Historia języka C

① 1972: Dennis Ritchie i system UNIX;

① 1978: K & R (Kernighan & Ritchie) C;

① 1989: standard ANSI (C90).

Struktura programu

```
dyrektywy preprocesora
int main(void)
{
   opis nazw zmiennych i ich typu;
   ciąg instrukcji;
   wyprowadzenie rezultatu
   return 0;
}
```

Analogie do przepisu kulinarnego:

✓ nazwa potrawy $\longrightarrow$ nazwa programu,

✓ składniki $\longrightarrow$ deklaracje zmiennych,

✓ przepis $\longrightarrow$ ciąg instrukcji.

Książka: S. Prata (1999): *Język C. Szkoła programowania*. – Wrocław: Robomatic.

Środowisko programistyczne: Dev-C++,
`www.bloodshed.net/dev/devcpp.html`
przejść do *Downloads* i wybrać
Dev-C++ 5.0 beta 9.2 (4.9.9.2) with Mingw/GCC 3.4.2

```
#include <stdio.h>
int main(void)
{
    int a, b, c;
    scanf("%d %d", &a, &b);
    c = a + b;
    printf("%d %d %d\n", a, b, c);
    getch(); /* Nie zamykaj okna! */
    return 0;
}
```

Efekt wykonania programu:

```
13   11   ↵
13   11   24
_
```

Możliwe wariacje:

- Poprzez `scanf` uzyskuje się taki sam efekt jak `a = 13, b = 11`, ale nie trzeba zmieniać programu.
- Można napisać

```
scanf("%d", &a);
scanf("%d", &b);
```

  ale `scanf("%d %d", &a, &b);` jest bardziej zwarte.

- `printf("%d %d %d\n", a, b, c);` jest równoważne

```
printf("%d ", a);
printf("%d ", b);
printf("%d\n", c);
```

Komunikacja z użytkownikiem

```
#include <stdio.h>
int main(void)
{
  int a, b, c;
  printf("Podaj dwie liczby: ");
  scanf("%d %d", &a, &b);
  c = a + b;
  printf("1-szy skladnik = %d\n", a);
  printf("2-gi skladnik = %d\n", b);
  printf("Suma skladnikow = %d\n",
     c);
  return 0;
}
```

Efekt wykonania programu:

```
Podaj dwie liczby: 13   11   ↵
1-szy składnik = 13
2-gi składnik = 11
Suma składników = 24
```

Komentarze

Wszystko, co stoi między symbolami `/*` i `*/` jest ignorowane przy tłumaczeniu programu źródłowego na kod maszynowy.

```
#include <stdio.h>
int main(void)
{
  int a;
  /* Maly program z komentarzami */
  a = 1;
  printf("To na ekranie: a = %d\n",
     a);
  /* Ten komentarz jest zbyteczny */
  return 0;
}
```

Efekt wykonania programu:

```
To na ekranie: a = 1
```

Styl („charakter pisma”)

```
#include <stdio.h>
int
   main(void){int
    a, b, c; scanf("%d %d", &a, &b);
c =
         a +
      b
  ; printf("%d %d %d\n",
    a, b, c); return 0;}
```

Instrukcja przypisania

```
#include <stdio.h>
int main(void)
{
   int a;
   a = 4;     printf("%d\n", a);
   a = a + 1; printf("%d\n", a);
   a = 8;     printf("%d\n", a);
   return 0;
}
```

Operacje na liczbach całkowitych

```
#include <stdio.h>
int main(void)
{
   int a, b, c;

   a = 17;
   b = 3;
   c = a * b;
   printf("17 * 3 = %d\n", c);
   c = a / b;
   printf("17 / 3 = %d\n", c);
   c = a % b;
   printf("17 %% 3 = %d\n", c);
   c = a + b;
   printf("17 + 3 = %d\n", c);
   c = a - b;
   printf("17 - 3 = %d\n", c);
   return 0;
}
```

Efekt wykonania programu:

```
17 * 3 = 51
17 / 3 = 5
17 % 3 = 2
17 + 3 = 20
17 - 3 = 14
```

Co to jest *float*?

```
#include <stdio.h>
int main(void)
{
   float a, b, c;
   a = 3.5;
   b = 7.6;
   c = a + b;
   printf("Suma = %f\n", c);
   return 0;
}
```

Efekt wykonania programu:

```
Suma = 11.100000
```

Zapis liczb

Reguła: W liczbie rzeczywistej pojawia się kropka dziesiętna, a nie przecinek!

.7	0.	⟵ poprawne!
0.7	0.0	⟵ poprawne!

Postać z wykładnikiem:

$$\text{liczba} = \text{mantysa} \times 10^{\text{potęga}}$$

Przykłady

$$\texttt{2.7e3} = 2,7 \times 10^{3} = 2,7 \times 1000 = 2700$$

$$\texttt{-1.51e-5} = -1,51 \times 10^{-5} = -0,0000151$$

$$\texttt{2.753e12} = 2753000000000$$

$$\texttt{2.753e-12} = 0.000000000002753$$

Specyfikatory konwersji

```
a = 0.000023;
printf("%f\n", a);   ⟶   0.000023
printf("%e\n", a);   ⟶   2.300000e-005
printf("%g\n", a);   ⟶   2.3e-005

a = 0.45;
printf("%f\n", a);   ⟶   0.450000
printf("%e\n", a);   ⟶   4.500000e-001
printf("%g\n", a);   ⟶   0.45

a = 1234567.0;
printf("%f\n", a);   ⟶   1234567.000000
printf("%e\n", a);   ⟶   1.234567e+006
printf("%g\n", a);   ⟶   1.23457e+006
```

int i *float* razem

```
#include <stdio.h>
#include <math.h>
int main(void)
{
   int n, k;
   float a, b;
   a = -3.6; n = 4;
   b = n; printf("%g ", b);
   n = a;
   printf("%d ", n);
   k = floor(a);
   printf("%d\n", k);
   return 0;
}
```

```
4 -3 -4
```

Czy dozwolone są poniższe instrukcje przypisania?

```
b := n + 4.6;
b := 3 * 7.2 + n;
n := 2.5 * 4;
```

Wykorzystanie nawiasów i priorytety

Jak interpretować poniższą instrukcję przypisania?

```
n = 4 - 3 * 8;
```

Mamy dwie możliwości:

① $4 - (3 \times 8)$

② $(4 - 3) \times 8$

Mamy następujące reguły priorytetów :

- Mnożenie i dzielenie wykonuje się wcześniej niż dodawanie i odejmowanie (lub: mnożenie i dzielenie mają wyższy priorytet niż dodawanie i odejmowanie).
- Operacje o identycznym priorytecie wykonują się od lewej na prawo. Mnożenie i dzielenie mają jednakowy priorytet (podobnie dodawanie i odejmowanie).
- Operacje na zmiennych zawartych w nawiasach mają pierwszeństwo.

Przykład wykorzystania nawiasów:

```
#include <stdio.h>
int main(void)
{
   float far, cel;
   printf("Podaj temperature "
          "Fahrenheita\n");
   scanf("%f", &far);
   cel = ((far - 32.0) / 9.0) * 5.0;
   printf("Temperatura "
          "Celsjusza: %g", cel);
   return 0;
}
```

Funkcje matematyczne na typie *float*

```
#include <stdio.h>
#include <math.h>
int main(void)
{
   float a = 2.0, b;
   b = pow(a, 2.0);
   printf("pow(2.0, 2.0) = %g\n", b);
```

```
    b = sqrt(a);
    printf("sqrt(2.0) = %g\n", b);
    b = sin(a);
    printf("sin(2.0) = %g\n", b);
    b = cos(a);
    printf("cos(2.0) = %g\n", b);
    b = atan(a);
    printf("atan(2.0) = %g\n", b);
    b = log(a);
    printf("log(2.0) = %g\n", b);
    b = exp(a);
    printf("exp(2.0) = %g\n", b);
    return 0;
}
```

```
pow(2.0, 2.0) = 4
sqrt(2.0) = 1.41421
sin(2.0) = 0.909297
cos(2.0) = -0.416147
atan(2.0) = 1.10715
log(2.0) = 0.693147
exp(2.0) = 7.38906
```

Funkcje te można składać:

```
a = -4.0;
b = sqrt(25.6 + abs(a * 6.5));
```

Stałe

```
#include <stdio.h>
int main(void)
{
   float obwod, r, pi;
   pi = 3.14159;
   scanf("%f", &r);
   obwod = 2 * pi * r;
   printf("obwod = %g\n", obwod);
   return 0;
}
```

Jak zabezpieczyć się przed przypadkową zmianą wartości `pi`?

Lepiej napisać

```
#include <stdio.h>
#define PI 3.14159
int main(void)
{
   float obwod, r;
   scanf("%f", &r);
   obwod = 2 * PI * r;
   printf("obwod = %g\n", obwod);
   return 0;
}
```

Zadanie. Obliczyć czas lotu t_u i drogę $x(t_u)$ ciała rzuconego z prędkością v pod kątem α.

Fizyka daje zależności

$$x(t) = vt\cos(\alpha),$$

$$y(t) = vt\sin(\alpha) - \frac{1}{2}gt^2.$$

Z równania $y(t_u) = 0$ mamy $t_u = \dfrac{2v\sin(\alpha)}{g}$,
i dalej $x(t_u) = vt_u\cos(\alpha)$.

```
#include <stdio.h>
#include <math.h>
#define G 9.81
#define PI 3.1415926
int main(void)
{
    float v, alfa, t_u, x_u;

    printf("Podaj predkosc "
       "poczatkowa [m/s]:\n");
    scanf("%f", &v);
    printf("Podaj kat [stopnie]:\n");
    scanf("%f", &alfa);
    alfa = alfa * PI / 180.0;
    t_u = 2.0 * v * sin(alfa) / G;
    x_u = v * t_u * cos(alfa);
    printf("Czas lotu: %.3f s\n",
                              t_u);
    printf("Przebyta odleglosc:"
           " %.3f m\n", x_u);
    return 0;
}
```

Typ *double* i dokładność obliczeń

```
#include <stdio.h>
int main(void)
{
    float x;
    double y;
    x = 1.0f - 0.2f - 0.2f - 0.2f
      - 0.2f - 0.2f;
    y = 1.0  - 0.2  - 0.2  - 0.2
      - 0.2  - 0.2;
    printf("x = %e\n y = %e\n",
                              x, y);
    return 0;
}
```

```
x = 2.980232e-008
y = 5.151115e-017
```

Rezultat wcale nie jest zerem!

```
#include <stdio.h>
#include <math.h>
#define G 9.81
#define PI 3.1415926
int main(void){
    double v, alfa, t_u, x_u;
    printf("Podaj predkosc "
           "poczatkowa [m/s]:\n");
    scanf("%lf", &v);
    printf("Podaj kat [stopnie]:\n");
    scanf("%lf", &alfa);
    alfa = alfa * PI / 180.0;
    t_u = 2.0 * v * sin(alfa) / G;
    x_u = v * t_u * cos(alfa);
    printf("Czas lotu: %.3f s\n",
                                t_u);
    printf("Przebyta odleglosc: "
           "%.3f m\n", x_u);
    return 0;
}
```

Zwrócić uwagę na zmieniony łańcuch sterujący w funkcji `scanf` i ten sam w funkcji `printf`!

Typy zmiennych i ich rozmiary

```
#include <stdio.h>
int main(void){
    printf("Rozmiary w bajtach:\n");
    printf("char:       %d\n",
                  sizeof(char));
    printf("int:        %d\n",
                  sizeof(int));
    printf("unsigned:   %d\n",
                  sizeof(unsigned));
    printf("short:      %d\n",
                  sizeof(short));
    printf("long:       %d\n",
                  sizeof(long));
    printf("long long:  %d\n",
                  sizeof(long long));
    printf("float:      %d\n",
                  sizeof(float));
    printf("double:     %d\n",
                  sizeof(double));
    return 0;   }
```

```
Rozmiary w bajtach:
char: 1
int: 4
unsigned: 4
short: 2
long: 4
long long: 8
float: 4
double: 8
```

Sekwencje sterujące:

typ	scanf	printf
char	%c	%c
int	%d	%d
unsigned	%u	%u
short	%hd	%hd
long	%ld	%ld
long long	%lld	%lld
float	%f	%f
double	%lf	%f

Formatowane wyjście

```
#include <stdio.h>
int main(void)
{
   float a1, a2, b1, b2;
   a1 = 1.111; a2 = a1 * a1;
   b1 = 1.222; b2 = b1 * b1;
   printf("%10s %10s\n",
               "Wielkosc", "Kwadrat");
   printf("%10.2f %10.2f\n", a1, a2);
   printf("%10.2f %10.2f\n", b1, b2);
   return 0;
}
```

```
  Wielkosc    Kwadrat
      1.11       1.23
      1.22       1.49
_
```

Type znakowy

`char` – uporządkowany zbiór wszystkich znaków graficznych oraz sterujących reprezentowanych na danym komputerze (zależy od typu komputera).

Znaki ASCII:

- Litery: `'A'`, `'B'`, `'C'`, `'D'`, ..., `'Z'`, `'a'`, `'b'`, `'c'`, `'d'`, ..., `'z'`;
- Cyfry: `'0'`, `'1'`, `'2'`, `'3'`, ..., `'9'`;
- Znaki specjalne: `'!'`, `'@'`, `'#'`, `'%'`, ...;
- Spacja: `' '`;
- Znaki sterujące, np. `'\r'`, czyli CR (*ang.* carriage return) – powrót kursora na początek wiersza, `'\n'` czyli NL (*ang.* new line) – przejście kursora do następnego wiersza.

Pytanie: Jak uzyskać apostrof? ➜ `'\''`

Znaki są wewnętrznie reprezentowane przez ciąg zer i jedynek. Jeżeli te ciągi zinterpretuje się jako liczby całkowite, zdefiniuje się w ten sposób *uporządkowanie* zbioru znaków, np.:

'0'$\mapsto$48 '1'$\mapsto$49 '2'$\mapsto$50

'A'$\mapsto$65 'B'$\mapsto$66 'C'$\mapsto$67

'a'$\mapsto$97 'b'$\mapsto$98 'c'$\mapsto$99

```
#include <stdio.h>
int main(void)
{
    char ch;

    printf("Wpisz jakis znak\n");
    scanf("%c", &ch);
    printf("Kod znaku %c to %d.\n",
                              ch, ch);
    return 0;
}
```

Jak uzyskać następny znak? Bardzo prosto:

```
ch = ch + 1;
```

Łańcuchy znakowe

```
char imie[7];
imie = "Ola";
```

'O'	'l'	'a'	'\0'			

```
#include <stdio.h>
#include <string.h>
int main(void)
{
   char imie[20], nazwisko[20];
   printf("Podaj swoje imie "
          "i nazwisko:\n");
   scanf("%s %s", imie, nazwisko);
   printf("%s %s\n", nazwisko, imie);
   printf("Twoje imie ma %d liter "
          "i zajmuje %d bajtow.\n",
      strlen(imie), sizeof imie);
   return 0;
}
```

Pętla *while*

```
#include <stdio.h>
#define KOREKTA -1
#define MNOZNIK 0.666666
int main(void){
    double but, stopa;
    printf("Rozmiar buta    "
           "Dlugosc stopy\n");
    but = 24.0;
    while (but < 45){
       stopa = MNOZNIK * but
               + KOREKTA;
       printf("%8.1f %15.2f cm\n",
           but, stopa);
       but = but + 1.0;
    }
    return 0;
}
```

```
Rozmiar buta  Dlugosc stopy
    24.0         15.00 cm
    25.0         15.67 cm
    26.0         16.33 cm
     :            :
    43.0         27.67 cm
    44.0         28.33 cm
```

Operator inkrementacji

```
#include <stdio.h>
int main(void){
    int a = 1, b = 1;
    int aplus, plusb;
    aplus = a++;
    plusb = ++b;
    printf("a   aplus   "
           "b   plusb \n");
    printf("%1d %5d %5d %5d\n",
       a, aplus, b, plusb);
    return 0;
}
```

```
a   aplus   b   plusb
2     1     2     2
```

```
#include <stdio.h>
int main(void){
    int a, q;
    a = 2; q = 2 * ++a;
    printf("a = %d, q = %d\n",
                        a, q);
    a = 2; q = 2 * a++;
    printf("a = %d, q = %d\n",
                        a, q);
    return 0;
}
```

```
a = 3, q = 6
a = 3, q = 4
```

Pytanie: Jak wykorzystać operator inkrementacji do uproszczenia programu o numeracji butów?

Wyrażenia logiczne

```
#include <stdio.h>
int main(void)
{
   int x = 4, b;

   b = x > 3;
   printf("%d\n", b);
   b = x < 3;
   printf("%d\n", b);
   return 0;
}
```

```
1
0
```

W wielu programach występuje sytuacja, kiedy na podstawie określonego warunku będzie lub nie będzie wykonany szereg instrukcji. Taki warunek możemy rozumieć jako stwierdzenie *prawdziwe* lub *fałszywe* (warunek spełniony lub niespełniony).

symbol	relacja	przykład
<	$<$	`x < 3`
<=	$\leq$	`x <= 3`
>	$>$	`x > 3`
>=	$\geq$	`x >= 3`
==	$=$	`x == 3`
!=	$\neq$	`x != 3`

Warunki mogą zawierać wyrażenia arytmetyczne:

```
x + 6.5 < y * 5
```

Można też rozważać bardziej złożone warunki, np.

```
x < 7  i  x > 3
```

```
x > 100  lub  x < 10
```

dla których odpowiednio mamy zapis

```
x < 7 && x > 3
```

```
x > 100 || x < 10
```

a	b	a && b	a || b
1	1	1	1
1	0	0	1
0	1	0	1
0	0	0	0

Oprócz tego istnieje operator `!`, który neguje wartość wyrażenia:

```
!(a < b)   ≡   a >= b
```

Pytanie: Jak zinterpretować poniższy warunek?

```
3 < x && x < 10 || 13 < x && x<20
```

Instrukcja warunkowa *if*

Instrukcje warunkowe służą do wpływania na kolejność wykonania instrukcji programu.

```
if (x < 3)
   printf("%d\n", x);
```

```
printf("Podaj swoj wiek\n");
scanf("%d", &wiek);
if (wiek >= 18)
   printf("Jestes pelnoletni\n");
```

```
if (x < 3)
   printf("x < 3\n");
else
   printf("x >= 3\n");
```

```
printf("Podaj swoj wiek\n");
scanf("%d", &wiek);
if (wiek >= 18)
   printf("Jestes pelnoletni\n");
else
   printf("Jestes maloletni\n");
```

Uwaga! Zwrócić uwagę na średnik przed `else`!

Pytanie: Jaki będzie rezultat poniższych instrukcji?

```
#include <stdio.h>
#include <math.h>
#define NUM 500.0
#define EPS 1.0e-10
int main(void)
{
    double x = 1.0;

    printf("%d\n",
       x / NUM * NUM == x);
    printf("%d\n",
       fabs(x / NUM * NUM - x)
                          < EPS);

    return 0;
}
```

```
0
1
```

Pytanie: Jaki zamieniać małe litery na duże?

```
#include <stdio.h>
int main(void)
{
   char ch;

   printf("znak\n");
   scanf("%c", &ch);
   if ('a' <= ch && ch <= 'z')
      ch = ch - 'a' + 'A';
   printf("%c\n", ch);

   return 0;
}
```

Pętla *while*

Zadanie. Obliczyć sumę ciągu liczb, w którym pierwszy składnik jest zadany, a każdy następny jest o jeden mniejszy od poprzedniego. Ciąg ma się kończyć na ostatniej liczbie większej od 5.5.

```
#include <stdio.h>
#define KONIEC 5.5f
int main(void)
{
   float x, sum;

   x = 10.0f;
   sum = 0.0f;
   while (x > KONIEC)
   {
      sum = sum + x;
      x = x - 1;
   }
   printf("sum = %6.2f\n", sum);
   printf("  x = %6.2f\n", x);
   return 0;
}
```

```
sum =  40.00
  x =   5.00
_
```

Można jeszcze zwięźlej:

```
x = 10.0f;
sum = 0.0f;
while (x > KONIEC)
{
    sum += x;
    x--;
}
```

- ✍ Wiele instrukcji wewnątrz pętli ➽ zastosować instrukcję złożoną (blok) `{ ... }`.
- ✍ Wartość wyrażenia logicznego musi być modyfikowana wewnątrz pętli.
- ✍ Liczba wykonań nie jest z góry wiadoma.
- ✍ Kiedy instrukcje wewnątrz pętli nie wykonanją się?

Pętla *do ... while*

```
#include <stdio.h>
#define KONIEC 5.5f
int main(void)
{
   float x, sum;

   x = 10.0f;
   sum = 0.0f;
   do
   {
      sum = sum + x;
      x = x - 1;
   } while (x > KONIEC);
   printf("sum = %6.2f\n", sum);
   printf("  x = %6.2f\n",   x);
   return 0;
}
```

Wersja skrócona:

```
x = 10.0f;
sum = 0.0f;
do
    sum += x;
while (--x > KONIEC);
```

Pętla *for*

```
for (k = 3; k <= 23; k++)
    printf("%d\n", k * k);

for (k = 99; k >= 78; k--)
    printf("%d\n", k * k);
```

Pytanie: Co wypisze następujący fragment?

```
int sum, x;
sum = 0;
for (x = 5; x >= 1; sum += x--)
   ;
printf("sum = %d", sum);
```

Zadanie. Wyznaczyć wartość sumy

$$\frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \frac{1}{5^2}$$

```
#include <stdio.h>
int main(void)
{
   float sum, a;
   int k;

   sum = 0.0f;
   for (k = 1; k <= 5; k++)
   {
      a = 1.0f / (float) k;
      sum += a * a;
   }
   printf("sum = %8.5f\n", sum);
   return 0;
}
```

Jeszcze krótsza wersja:

```
   sum = 0.0f;
   for (k = 1; k <= 5; sum += a * a)
      a = 1.0f / (float) k++;
```

Ćwiczenia nt. pętli

Zadanie. Napisać program znajdujący cyfry reprezentacji binarnej liczby naturalnej n.

```
#include <stdio.h>
int main(void){
   int n, i;
   printf("Podaj liczbe "
          "naturalna:\n");
   scanf("%d", &n);
   printf("Reprezentacja "
          "binarna %d to ", n);
   i = n;
   do {
      printf("%d", i % 2);
      i /= 2;
   } while (i != 0);
   return 0; }
```

Pytanie: Jak odwrócić kolejność cyfr w wyniku?

Zadanie. Jeżeli y jest przybliżeniem $\sqrt[3]{x}$, to $(2y^3 + x)/3y^2$ jest jeszcze dokładniejszym przybliżeniem tego pierwiastka. Wykorzystać ten schemat do wyznaczenia $\sqrt[3]{x}$.

```
#include <stdio.h>
#include <math.h>
#define ZAWSZE 1
#define EPS 1.0e-7
int main(void) {
   float x, y, nowe_y;
   printf("Podaj liczbe:\n");
   scanf("%f", &x);
   nowe_y = 1.0f;
   do {
      y = nowe_y;
      nowe_y = (2.0f * y * y * y
                + x)
             / (3.0f * y * y);
   } while (fabs(y - nowe_y) >= EPS);
   printf("%g\n", nowe_y);
   return 0;
}
```

Zadanie. Rozłożyć liczbę n na czynniki pierwsze.

```
#include <stdio.h>
#include <math.h>
int main(void){
   int i, k, n;
   printf("Podaj liczbe:\n");
   scanf("%d", &n);
   printf("Czynniki pierwsze:\n");
   k = n;
   while (k % 2 == 0) {
      printf("%d\n", 2);
      k /= 2;
   }
   i = 3;
   while (i <= sqrt(k) + 1.0){
      if (k % i == 0){
         printf("%d\n", i);
         k /= i;
      } else
         i += 2; }
   if (k > 1) printf("%d\n", k);
   return 0; }
```

Interaktywne wejście

```
#include <stdio.h>
int main(void)
{
    long num, stan;
    long suma = 0L;
    printf("Podaj liczbe do "
           "zsumowania lub q aby"
           " zakonczyc.\n");
    while (scanf("%ld", &num) == 1)
    {
       suma += num;
       printf("Podaj liczbe do "
              "zsumowania lub q aby"
              " zakonczyc.\n");
    }
    printf("Suma liczb wynosi "
           "%ld.\n", suma);
    return 0;
}
```

Operator przecinkowy

Zadanie. Problem Zenona z Elei, czyli przybliżanie sumy $1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots$.

```
#include <stdio.h>
int main(void){
   int licznik, granica;
   double czas, x;
   printf("Ile wyrazow?\n");
   scanf("%d", &granica);
   for (czas = 0.0, x = 1.0,
                licznik = 1;
        licznik <= granica;
        licznik++, x *= 2.0)
   {
      czas += 1.0 / x;
      printf("czas = %f gdy licznik"
             " = %d.\n", czas,
             licznik);
   }
   return 0; }
```

Pętle zagnieżdżone

```
#include <stdio.h>
#define WIERSZE 6
#define ZNAKI 6
int main(void)
{
   int w;
   char ch;
   for (w = 0; w < WIERSZE; w++) {
      for (ch = 'A' + w;
                ch < 'A' + ZNAKI; ch++)
         printf("%c", ch);
      printf("\n");
   }
   return 0;
}
```

```
ABCDEF
BCDEF
CDEF
DEF
EF
F
```

Typ tablicowy

Zadanie. Pewna 100-osobowa grupa studencka zdawała egzamin z informatyki. Napisać program pytający użytkownika o numery ewidencyjne i liczby zdobytych punktów kolejnych studentów, a następnie komunikujący którzy studenci uzyskali wynik powyżej średniej.

Nasuwające się rozwiązania:

❶ Wczytać dane, obliczyć średnią, po czym ponownie wczytać dane.

❷
```
int ocena1, ocena2, ..., ocena100;
int numSt1, numSt2, ..., numSt100;
```

Co jednak zrobić, gdy taką statystykę należy wykonać dla setek lub nawet tysięcy wartości?

Trudność: Jedna zmienna poznanych typów może przechowywać tylko jedną wartość.

Remedium: Zastosować typ tablicowy; zmienna tego typu może przechowywać cały zbiór wartości.

Tablica = uporządkowany zbiór elementów *tego samego typu*, do których dostęp uzyskuje się poprzez podanie ich pozycji.

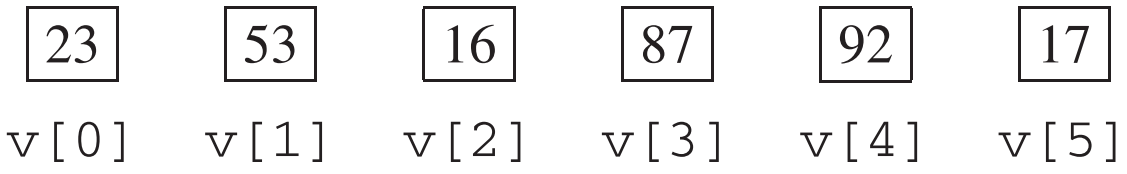
Wymiary tablicy – liczba wartości koniecznych do określenia położenia elementu w tablicy

Tablica jednowymiarowa (analogia: wektor):

`int v[6];` — deklaracja

`v` – nazwa całej tablicy

`v[0], ..., v[5]` – poszczególne elementy



```
#include <stdio.h>
#define L_STUDENTOW 100
int main(void)
{
  int nrStud[L_STUDENTOW];
  int ocena[L_STUDENTOW];
  int sumaOcen = 0, i;
  float srednia = 0.0f;
  for (i = 0; i < L_STUDENTOW; i++){
     printf("Podaj nr studenta"
            " i jego ocene\n");
     scanf("%d %d", &nrStud[i],
                    &ocena[i]);
     sumaOcen += ocena[i];
  }
  srednia = (float) sumaOcen
          / (float) L_STUDENTOW;
  printf("Numery studentow "
   "z ocenami powyzej sredniej:\n");
  for (i = 0; i < L_STUDENTOW; i++)
     if (ocena[i] > srednia)
        printf("%d\n", nrStud[i]);
   return 0;              }
```

Funkcje *getchar* i *putchar*

```
#include <stdio.h>
#define ODSTEP ' '
int main(void)
{
   char ch;
   ch = getchar();
   while (ch != '\n')
   {
      if (ch == ODSTEP)
         putchar(ch);
      else
         putchar(ch + 1);
      ch = getchar();
   }
   putchar(ch);
   return 0;
}
```

```
mow mi Hal ↵
npx nj Ibm
_
```

Wersja skrócona:

```
#include <stdio.h>
#include <ctype.h>
int main(void)
{
   char ch;
   while ((ch = getchar()) != '\n')
      if (isspace(ch))
         putchar(ch);
      else
         putchar(ch + 1);
   putchar(ch);
   return 0;
}
```

Rodzina *ctype.h*

- `tolower()` — zamienia wielkie litery na małe;
- `toupper()` — zamienia małe litery na duże.

Nazwa	Prawdziwa jeśli argument jest
`isalnum()`	literą lub cyfrą
`isalpha()`	literą
`iscntrl()`	znakiem sterującym
`isdigit()`	cyfrą
`isgraph()`	znakiem drukowanym
`islower()`	małą literą
`isupper()`	dużą literą

Operator warunkowy

```
x = y < 0 ? -y : y;
```

oznacza to samo, co

```
if (y < 0)
   x = -y;
else
   x = y;
```

```
#include <stdio.h>
#define POKRYCIE 18
int main(void)
{
  int m_kw, puszki;
  printf("Podaj liczbe m kw. do "
         "pomalowania:\n");
  while (scanf("%d", &m_kw) == 1)
  {
    puszki = m_kw / POKRYCIE;
    puszki +=
       m_kw % POKRYCIE == 0 ? 0 : 1;
    printf("Potrzeba %d %s "
           "farby.\n", puszki,
           puszki == 1 ? "puszki"
                       : "puszek");
    printf("Podaj kolejno wartosc "
           "(q konczy program):\n");
  }
  return 0;
}
```

Instrukcja *continue*

```
#include <stdio.h>
#define MIN 0.0f
#define MAX 100.0f
int main(void)
{
  float wynik;
  float suma = 0.0f;
  int n = 0;
  float min = MAX;
  float max = MIN;
  printf("Podaj wyniki:\n");
  while (scanf("%f", &wynik) == 1)
  {
    if (wynik < MIN || wynik > MAX)
    {
      printf("%0.1f jest "
             "nieprawidlowa "
             "wartoscia.\n", wynik);
      continue;
    }
```

```
    printf("Przyjeto %0.1f:\n",
                wynik);
    min = wynik < min ? wynik : min;
    max = wynik > max ? wynik : max;
    suma += wynik;
    n++;
  }
  if (n > 0)
  {
    printf("Srednia z %d wynikow "
           "wynosi %0.1f.\n", n,
           suma / n);
    printf("Najnizszy = %0.1f, "
           "najwyzszy = %0.1f\n",
           min, max);
  }
  else
    printf("Nie podano zadnych "
         "prawidlowych wynikow.\n");
  return 0;
}
```

Instrukcja *break*

```
#include <stdio.h>
int main(void) {
  float dlug, szer;
  printf("Podaj dlugosc "
         "prostokata:\n");
  while (scanf("%f", &dlug) == 1) {
    printf("Dlugosc = %0.2f:\n",
                          dlug);
    printf("Podaj szerokosc "
           "prostokata:\n");
    if (scanf("%f", &szer) != 1)
       break;
    printf("Szerokosc = %0.2f:\n",
                            szer);
    printf("Pole = %0.2f:\n",
                 dlug * szer);
    printf("Podaj dlugosc "
           "prostokata:\n");
  }
  return 0; }
```

Instrukcja *switch*

```
#include <stdio.h>
#include <ctype.h>
int main(void)
{
   char ch;
   int a_licz, e_licz, i_licz,
       o_licz, u_licz, y_licz;

   a_licz = e_licz = i_licz = o_licz
          = u_licz = y_licz = 0;
   printf("Wpisz tekst; "
          "# konczy program.\n");
   while ((ch = getchar()) != '#')
   {
      ch = toupper(ch);
      switch (ch)
      {
         case 'A' : a_licz++;
                    break;
         case 'E' : e_licz++;
```

```
                    break;
            case 'I' : i_licz++;
                    break;
            case 'O' : o_licz++;
                    break;
            case 'U' : u_licz++;
                    break;
            case 'Y' : y_licz++;
                    break;
            default  : break;
        }
    }
    printf("Liczba samoglosek: "
           "A = %d, E = %d, I = %d, "
           "O = %d, U = %d, Y = %d\n",
            a_licz, e_licz, i_licz,
            o_licz, u_licz, y_licz);
    return 0;
}
```

Wstęp do funkcji

```
#include <stdio.h>
double potega(double a, int b);
int main(void)
{
  double x, xpot;
  int n;
  printf("Podaj liczbe oraz potege "
         "naturalna. Wpisz q aby "
         "zakonczyc program.\n");
  while (scanf("%lf%d", &x, &n)==2)
  {
    xpot = potega(x, n);
    printf("%.3g do potegi %d to "
           "%.5g\n", x, n, xpot);
    printf("Podaj kolejna pare "
           "liczb lub wpisz q aby "
           "zakonczyc.\n");
  }
  return 0;
}
```

```
double potega(double a, int b)
{
  double pot = 1.0;
  int i;

  for (i = 1; i <= b; i++)
    pot *= a;
  return pot;
}
```

Zadanie. Na podstawie zadanych wartości x i y należy obliczyć

$$u = \max(x + y, xy),$$
$$v = \max(1/2, u).$$

Pierwsze podejście

```
#include <stdio.h>
int main(void)
{
   float x, y, u, v;

   scanf("%f %f", &x, &y);
   if (x + y > x * y)
      u = x + y;
   else
      u = x * y;
   if (0.5 > u)
      v = 0.5;
   else
      v = u;
   printf("u = %f, v = %f", u, v);

   return 0;
}
```

Istnieją powtórzenia

```
#include <stdio.h>
int main(void) {
   float x, y, u, v;
   float a, b, s;
   scanf("%f %f", &x, &y);
   a = x + y; b = x * y;
   if (a > b)
      s = a;
   else
      s = b;
   u = s;
   a = 0.5; b = u;
   if (a > b)
      s = a;
   else
      s = b;
   v = s;
   printf("u = %f, v = %f", u, v);
   return 0;
}
```

Jak ich uniknąć?

```
#include <stdio.h>
float max(float a, float b);
int main(void) {
   float x, y, u, v;

   scanf("%f %f", &x, &y);
   u = max(x + y, x * y);
   v = max(0.5, u);
   printf("u = %f, v = %f", u, v);
   return 0;
}

float max(float a, float b) {
   float s;
   if (a > b)
      s = a;
   else
      s = b;
   return s;
}
```

Wersja „kompakt”

```
#include <stdio.h>
float max(float, float);
int main(void)
{
   float x, y, u, v;

   scanf("%f %f", &x, &y);
   u = max(x + y, x * y);
   v = max(0.5, u);
   printf("u = %f, v = %f", u, v);
   return 0;
}

float max(float a, float b)
{
   return a > b ? a : b;
}
```

Największy wspólny dzielnik

```
#include <stdio.h>
float nwd(int, int);
int main(void) {
   int m, n, q;
   scanf("%d %d", &m, &n);
   q = nwd(m, n);
   printf("m = %d, n = %d, q = %d\n",
                    m, n, q);
   return 0;
}

float nwd(int a, int b) {
   while (a != b)
      if (a > b)
         a = a - b;
      else
         b = b - a;
   return a;
}
```

Można bez zwracania wartości...

```
#include <stdio.h>
#define WIERSZE 5
void n_znak(char, int);
int main(void)
{
   int i;
   for (i = 1; i <= WIERSZE; i++)
   {
      n_znak('#', i);
      n_znak('*', WIERSZE - i + 1);
      putchar('\n');
   }
   return 0;
}

void n_znak(char ch, int n) {
   int i;
   for (i = 1; i <= n; i++)
      putchar(ch);
}
```

Silnia — wersja klasyczna

```
#include <stdio.h>
long silnia(int);
int main(void)
{
   int num;
   scanf("%d", &num);
   printf("%d! = %ld\n",
            num, silnia(num));
   return 0;
}

long silnia(int n)
{
   long odp;
   for (odp = 1; n > 1; n--)
      odp *= n;
   return odp;
}
```

Silnia — wersja rekurencyjna

```
#include <stdio.h>
long silnia(int);
int main(void)
{
   int num;
   scanf("%d", &num);
   printf("%d! = %ld\n",
              num, silnia(num));
   return 0;
}

long silnia(int n)
{
   return n > 1 ? n * silnia(n - 1)
                : 1;
}
```

Zamiana wartości zmiennych

```
#include <stdio.h>
void zamiana(int, int);
int main(void) {
   int x = 5, y = 10;
   printf("x = %d, y = %d\n", x, y);
   zamiana(x, y);
   printf("x = %d, y = %d\n", x, y);
   return 0;
}

void zamiana(int u, int v) {
   int temp;
   temp = u;
   u = v;
   v = temp;
}
```

```
x = 5, y = 10
x = 5, y = 10
```

Wersja poprawiona

```
#include <stdio.h>
void zamiana(int *, int *);
int main(void) {
   int x = 5, y = 10;
   printf("x = %d, y = %d\n", x, y);
   zamiana(&x, &y);
   printf("x = %d, y = %d\n", x, y);
   return 0;
}

void zamiana(int * u, int * v) {
   int temp;
   temp = *u;
   *u = *v;
   *v = temp;
}
```

```
x = 5, y = 10
x = 10, y = 5
```