykład 15

Napisz program, który zlicza liczbę wystąpień każdej cyfry, każdego białego znaku oraz wszystkich innych znaków dla danych wprowadzonych z klawiatury.

Program

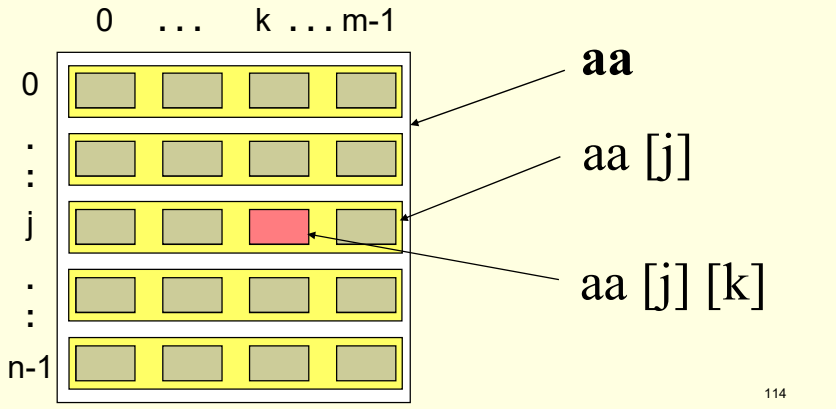
```
# include <stdio.h>
/* program zliczający cyfry, białe znaki i inne*/

int main ()
{
  int c, i, nwhite, nother;
  int ndigit [10];
  nwhite = nother =0;
  for ( i =0; i < 10; ++i) ndigit [i] = 0; //zerowanie tablicy
```

```
while ((c = getchar ()) != EOF)
        if (c>= '0' && c<='9')
                ++ndigit [c-'0'];
        else if (c == ' ' || c == '\n' || c == '\t')
                ++nwhite;
        else ++nother;
  printf("cyfry =");
  for (i =0; i <10; ++i)     printf("%d", ndigit[i]);
  printf(",biale znaki = %d, inne = %d\n",nwhite,nother);
  getchar();
  return 0;
}
```


Tablice wielowymiarowe

int aa [n] [m]; // definicja tablicy dwuwymiarowej
int aaa [n] [m] [p]; // definicja tablicy trójwymiarowej



Inicjalizacja tablic wielowymiarowych

Przykłady:

```
short aa[2][3] = {3, 7, 13, 8, 50, 71};
```

lub

```
short aa[2][3] = {
                 {3, 7, 13 },
                 {8, 50, 71}
                };
```

// inicjalizacja
// zbiorcza

separator

terminator

są równoważne:

```
short aa[2][3];
aa[0][0] = 3;  aa[0][1] = 7;    aa[0][2] = 13;
aa[1][0] = 8;  aa[1][1] = 50;   aa[1][2] = 71;
```


Inicjalizacja tablic wielowymiarowych

Przykłady:

```
short aa[ ][3] = {                    // poprawnie
                 {3, 7, 13 },
                 {8, 50, 71}
                };
short aa[ ][3] = {3, 7, 13, 8, 50, 71}; // poprawnie
int bbb[ ][2][2] = {0, 7, 8, 256, 90, 8, 675, 1};
                                      // poprawnie
int bbb[ ] [ ] [2] = {0, 7, 8, 256, 90, 8, 675, 1}; // błąd
```

poprawnie

poprawnie

błąd

Przykład 16

Napisz program, który wpisuje do tablicy imiona i nazwiska dla grupy 15 studentów, a następnie wyświetla je na ekranie w kolejności:

- imię nazwisko
- lub odwrotnie.

117

```
# include <stdio.h>
# include <stdlib.h>
/* program ilustrujący wpisywanie (wypisywanie) danych
   do (z) tablicy wielowymiarowej */
int main (void)
{
   char c;
   char tab [15] [2] [20];
   int n,i;
   do {
   printf("Podaj liczbe studentow (n<16): ");
   i=scanf("%d", &n);
   fpurge(stdin); //stdio.h - czyszczenie bufora (GCC, UNIX)
   } while (i==0);
   for (i =0; i < n; i++)
   {
```

```
        printf(”Podaj imię %d studenta: ”, i + 1 );
        scanf("%19s", & tab [i] [0]);          // wpisanie danych
        scanf("%*s”);
        printf(”  Podaj nazwisko %d studenta: \n”, i + 1);
        scanf("%19s", & tab [i] [1]);          // wpisanie danych
        scanf("%*s”); printf(" \n");
    }
    printf(”Czy wypisać najpierw imię, potem nazwisko? t/n”);
    fpurge(stdin);
    c=getchar();
    printf("\n");
    if (c==‘t’ || c==‘T’)
        for (i = 0 ; i <n; i++)
        printf(”%d)  %s  %s\n”, i+1, tab [i] [0], tab [i] [1]);
    else for (i = 0 ; i <n; i++)
        printf(”%d)  %s  %s\n”, i+1, tab [i] [1], tab [i] [0]);
    return 0;
}
```

119

© Instytut Informatyki Stosowanej, Politechnika Łódzka

Funkcje

- Funkcja w języku C jest równoważna podprogramowi (funkcji lub procedurze) w Pascalu.
- Funkcja jest to fragment kodu, który stanowi logiczną całość i jest wywoływany z innego miejsca w programie.
- Każdy program w języku C to zbiór funkcji, najważniejsza z nich to **main** – musi wystąpić w każdym programie.

120

Prototyp funkcji

- Deklaracja funkcji w języku C występuje przed funkcją **main** i nazywana jest **prototypem funkcji**.
- Prototyp funkcji musi być zgodny z jej definicją i wywołaniem.

Prototyp funkcji:

typ-wyniku nazwa-funkcji (lista typów parametrów);

Przykład:

```
int potega (int, int);  // prototyp funkcji
```


Definicja funkcji

typ-wyniku nazwa-funkcji (deklaracje parametrów formalnych) ***// nagłówek***

{

deklaracje ***// deklaracje zmiennych lokalnych***

ciąg instrukcji ***ciało funkcji***

}

Definicja funkcji

Przykład: parametry formalne

```
int potega (int liczba, int wykladnik)  // nagłówek
{
   int i, p;            // deklaracja zmiennych lokalnych

   p=1;
   for (i=1; i<=wykladnik; ++i)
        p = p * liczba;
   return p;            // przekazanie wyniku
}
```

wyrażenie

Przykłady:

```
float z (float x, float y)
{  /*Funkcja rzeczywista dwóch zmiennych rzeczywistych*/
     float z;
     z = x*y- sin(x+y);              //wymagany plik math.h
     return z;
}

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

float Fx (float x)
{ /*Funkcja rzeczywista  zmiennej rzeczywistej*/
    float fx;
    if ((x <= 0 ) || (x >= 5) &&(x <= 8))
       fx = 0;
    else fx = sqrt(x*(x-5)*(x-8));      //wymagany plik math.h
    return fx;
}
```

Przykłady programów z funkcjami

Przykład 17

Problem:

*Oblicz pole prostokąta o bokach **a** i **b** korzystając z funkcji obliczającej pole prostokąta.*


```
#include <stdio.h>

float Pole(float a, float b);          // deklaracja funkcji

int main()
{
 float b1, b2;
 printf("Podaj dlugosci bokow prostokata: ");
 scanf("%f %f", &b1, &b2);
 printf("Pole prostokata wynosi: %.4f\n", Pole(b1, b2));
 getch();
 return 0;
}

float Pole(float a, float b)          //definicja funkcji
{
 float pole;
 pole=a*b;
 return pole;
}
```

Przykład 17

Napisz program, który oblicza kwadraty i sześciany kolejnych liczb naturalnych od 1 do 10 i wyświetla je na ekranie.


```
/* Program oblicza kwadraty i sześciany liczb od 1 do10 */
# include <stdio.h>
int potega (int, int);  // prototyp        deklaracja funkcji

void main(void)
{
   int i; // deklaracja zmiennych lokalnych
   for (i=1; i<=10; ++i)                          wywołanie funkcji
        printf("%d %d %d\n", i, potega(i,2), potega(i,3));
   }

int potega (int liczba, int wykladnik)  // nagłówek funkcji
{                                                  definicja funkcji
   int i, p;      // deklaracja zmiennych lokalnych
   p=1;
   for (i=1; i<=wykladnik; ++i)  p = p*liczba;
   return p;   // przekazanie wyniku do funkcji main
}
```

Przykład 18

Napisz program, który oblicza potęgę dla dowolnej liczby naturalnej i wykładnika naturalnego oraz wyświetla wynik na ekranie.

Zastosuj algorytm binarny potęgowania.


```
/* program do wyznaczania potegi */
#include <stdio.h>

int potega1(int, int);

int main(void)
{
        int w,x,n;
        printf("Program do potęgowania liczb calkowitych
                        z wykladnikiem naturalnym\n\n");
        printf("Wpisz liczbe potegowana oraz wykladnik ");
        scanf("%d %d", &x, &n);
        w=potega1(x,n);
        printf("Liczba %d do potegi %d to %d\n", x,n,w);

        return 0;
}
```

```
int potega1(int x, int n) //potegowanie binarne
{
        int z,m,y;
        z=x; y=1; m=n;
        while (m!=0)
        {
                // {G: x^n = y*z^m and m>0}
                if (m%2==1) y=y*z;
                m=m/2;
                z=z*z;
        }
        // {y = x^n}

        return y;
}
```


Przykład 19

1. Napisz program, który wczytuje z klawiatury krótki tekst (1 linijka), odwraca kolejność znaków tekstu i drukuje odwrócony tekst na ekranie.
2. Wyodrębnij w programie funkcję, która odwraca tekst w miejscu.

```
#include <stdio.h>
#include <string.h>
/* program do odwracania tekstu w miejscu */
int main(void)
{
   int  c, i, j, n;
   char s[81];
   gets(s);
   n=strlen(s);  // zwraca długość tekstu bez '\0'
   for (i=0, j=n-1; i<j; i++, j--)
   {
        c=s[i];   s[i]=s[j];   s[j]=c;
   }
   printf(" n= %d\n",n);
   printf(" %s\n",s);
   return 0;
}
```


© Instytut Informatyki Stosowanej, Politechnika Łódzka

Funkcja „*reverse*”

```
# include <string.h>
/* funkcja odwracająca tekst w miejscu */

void reverse(char s[ ])
{
   int c, i, j;
   for (i=0, j=strlen(s)-1; i<j; i++, j--)
   {
        c=s[i];   s[i]=s[j];   s[j]=c;
   }
}
```

```c
/* program odwracający tekst w miejscu */
#include <stdio.h>
#include <string.h>

void reverse(char s[ ]); //prototyp funkcji- deklaracja

int main(void)
{
   char stab[81];
   scanf("%80s", &stab);
   fflush(stdin);
   reverse(stab);
   printf("%s\n",stab);
   return 0;
}
```

parametr formalny

parametr aktualny - argument


```c
/* funkcja odwracająca tekst w miejscu */
void reverse(char s[ ]) //definicja funkcji
{
   int c, i, j;
   for (i=0, j=strlen(s)-1; i<j; i++, j--)
   {
        c=s[i];   s[i]=s[j];   s[j]=c;
   }
}
```

Uwaga!
Po zakończeniu działania funkcji **reverse** **zawartość tablicy s ulegnie zmianie !!!**

Przykład 20

Problem:

Znajdź największy wynik w każdej z 10 serii po 50 pomiarów.

Dyskusja:

Dane wejściowe: 10 serii po 50 pomiarów

Dane wyjściowe: maksymalne wyniki w każdej serii

Algorytm:

W programie wyodrębniona będzie funkcja do wczytywania pomiarów do tablicy oraz funkcja do wyznaczania największej wartości w każdej serii.


```
#include <stdio.h>
#define LSERII 10
#define LPOM 50

typedef float seria[LPOM];  //pomiary w jednej serii
typedef seria serie[LSERII];  //wszystkie serie

void CzytajSerie(serie tab);  //deklaracja funkcji CzytajSerie
float Maximum(seria tab); //deklaracja funkcji Maximum

int main() { // część główna programu
 int nrs;
 float maxpom[LSERII];
 serie pomiary;

 CzytajSerie(pomiary);
 printf("\n\n");

 for(nrs=0; nrs<LSERII; nrs++) {
  maxpom[nrs]=Maximum(pomiary[nrs]);
  printf("Największy pomiar w serii %d wynosi %8.3f\n", nrs, maxpom[nrs]);
 }
 getch();
 return 0;
} // koniec części głównej programu
```

```
void CzytajSerie(serie tab) {
/*Funkcja do wczytania serii danych do tablicy*/
 int nrw, nrp;
 for(nrw=0; nrw<LSERII; nrw++) { //czytaj kolejne serie
  printf("Seria %d\n", nrw+1);
  for(nrp=0; nrp<LPOM; nrp++) { //czytaj kolejne pomiary w serii
    printf("Podaj pomiar %d: ", nrp);
    scanf("%f", &tab[nrw][nrp]);
  }
 }
}
float Maximum(seria tab) {
 /*Funkcja do wyznaczania maksymalnej wartości w tablicy*/
 int n;
 float max;
 max=tab[0];
 for(n=1; n<LPOM; n++) {
  if(tab[n]>max)
   max=tab[n];
 }
 return max;
}
```

139

© Instytut Informatyki Stosowanej, Politechnika Łódzka

Przykład 21

Problem: *Napisz program tablicowania funkcji: $f(x) = x^n$, dla $x > 0$ w danym przedziale $< x_p , x_k >$ równomiernie dla 200 wartości.*

Algorytm: *Ponieważ problem sprowadza się do tablicowania funkcji, należy wykorzystać zmienną typu tablicowego do zapisu wyniku tablicowania oraz ułożyć odpowiedni, uniwersalny podprogram tablicowania i zdefiniować funkcję w postaci podprogramu. Ponieważ $x > 0$, można wykorzystać zależność:*

$$x^n = e^{\ln(x^n)} = e^{n\ln(x)}$$

140

```
/*Program do tablicowania funkcji*/

#include <stdio.h>
#include <math.h>

/*definicje stałych*/
#define LP 200
#define N 1.5

/*definicja nowego typu zmiennych*/
typedef float tab[LP][2];

/*deklaracje funkcji*/
float f(float x, const float n);
void tablicuj(float xpocz, float xkon, float krok, tab t);
```



```
int main() {                          /*część główna programu*/
 int i;
 tab fx;
 float xp, xk, dx;

 printf("Program tablicowania funkcji\n");
 printf("Podaj kolejno początek i koniec przedziału ");
 printf("zmiennej x  (x1, x2, dla x > 0 i x1<x2)\n");
 scanf("%f %f", &xp, &xk);

 if((xp>0)&&(xk>0)&&(xk>xp)) {
  dx=(xk-xp)/(LP-1);
  tablicuj(xp, xk, dx, fx);  // wywołanie funkcji tablicuj
  printf("\n\nTablicowanie funkcji f(x):\n");
  printf("x: \t\t f(x):\n");
  for (i=0; i<LP; i++)
   printf("%f \t\t %5.8f\n", fx[i][0], fx[i][1]);
 }
 else printf("Zly przedzial tablicowania\n");

 getch();
 return 0;
}
```

/*Definicje funkcji*/

```
float f(float x, const float n) {
 float y;
 y=exp(n*log(x));
 return y;
}
void tablicuj(float xpocz, float xkon, float krok, tab t) {
 int i;
 float x;

 i = 0;
 x = xpocz;

 while(x<=xkon) {
  t[i][0]=x;
  t[i][1]=f(x,N);
  x=xpocz+krok*i;
  i=i+1;
 }
 if (x<(xkon+krok)) {
  t[i-1][0]=xkon;
  t[i-1][1]=f(xkon,N);
  }
}
```


143

Instrukcje sterujące

- Język C zawiera cały zestaw instrukcji sterujących, które umożliwiają pisanie programów zgodnych z ideą programowania strukturalnego.
- Ich działanie polega na przełączeniu sterowania między występującymi w programie instrukcjami złożonymi i wszystkie razem pozwalają pisać zwięzły i logicznie skonstruowany kod.

144

Instrukcje sterujące

- Instrukcja warunkowa **if**
- Instrukcja **while**
- Instrukcja **do … while**
- Instrukcja **for**
- Instrukcja **switch**
- Instrukcja **break**
- Instrukcja **continue**
- Instrukcja **goto**

Instrukcja *if* – niejednoznaczność składni

if (wyrażenie) instrukcja 1 [else instrukcja 2]

Przykłady:

1.
```
if (a!=0 && b!=0)
    if (a>b)
      x = 4*a;
    else
      x = 4 *b;
```

2.
```
if (a!=0 && b!=0) {
    if (a>b)
    x = 4*a;
  }
  else
    x = 4 *b;
```

Opcja else odnosi się zawsze do ostatniego if !

Konstrukcja *else if*

```
if (wyrażenie)  instrukcja 1
else  if (wyrażenie) instrukcja 2
      else if (wyrażenie) instrukcja 3
      …
            else instrukcja n
```

Konstrukcja else if umożliwia wybór wielowariantowy.

147

Przykład:

```
if (dd == 1)
   printf (" Poniedziałek\n");
else if (dd == 2)
   printf (" Wtorek\n");
else if (dd == 3)
   printf (" Środa\n");
else if (dd == 4)
   printf ("Czwartek\n");
else if (dd == 5)
   printf ("Piatek\n");
else if (dd == 6)
   printf ("Sobota\n");
else if (dd == 7)
   printf ("Niedziela\n");
else printf ("BŁĄD\n");
```

148

Instrukcja *switch*

```
switch (wyrażenie)
{
   case wyrażenie-stałe: instrukcje
   …
   case wyrażenie-stałe: instrukcje
   default: instrukcje
}
```


Przykład:

```
switch (dd)
{
   case 1: printf (" Poniedziałek\n");
           break;
   case 2: printf (" Wtorek\n");
           break;
   case 3: printf (" Środa\n");
           break;
   case 4: printf ("Czwartek\n");
           break;
   case 5: printf ("Piatek\n");
           break;
   case 6: printf ("Sobota\n");
           break;
   case 7: printf ("Niedziela\n");
           break;
   default: printf ("BŁĄD\n");
           break;
}
```

Przykład 22

Napisz program, który zlicza liczbę wystąpień każdej cyfry, każdego białego znaku oraz wszystkich innych znaków dla danych wprowadzonych z klawiatury.

Zastosuj instrukcję *switch*.

Program 22

```
# include <stdio.h>
/* program zliczający cyfry, białe znaki i inne*/

int main (void)
{
   int c, i, nwhite = 0, nother = 0;
   int ndigit [10] = {0};    // zerowanie tablicy
```

```
while ((c = getchar ()) != EOF) {
    switch (c)
    { case '0': case '1': case '2': case '3': case '4':
      case '5': case '6': case '7': case '8': case '9':
          ndigit [c-'0'] ++;
          break;
      case ' ': case '\n': case '\t':
          nwhite ++;
          break;
      default:  nother ++;
                break;
    }
}
```



```
printf("Liczba wystapien cyfr:\n");
for (i =0; i <10; ++i)
        printf("%d - %d,\n", i, ndigit[i]);
printf("biale znaki = %d\n inne = d\n",nwhite,nother);
return 0;
}
```

Przykład 23

Problem: Zbadać, czy wczytany znak jest samogłoską, spółgłoską polską lub jedną z liter obcych: q,x,v. Powtarzaj sprawdzanie liter, dopóki użytkownik sobie tego życzy.

Dyskusja:

Dane wejściowe: znak

Dane wyjściowe: znak i informacja o rodzaju znaku

Do badania znaku przy tylu możliwościach najlepiej będzie wykorzystać instrukcję wyboru switch-case.

Algorytm:

1. Wczytać znak i wypisać go na ekranie.
2. Sprawdzić, czy jest on samogłoską, spółgłoską, literą obcą, czy też żadną z nich.
3. Wypisać odpowiednią informację.
4. Powtórzyć kroki 1-3 jeśli użytkownik wpisze ‘t' lub ‘T'.

155

```
#include <stdio.h>                                  /*Program rozpoznaje litery*/
int main() {
   char znak;                                       //litera
   printf("Program do rozpoznawania liter\n");
   do {                                             //początek pętli do-while
    printf("\nPodaj znak: ");
    znak = getch();
    printf("\nPodales znak: %c", znak);
    switch(znak) {                                  //początek instrukcji switch
     case 'a': case 'e': case 'i': case 'o': case 'u': case 'y':
         printf("\n samogloska\n");
         break;
     case 'b': case 'c': case 'd': case 'f': case 'g': case 'h': case 'j': case 'k': case 'l':
     case 'm': case 'n': case 'p': case 'r': case 's': case 't': case 'w': case 'z':
         printf("\n spolgloska\n");
         break;
     case 'q': case 'v': case 'x':
         printf("\n spolgloska obca\n");
         break;
     default:
         printf("\n to nie byla litera\n");
    }                                               //koniec instrukcji switch
    printf("\nCzy kontynuowac (t/n)? ");
    znak=getchar();
   }while((znak=='T')||(znak=='t'));                //koniec pętli do-while
   return 0;
}
```

156

Instrukcja *break*

Instrukcja *break* stosowana jest do przerywania działania pętli i instrukcji *switch*.

Przykład:

```
int i, m;
int dlugosc_linii = 5;
for (i=0; i < 4; i++) {
    for (m=0; m<10; m++) {
        printf("*");
        if (m > dlugosc_linii) break;
    }
    printf("\nKontynuujemy zewnętrzną pętlę dla i = %d\n", i);
}
```


Instrukcja *goto* i etykiety

goto nazwa_etykiety;

Instrukcja *goto* stosowana jest do przerywania działania wielokrotnie zagnieżdżonych pętli.

Instrukcja goto nie jest zalecana !!!

Przykład:

```
for (…)
    for (…) {
        …
        if (niepowodzenie)
            goto error; // skocz do obsługi błędów
    }
…
error: printf("Blad");
```

Przykład 24

Napisz program, który sprawdza, czy liczba wpisana z klawiatury jest liczbą pierwszą.


```
/* program do wyznaczania liczb pierwszych */
#include <stdio.h>
#include <math.h>

int pierwsze (int);

int main (void)
{
        int n,p;
        do {
        printf ("Wpisz liczbe naturalna, n>0, n= ");
        scanf ("%d", &n);
        } while (n<= 0);
        p = pierwsze (n);
        if (p) printf("\nLiczba %d nie jest liczba pierwsza\n", n);
        else printf("\nLiczba %d jest liczba pierwsza\n", n);
return 0;
}
```

```
int pierwsze (int n)
{
    int i;
    int p=0;

    for (i=2; i <= sqrt (n); i++)
        if (n % i == 0)
        {
            p=1;
            break;
        }
    return p;
}
```


Instrukcja *continue*

Instrukcja *continue* powoduje przerwanie bieżącego i wykonanie od początku następnego kroku zawierającej ją pętli.

Przykład:

```
for (i=0; i<n; i++) {
    if (a[i] < 0) // pomiń element ujemny
        continue;
    … // przetwarzaj element nieujemny
}
```

Organizacja programu w języku C

```
//plik źródłowy prog1.c
# include <stdio.h>
# include ”mojprog.h”

void funkcja3 (void)
{ …..
}
void main (void)
{
   funkcja1 ( );
   funkcja2 ( );
}
```

```
//plik źródłowy prog2.c
# include <stdio.h>
# include ”mojprog.h”

void funkcja1 (void)
{ …..
}
void funkcja2 (void)
{
   funkcja3 ( );
}
```


Zasięg funkcji i zmiennych

W języku C występują cztery klasy pamięci, identyfikowane za pomocą kwalifikatorów:

- ✓ auto
- ✓ register
- ✓ static
- ✓ extern

Zasięg funkcji i zmiennych

static

auto

extern

register

```
//plik prog1.c
……
int b = 3; //zm. globalna

void main (void)
{
    int x; //zm. lokalna
    funkcja1 ( );
}
void funkcja2 (void)
{ …..
    char z; //zm. lokalna
}
```

```
//plik prog2.c
……
extern int b; //zm. zewnętrzna

void funkcja1 (void)
{
  register int x;//zm.wewnętrzna
  funkcja2 ( );
}
```


Zasięg funkcji i zmiennych

definicja

deklaracja

```
//plik prog1.c
……
int b = 3; //zm. globalna

void main (void)
{
    int x; //zm. lokalna
    funkcja1 ( );
}
void funkcja2 (void)
{ …..
    char z; //zm. lokalna
}
```

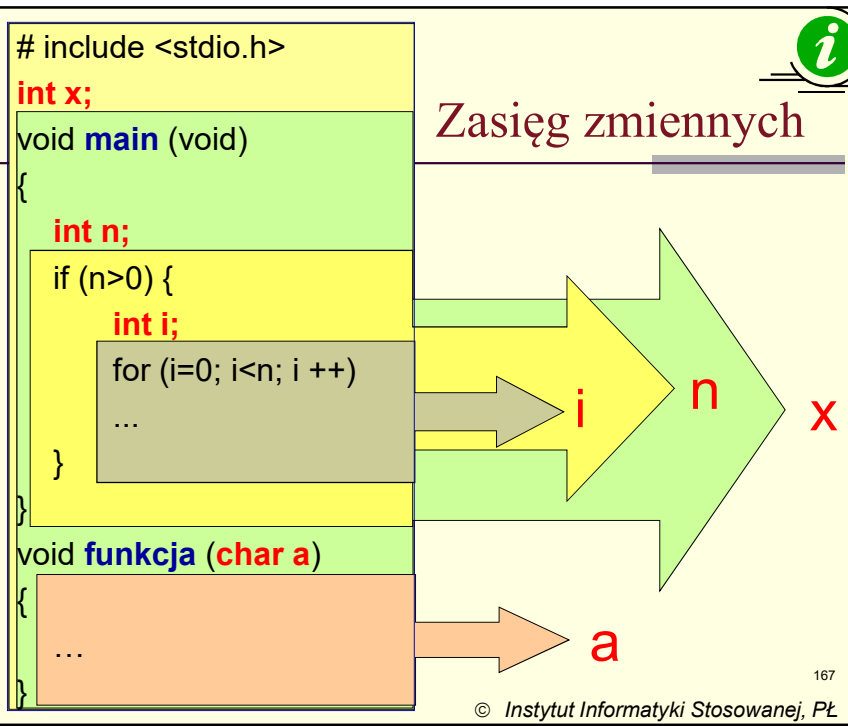
```
//plik prog2.c
……
extern int b; //zm. zewnętrzna

void funkcja1 (void)
{
  register int x;//zm.wewnętrzna
  funkcja2 ( );
}
```

obiekty zewnętrzne

obiekty wewnętrzne

Zasięg zmiennych



167

Deklaracje i definicje

- **Deklaracja** określa nazwę obiektu, jego typ i klasę, ale nie przydziela mu miejsca w pamięci.
- **Definicja** to deklaracja, która dodatkowo przydziela obiektowi miejsce w pamięci.
- Deklaracje funkcji (prototypy) oraz deklaracje zmiennych zewnętrznych można umieszczać w **plikach nagłówkowych**.

W plikach nagłówkowych nie wolno umieszczać definicji zmiennych !

168

Efekty przesłaniania

```
# include <stdio.h>
int x, y;

void main (void)
{
   funkcja ( );
}
void funkcja (double x)
{
   double y;
   …
}
```

y x

x

y

Klasa static

Zmiennymi statycznymi są wszystkie zmienne zewnętrzne oraz te zmienne wewnętrzne, które zostały zdefiniowane jako „*static*”.

Przykład:

```
void suma_zdarzen (void)
{
   static int suma = 1;
   suma = suma +1;
}
```

Zasady programowania strukturalnego

1. Programy są projektowane metodą zstępującą.
   - Główne funkcje odpowiedzialne za realizację zadania są wywoływane z najwyższego poziomu programu.
   - Każda z wywoływanych funkcji uszczegóławia rozwiązanie i jeśli jest to konieczne wywołuje kolejne funkcje.
2. Funkcje są krótkie i odpowiedzialne za wykonanie jednego logicznie określonego zadania.

Zasady programowania strukturalnego

3. Każda funkcja jest maksymalnie niezależna od pozostałych.
   - Informacje wymieniane są między funkcjami za pomocą argumentów i generowanych przez funkcje wartości.
4. Unika się skoków bezwarunkowych.

Przykład 1

Dany jest ciąg $x_1, x_2, \ldots, x_n$ ($n < 201$) o wartościach całkowitych. Uporządkować ciąg niemalejąco (posortować zbiór).

Ciąg $\{ x_n \}$ jest uporządkowany niemalejąco, jeśli dla każdego $i < n$ zachodzi: $x_i \leq x_{i+1}$.

Algorytm bąbelkowy

- Przestawiając sąsiednie wyrazy ciągu kolejno dla $i=1,2,\ldots,n-1$ spełniające nierówność $x_i > x_{i+1}$ spowodujemy umieszczenie największego wyrazu ciągu na pozycji ostatniej.
- Podobną serię przestawień możemy zastosować do ciągu pomniejszonego o wyraz ostatni. W ten sposób dwa ostatnie wyrazy ciągu będą już na właściwych pozycjach. Kontynuując takie serie przestawień dla coraz mniejszych ciągów doprowadzimy w końcu do uporządkowania wszystkich wyrazów ciągu $\{x_n\}$.

Algorytm bąbelkowy, cd.

- Po *k* przebiegach ostanie *k* pozycji zajmować muszą właściwe wyrazy ciągu (wytłuszczone). Pozycje poprzedzające niekoniecznie musiały być już uporządkowane.
- Metoda ta jest zalecana do sprawdzania czy ciąg jest uporządkowany oraz porządkowania ciągu, jeśli spodziewamy się niewielkiej liczby przestawień, tj. gdy uporządkowanie ciągu zostało zakłócone tylko na kilku pozycjach.

Przykład

x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]	x[8]	uwagi
6	2	7	1	8	3	9	5	ciąg oryginalny
2	6							zamiana x[1] i x[2]
		1	7					zamiana x[3] i x[4]
				3	8			zamiana x[5] i x[6]
						5	9	zamiana x[7] i x[8]
2	6	1	7	3	8	5	**9**	po I przebiegu

x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]	x[8]	uwagi
2	6	1	7	3	8	5	**9**	po I przebiegu
	1	6						zamiana x[2] i x[3]
			3	7				zamiana x[4] i x[5]
					5	8		zamiana x[6] i x[7]
2	1	6	3	7	5	**8**	**9**	po II przebiegu

177

x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]	x[8]	uwagi
2	1	6	3	7	5	**8**	**9**	po II przebiegu
1	2							zamiana x[1] i x[2]
		3	6					zamiana x[3] i x[4]
				5	7			zamiana x[5] i x[6]
1	2	3	6	5	**7**	**8**	**9**	po III przebiegu
			5	6				zamiana x[4] i x[5]
1	2	3	5	**6**	**7**	**8**	**9**	po IV przebiegu

178

Program 1

```
// plik nagłowkowy tenprog.h
/* program ilustrujący organizację programu w
   języku C */

void sort_b (int, int[]); //prototypy funkcji
void druk (int, int[]);
```



```
// plik źródłowy sortowanie.c
/* program ilustrujący organizację programu w
   języku C */

# include <stdio.h>
# include <stdlib.h>
# include "tenprog.h"


void sort_b (int n, int a[ ])
{
   int p,i,aa;
```

```
    do {
        p=0;
        for (i=0; i<n-1; i++) {
            if (a[i]>a[i+1])
            {
                aa=a[i];
                a[i]=a[i+1];
                a[i+1]= aa;
                p=1; //zamiana
            }
        }
    } while (p!=0);
}
```



```
// plik źródłowy program.c
/* program ilustrujący organizację programu w
   języku C */

# include <stdio.h>
# include <stdlib.h>
# include "tenprog.h"

int main (void)
{
   int n,j;
   int a[200];
   char c;
```

```
system("cls"); //stdlib.h - czyszczenie ekranu
fflush(stdin); //stdio.h - czyszczenie bufora
printf("Program do sortowania elementów
                        całkowitych ciągu\n\n");
printf("Podaj liczbe elementów ciągu n<201 ");
scanf("%d", &n);
printf("\nWpisz kolejne elementy ciągu \n\n");
for (j=0; j<n; j++) {
        printf("a[%d] = ", j+1);
        scanf("%d", &a[j]);
}
```



```
system ("cls");
printf ("Ciąg dany:\n\n");
druk (n,a);
printf ("\n\n");
sort_b (n,a);
printf ("Ciąg posortowany niemalejąco:\n\n");
druk (n,a);
printf ("\n\n");
c = getchar();

return 0;
}
```

```
void druk (int k, int b[])
{
   int i;

   for (i=0; i<k; i++) {
        printf("  %6d", b[i]);
        if ((i+1)%10==0) printf("\n");
   }
}
```


Algorytmy iteracyjne

1. **Funkcja silnia - n!**
   (dla argumentów całkowitych, nieujemnych)
   (a) **0! = 1**,
   (b) **n! = 1·2·3·…·(n-1)·n,** dla n>0

Przykład 1

```
int i,s,n;
i=0; s=1;
while (i<n) {
    i=i+1;    s = s*i;
}
```

Algorytmy rekurencyjne

1. **Funkcja silnia - n!**
   (dla argumentów całkowitych, nieujemnych)
   (a) **0! = 1**,
   (b) jeśli n>0 to **n! = n (n-1)!**

Przykład 2

```
int silnia (unsigned int n)
{   int s=1;
    if (n>0) s = n* silnia (n-1);
    return s;
}
```


Rekurencja

- W języku C funkcje mogą być wywoływane **rekurencyjnie**, tzn. funkcja może wywoływać siebie zarówno bezpośrednio, jak i pośrednio.
- Funkcja wywołana rekurencyjnie otrzymuje nowy komplet wszystkich zmiennych automatycznych, niezależny od zmiennych z poprzedniego wywołania funkcji.
- Rekurencja nie przynosi oszczędności pamięci, nie przyspiesza działania funkcji, ale ułatwia zapisanie i zrozumienie niektórych algorytmów.

Sortowanie szybkie "quick sort"

- Jest to metoda sortowania przez podział.
- Metoda ta zapewnia efektywność sortowania dzięki zamianie elementów położonych daleko od siebie.
- Po wyborze elementu wzorcowego **w** przeglądamy tablicę jednocześnie od strony lewej do prawej szukając elementu $x_i \geq w$ i od strony prawej do lewej szukając elementu $x_i \leq w$.
- Zamieniamy znalezione kolejne pary elementów, aż przeszukamy całą tablicę i podzielimy ciąg na 2 podciągi: elementów $x_i \leq w$ i elementów $x_i \geq w$.
- Następnie dla każdego z podciągów stosujemy tę samą metodę podziału, aż otrzymamy ciągi jednoelementowe.

Przykład 1

i → ← j

x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]	x[8]	uwagi
6	2	7	**3**	8	1	9	5	ciąg oryginalny
1					6			zamiana x[1] i x[6]
		3	7					zamiana x[3] i x[4]
1	2	**3**	7	8	6	9	5	podział ciągu i=4, j=3
1	**2**	**3**						pierwsze pod-zadanie, i=1, j=3
1	**2**	**3**						podzadania jednoelementowe

x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]	x[8]	uwagi
			7	8	**6**	9	5	drugie podzadanie
			5 ←				7 →	zamiana x[4] i x[8]
				6 ←	8 →			zamiana x[5] i x[6]
			5	**6**	8	9	7	podział ciągu i=6, j=5
			5 ↔	**6**				podzadanie 2-elementowe
			5					podzadanie jednoelementowe

x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]	x[8]	uwagi
					8	**9**	7	podzadanie 3-elementowe
						7 ↔	**9**	zamiana x[7] i x[8]
					8	7	**9**	podział ciągu, i=8, j=7
					8	7		podzadanie 2-elementowe
					7 ↔	**8**		zamiana x[6] i x[7]
					7	**8**		podzadania jednoelementowe
1	**2**	**3**	**5**	**6**	**7**	**8**	**9**	**ciąg uporządkowany**

Program 2

```
// plik nagłowkowy tenprog.h
/* program do sortowania elementów ciągu */

void sort_b (int, int[]); //prototypy funkcji
void sort_s (int, int, int[ ]);
void wybor (int, int[]);
void druk (int, int[]);
```



```
// plik źródłowy sortowanie.c
# include <stdio.h>
# include <stdlib.h>
# include "tenprog.h"

void sort_b (int n, int a[ ]) //sortowanie babelkowe
{
   ...
}

void sort_s (int l, int p, int a[ ]) //sortowanie szybkie
{
   int i,j,aa,w;
   i=l; j=p;
   w=a[(l+p)/2];
```

```
    do {
        while (a[i]<w)  i++;
        while (w<a[j])  j--;
        if (i<=j)
                {
                  aa=a[i];          //zamiana a[i] z a[j]
                  a[i]=a[j]; a[j]= aa;
                  i++; j--;
                }
    } while (i<=j);
    if (l < j) sort_s (l, j, a);
    if (i < p) sort_s (i, p, a);
}
```

195


```
void wybor (int n, int a[])
/* wybor metody sortowania tablic */
{
    int p;
    char c;

    do {
        fflush(stdin);  //stdio.h - czyszczenie bufora
        p=0;
        printf("Wybierz metode sortowania:
                              b - babelkowe, s - szybkie\n");
        c = getchar();
```

196

```
        switch (c)
        {
                case 'b': sort_b (n,a);
                          break;
                case 's': sort_s (0,n-1,a);
                          break;
                default: printf ("Blad!   "); p=1;
                          break;
        }
    } while (p);
}
```

197


```
// plik źródłowy program.c

# include <stdio.h>
# include <stdlib.h>
# include "tenprog.h"

int main (void)
{
   ...
   /* sort_b (n,a); */  wybor (n,a);
   ...
}
```

198

Program 3

```
// plik nagłowkowy tenprog.h
/* program do porównywania 2 metod
   sortowania - ich zlozonosci obliczeniowej*/

int sort_b (int, int[]); //prototypy funkcji
int sort_s (int, int, int[ ]);
int wybor (int, int[]);
void druk (int, int[]);
void wpisz (int, int[]);
```

199


```
// plik źródłowy sortowanie.c
# include <stdio.h>
# include <stdlib.h>
# include "tenprog.h"
int sort_b (int n, int a[ ]) //sortowanie babelkowe
{
   int l=0;  // do zliczania liczby zamian
   ...
        p=1; //zamiana
        l++;
   ...
   return l;
}
```

200

```
int sort_s (int l, int p, int a[ ]) //sortowanie szybkie
{
   static int lz=0; //do zliczania liczby zamian
   int i,j,aa;
   i=l; j=p;
   register int w;
   w=a[(l+p)/2];
```



```
   do {
        while (a[i]<w)  i++;
        while (w<a[j])  j--;
        if (i<=j)
                {
                   aa=a[i];            //zamiana a[i] z a[j]
                   a[i]=a[j]; a[j]= aa;
                   i++; j--; lz++;
                }
   } while (i<=j);
   if (l < j) sort_s (l, j, a);
   if (i < p) sort_s (i, p, a);
   return lz;
}
```

```
int wybor (int n, int a[])
/* wybor metody sortowania tablic */
{
   int p, l;
   char c;

   do {
        fflush(stdin);  //stdio.h - czyszczenie bufora
        p=0;
        printf("Wybierz metode sortowania:
                              b - babelkowe, s - szybkie\n");
        c = getchar(); c = getchar();
```



```
        switch (c)
        {
                case 'b': l = sort_b (n,a);
                          break;
                case 's': l = sort_s (0,n-1,a);
                          break;
                default: printf ("Blad!   "); p=1;
                          break;
        }
   } while (p);
   return l; // liczba zamian
}
```

```
// plik źródłowy program.c
# include <stdio.h>
# include <stdlib.h>
# include "tenprog.h"

int main (void)
{
   int n, lw;
   int a[200];
   char c;
   ...
   wpisz (n,a); // kolejne elementy ciągu
```

205


```
   ...
   druk(n,a);
   ...
   lw = wybor (n,a);   /* sort_b (n,a);  wybor (n,a);*/
   ...
   druk(n,a);
   ...
   printf("Liczba zamian = %d\n",lw);
   printf("\n");
   c = getchar();
return 0;
}
```

206

```
void wpisz (int n, int a[])
{
   char c;
   int j;
   unsigned int seed;
   //fflush(stdin); // czyszczenie bufora klawiatury
   fpurge(stdin); //stdio.h - czyszczenie bufora
                  //klawiatury w systemie UNIX
   printf("Wybierz sposob wprowadzania ciagu liczb\n");
   printf("\tk - z klawiatury,\n\tg - z generatora liczb
                                        losowych\n\n");
   c=getchar();
```



```
while (c!='k'&& c!='g') {
        fflush(stdin); //stdio.h
        printf("Blad!  Wpisz jeszcze raz\n");
        c=getchar();
   }
   if (c=='k') {  //fragment starego programu:
        printf("Wpisz kolejne elementy ciagu \n\n");
        for (j=0; j<n; j++) {
              printf("a[%d] = ", j+1);
              scanf("%d", &a[j]);
        } //koniec fragmentu
   }
```

```
    else {
        printf("\nWpisz ziarno ");
        scanf("%d", &seed);
        printf("\n\n");
        srand(seed); //stdlib.h
        for (j=0; j<n; j++) a[j]=rand(); //stdlib.h
    }
}

void druk (int k, int b[])
{ ... }
```


Przykład

Napisz program do znajdowania największego wspólnego dzielnika dwóch liczb naturalnych wykorzystujący rekurencyjny algorytm Euklidesa.

```
/* program do wyznaczania najwiekszego wspolnego
   podzielnika dwoch liczb naturalnych */
# include <stdio.h>
int NWD (int, int);

int main (void)
{
   int w,x,y;
   printf ("Program do wyznaczania najwiekszego dzielnika
                                   dwoch liczb naturalnych\n\n");
   printf ("Wpisz dwie liczby naturalne ");
   scanf ("%d %d", &x, &y);
   w = NWD (x,y);
   printf ("Najwiekszyn dzielnikiem liczb %d i %d jest %d\n",
                                                   x,y,w);
   return 0;
}
```

211


```
int NWD (int x, int y) //najwiekszy wspolny dzielnik
{
   int r, z;
   r = x % y;          //  x / y = c*y + r
   // {F: x>0 and y>0}
   if (r == 0)  z = y; else z = NWD (y,r);
   // {G: z = NWD(x,y)}
   return z;
}
```

212