

# Header stdlib.h

Header	Opis zawartych funkcji	Przykładowe funkcje
stdlib.h	Funkcje biblioteki standardowej: zarządzanie pamięcią, narzędzia programowe, konwersje ciągów znaków, liczby losowe, algorytmy	atof(), atoi(), atol(), itoa(), ltoa(), strtod(), strtol(), strtoul(), strtoll(), strtoull() rand(), srand() malloc(), calloc(), realloc(), free(), <del>calloc()</del> abort(), exit(), system() qsort(), bsearch() abs(), labs(), div(), ldiv()

## #include<stdlib.h>

### Konwersje łańcuchów znakowych do formatów liczbowych i odwrotnie

- **atof(s)** - konwertuje ciąg s na *double* (liczbę podwójnej precyzji; NIE na *float*)
- **atoi(s)** - konwertuje ciąg s na liczbę całkowitą (**a**lphanumeric **t**o **i**nteger)
- **atol(s)** - konwertuje ciąg s na liczbę *long integer* (**a**lphanumeric **t**o **l**ong integer)
- **itoa(n, bufor, podstawa systemu)** - konwertuje liczbę całkowitą n z systemu liczbowego (zwykle '10' bo z systemu dziesiętnego) na ciąg (**i**nteger **t**o **a**lpha) i umieszcza go w buforze
- **ltoa(n, bufor, podstawa systemu)** - konwertuje liczbę n *long integer* z systemu liczbowego (zwykle '10' bo z systemu dziesiętnego) na ciąg (**l**ong **t**o **a**lpha) i umieszcza go w buforze
- **strtod(s, end pointer)** - konwertuje ciąg s na *double* (wymagana jest tu deklaracja tzw *end pointera*)
- **strtol(s, end pointer, podstawa systemu)** - konwertuje ciąg s (wymagana jest tu deklaracja tzw *end pointera*) na *long integer* określonej podstawy systemu liczbowego (zwykle '10' - dziesiętnego)
- **strtoul(s, end pointer, podstawa systemu)** - konwertuje ciąg s (wymagana jest tu deklaracja tzw *end pointera*) na *unsigned long integer* określonej podstawy systemu liczbowego (zwykle '10' - dziesiętnego)
- **strtoll(s, end pointer, podstawa systemu)** - konwertuje ciąg s (wymagana jest tu deklaracja tzw *end pointera*) na *long long integer* określonej podstawy systemu liczbowego (zwykle '10' - dziesiętnego)
- **strtoull(s, end pointer, podstawa systemu)** - konwertuje ciąg s (wymagana jest tu deklaracja tzw *end pointera*) na *unsigned long long integer* określonej podstawy systemu liczbowego (zwykle '10' - dziesiętnego)

Jeżeli ciąg zostaje zamieniony na liczbę to dodaję do niej 5 aby udowodnić, że to naprawdę jest liczba:

```
// Konwersje łańcuchów znakowych do formatów liczbowych i odwrotnie
```

```
#include <stdio.h>
#include <stdlib.h>           // ten temat
#include <locale.h>           // dla 'setlocale()'
```

```
int main()
{ setlocale(LC_CTYPE, "Polish"); //polskie znaki

  char s1[50] = "123.456789";
  char s2[50] = "123";
  char s3[50] = "11223344";
  int n1 = 555;
  char bufor[50];
  long int n2 = 11223344;
  char *eptr;
```

```

printf("\n - - - - - Funkcje biblioteki standardowej, stdlib.h - - - - -");
printf("\n * Konwersje łańcuchów znakowych do formatów liczbowych i odwrotnie\n\n");
printf("    Zamiana ciągu \"%s\" na liczbę 'double' + 5: %lf\n", s1, atof(s1) + 5);
printf("    Zamiana ciągu \"%s\" na liczbę całkowitą + 5: %d\n", s2, atoi(s2) + 5);
printf("    Zamiana ciągu \"%s\" na liczbę 'long integer' + 5: %ld\n\n", s3, atol(s3) + 5);

itoa(n1, bufor, 10);
printf("    Zamiana liczby całkowitej %d na ciąg: \"%s\"\n", n1, bufor );
ltoa(n2, bufor, 10);
printf("    Zamiana liczby 'long integer' %ld na ciąg: \"%s\"\n\n", n2, bufor );

printf("    Zamiana ciągu \"%s\" na liczbę 'double' + 5: %lf\n", s3, strtod(s3, &eptr) + 5);
printf("    Zamiana ciągu \"%s\" na liczbę 'long integer' + 5: %ld\n", s3, strtol(s3, &eptr, 10) + 5);
printf("    Zamiana ciągu \"%s\" na liczbę 'unsigned long integer' + 5: %lu\n", s3, strtoul(s3, &eptr,
10) + 5);
printf("    Zamiana ciągu \"%s\" na liczbę 'long long integer' + 5: %lld\n", s3, strtoll(s3, &eptr, 10)
+ 5);
printf("    Zamiana ciągu \"%s\" na liczbę 'unsigned long long integer' + 5: %llu\n", s3, strtoull(s3,
&eptr, 10) + 5);

return 0;
}

```

## Wynik działania programu:

```

- - - - - Funkcje biblioteki standardowej, stdlib.h - - - - -
* Konwersje łańcuchów znakowych do formatów liczbowych i odwrotnie

Zamiana ciągu "123.456789" na liczbę 'double' + 5: 123.456789
Zamiana ciągu "123" na liczbę całkowitą + 5: 128
Zamiana ciągu "11223344" na liczbę 'long integer' + 5: 11223349

Zamiana liczby całkowitej 555 na ciąg: "555"
Zamiana liczby 'long integer' 11223344 na ciąg: "11223344"

Zamiana ciągu "11223344" na liczbę 'double' + 5: 11223349.000000
Zamiana ciągu "11223344" na liczbę 'long integer' + 5: 11223349
Zamiana ciągu "11223344" na liczbę 'unsigned long integer' + 5: 11223349
Zamiana ciągu "11223344" na liczbę 'long long integer' + 5: 11223349
Zamiana ciągu "11223344" na liczbę 'unsigned long long integer' + 5: 11223349

-----
Process exited after 15.22 seconds with return value 0
Press any key to continue . . . _

```

## Generowanie sekwencji pseudolosowych

- **int rand(void)** - generuje liczbę pseudolosową
- **void srand(unsigned int seed)** - generuje liczbę pseudolosową *rand()* w zależności od ustawionej liczby *seed*. Sekwencja liczb jest powtarzalna w każdym uruchamianiu programu co ułatwia jego testowanie.

```

// Generowanie sekwencji pseudolosowych

#include <stdio.h>
#include <stdlib.h>           // ten temat

```

```

#include <locale.h> // dla 'setlocale()'

int main()
{
    setlocale(LC_CTYPE, "Polish"); //polskie znaki

    printf("\n - - - - - Funkcje biblioteki standardowej, stdlib.h - - - - -");
    printf("\n * Generowanie sekwencji pseudolosowych\n\n");
    printf(" Maksymalna wartość generowanej liczby losowej to %d\n", RAND_MAX);

    for (int i=0; i<2; i++)
    {
        printf("\n Ciąg liczb pseudolosowych: ");
        for (int j=0; j<6; j++) printf("%d ", rand() );
        printf("...");
    }

    printf("\n");

    for (int i=0; i<2; i++)
    {
        printf("\n Ciąg liczb pseudolosowych o takiej samej sekwencji w każdym ranowaniu: ");
        srand(100);
        for (int j=0; j<6; j++) printf("%d ", rand() );
        printf("...");
    }

    printf("\n");
    return 0;
}

```

## Wynik działania programu:

```

- - - - - Funkcje biblioteki standardowej, stdlib.h - - - - -
* Generowanie sekwencji pseudolosowych

Maksymalna wartość generowanej liczby losowej to 32767

Ciąg liczb pseudolosowych: 41 18467 6334 26500 19169 15724 ...
Ciąg liczb pseudolosowych: 11478 29358 26962 24464 5705 28145 ...

Ciąg liczb pseudolosowych o takiej samej sekwencji w każdym ranowaniu: 365 1216 5415 16704 24504 11254 ...
Ciąg liczb pseudolosowych o takiej samej sekwencji w każdym ranowaniu: 365 1216 5415 16704 24504 11254 ...

-----
Process exited after 10.37 seconds with return value 0
Press any key to continue . . . _

```

## Alokacja pamięci i cofanie alokacji pamięci

- **malloc**(liczba bajtów) - przydziela pamięć i zwraca wskaźnik do lokalizacji
- **calloc**(liczba bajtów) - to samo co *malloc()*, ale inicjuje pamięć za pomocą zer
- **realloc**(adres, liczba bajtów) - zmienia rozmiar bloku pamięci wcześniej przydzielony przez *malloc()* lub *calloc()*
- **free**(x) - zwalnia miejsce w pamięci obiektu x, już niepotrzebnego, do którego wcześniej przydzielono pamięć
- **coreleft()** - podaje ilość nieużytej jeszcze pamięci dostępnej dla działania programu

Funkcje te są dostępne w 'Turbo C++' w *header'ze* <alloc.h>.

W 'Dev C++' są one w <stdlib.h> z tym, że brakuje tu **int coreleft(void)** - funkcji podającej ilość pozostałej pamięci w obszarze działania programu.

**Uwaga:** Funkcja **coreleft()** jest nieobecna w 'Dev C++' ale bardzo ważna w "C". Dlaczego?

Każda aplikacja uruchomiona na komputerze rezerwuje sobie rozmiar tzw. pamięci operacyjnej. Powinna ona być wystarczająco duża aby obsługiwać wymagane przez użytkownika (Ciebie) zadania. Tymczasem nieograniczony tematycznie kompilator języka "C" taki jak Turbo C++ rezerwuje sobie deklarację jednego z tzw. modeli pamięci - *Tiny*, *Small* (domyślny), *Compact*, *Medium*, *Large* i *Huge*. Jeżeli chcesz wprowadzić do programu element wymagający rezerwacji pokażnej ilości pamięci (np. obraz/*image*), to sprawdź ile on zajmuje pamięci [ funkcją **sizeof()** ] i ile wolnej pamięci pozostało w 'przestrzeni operacyjnej' [ funkcją **coreleft()** ]. Ta ostatnia musi być większą liczbą. Jeżeli tak nie jest to zmień 'model' na większy.

Tak samo ważne jest uwolnienie pamięci obiektu już w programie nieużywanego [ funkcją **free()** ].

```
// Alokacja pamięci i cofanie alokacji pamięci

#include <stdio.h>
#include <stdlib.h>           // ten temat
#include <locale.h>          // dla 'setlocale()'

int main()
{
    setlocale(LC_CTYPE, "Polish"); //polskie znaki
    int pracownicy, *lista, suma = 0;

    printf("Wprowadź liczbę pracowników: ");
    scanf("%d", &pracownicy);
    // dynamiczne przydzielanie pamięci
    lista = (int*) malloc(pracownicy * sizeof(int));

    printf("\nPodaj płacę każdego pracownika z osobna w liczbach całkowitych: \n");
    for(int licznik = 0; licznik < pracownicy; licznik++)
    {
        printf("Pracownik nr %d : ", licznik+1);
        scanf("%d",&lista[licznik]);
        suma += lista[licznik];
    }
    // Tablicę lista[] można wykorzystać do takich obliczeń statystycznych jak:
    // 1. medianę płac
    // 2. przedziału dominanty płac
    // 3. odchylenia standardowego
    // 4. ...

    printf("\nŚrednia płaca wynosi %d PLN", suma/pracownicy);
    free(lista);
    return 0;
}
```

### Wynik działania programu:

Wprowadź liczbę pracowników: 5

Podaj płacę każdego pracownika z osobna w liczbach całkowitych:

Pracownik nr 1 : 1000

Pracownik nr 2 : 2000

Pracownik nr 3 : 3000

Pracownik nr 4 : 4000

Pracownik nr 5 : 5000

```
Średnia płaca wynosi 3000 PLN
```

```
-----
```

```
Process exited after 39.48 seconds with return value 0
```

```
Press any key to continue . . .
```

## Sterowanie procesami

- **abort()** - wymusza przerwanie działania programu bez wyczyszczenia: nie zamyka pliku, nie usuwa plików tymczasowych
- **exit()** - przerywa działanie programu z wyczyszczeniem bufora
- **system()** - pozwala na wykonanie poleceń zewnętrznych

## Sortowanie, wyszukiwanie

- **qsort()** - sortowanie tablicy
- **bsearch()** - wyszukiwanie binarne w tablicy

```
// qsort() i bsearch() dla tablicy liczb całkowitych i znaków

#include <stdio.h>
#include <stdlib.h>           // ten temat
#include <string.h>           // dla strlen()
#include <locale.h>           // dla 'setlocale()'

// ***** Operacje na liczbach *****
// Funkcja porównująca dwie liczby całkowite - zarówno dla qsort() jak i bsearch()
int porownaj_liczby(const void* a, const void* b)
{   return (*(int*)a - *(int*)b);
}

int tablica[] = { 178, 224, 3, 254, 5, 6, 136, 9, 8, 10,
                  127, 12, 348, 141, 15, 616, 17, 18, 169, 20,
                  21, 22, 623, 24, 525, 26, 27, 456, 29, 30,
                  128, 832, 733, 34, 35, 36, 37, 638, 39, 145,
                  41, 75, 43, 44, 445, 46, 145, 48, 949, 150,
                  51, 52, 53, 54, 55, 56, 57, 58, 59, 360,
                  61, 124, 63, 64, 65, 66, 67, 376, 69, 740,
                  129, 72, 73, 74, 42, 76, 77, 78, 79, 680,
                  81, 282, 83, 384, 85, 122, 87, 88, 89, 690,
                  191, 92, 493, 594, 97, 96, 98, 95, 123, 100
                };

// ***** Operacje na znakach *****
// Funkcja porównująca dwa znaki - zarówno dla qsort() jak i bsearch()
int porownaj_znaki(const void* x, const void* y)
{   return (*(char*)x - *(char*)y);
}

char zdanie[] = "the quick brown fox jumps over the lazy dog";

int main()
{   setlocale(LC_CTYPE, "Polish"); //polskie znaki

// ***** Operacje na liczbach *****
// Obliczanie ilości elementów tablicy liczb całkowitych
```

```

int rozmiar_elementu = sizeof(int); // rozmiar pojedynczego elementu
int liczba_elementow = sizeof(tablica) / rozmiar_elementu;

int szukany_element = 141; // szukana liczba (do wymiany)
int *rezultat; // pointer do rezultatu przeszukiwania

// qsort() dla sortowania tablicy
qsort(tablica, liczba_elementow, rozmiar_elementu, porownaj_liczby);

//Wypisz wynik sortowania
printf("\n Tablica posortowana:\n");
for (int i = 0; i < liczba_elementow; i++)
{
    printf("%5d", tablica[i]);
    if((i+1) % 10 == 0) printf("\n");
}

// rezultat przeszukiwania tablicy w celu znalezienia 'szukany_element' (= \
'rezultat')
rezultat = (int*)bsearch(&szukany_element, tablica, liczba_elementow,
rozmiar_elementu, porownaj_liczby);

if (rezultat) // to samo co: if (rezultat != NULL) nawet przy szukaniu zera
    printf("\nLiczba %d została znaleziona na pozycji %d tablicy posortowanej\n",
*rezultat, rezultat - tablica + 1);
else
    printf("\nLiczby %d nie ma w tablicy\n", szukany_element);

// ***** Operacje na znakach *****
int rozmiar_elementu_znak = sizeof(char); // rozmiar pojedynczego elementu

char szukany_element_znak = 't'; // szukana litera (do wymiany)
char *rezultat_znak; // pointer do rezultatu przeszukiwania

// Obliczanie ilości znaków (elementów zdania, tablicy znaków)
liczba_elementow = strlen(zdanie);

// qsort() dla sortowania tablicy znaków
qsort(zdanie, liczba_elementow, sizeof(char), porownaj_znaki); // sizeof(char)=1

//Wypisz wynik sortowania bez powtórzeń i bez spacji
printf("\n\n Alfabet: ");
for (int i = 0; i < liczba_elementow; i++)
{
    if((i > 0 && zdanie[i] == zdanie[i-1]) || zdanie[i] == ' ') continue;
    printf("%c", zdanie[i]);
}

//Wypisz wynik sortowania z powtórzeniami w tym ze spacjami
printf("\n Znaki z powtórzeniami w tym ze spacjami: \n");
for (int i = 0; i < liczba_elementow; i++)
    printf("%c", zdanie[i]);
printf("\n");

// rezultat przeszukiwania tablicy w celu znalezienia 'szukany_element' (= \
'rezultat_znak')
rezultat_znak = (char*)bsearch(&szukany_element_znak, zdanie, liczba_elementow,
rozmiar_elementu_znak, porownaj_znaki);

if (rezultat_znak) // to samo co: if (rezultat_znak != NULL)
    printf("\nZnak %c został znaleziony na pozycji %d tablicy posortowanej\n",
*rezultat_znak, rezultat_znak - zdanie + 1);

```

```

else
    printf("\nZnak %c nie ma w tablicy znaków\n", szukany_element_znak);

return 0;
}

```

## Wynik działania programu:

```

Tablica posortowana:
  3   5   6   8   9  10  12  15  17  18
 20  21  22  24  26  27  29  30  34  35
 36  37  39  41  42  43  44  46  48  51
 52  53  54  55  56  57  58  59  61  63
 64  65  66  67  69  72  73  74  75  76
 77  78  79  81  83  85  87  88  89  92
 95  96  97  98 100 122 123 124 127 128
129 136 141 145 145 150 169 178 191 224
254 282 348 360 376 384 445 456 493 525
594 616 623 638 680 690 733 740 832 949

Liczba 141 została znaleziona na pozycji 73 tablicy posortowanej

Alfabet: abcdefghijklmnopqrstuvwxyz
Znaki z powtórzeniami w tym ze spacjami: '      abcdeefghhijklmnooopqrrsttuuvwxyz'
Znak t został znaleziony na pozycji 35 tablicy posortowanej

-----
Process exited after 11.69 seconds with return value 0
Press any key to continue . . .

```

Proponuję swoje własne przeszukiwanie binarne:

```

// Przeszukiwanie binarne
// Warunek: Dane muszą być wstępnie posortowane (tu: użyłem sortowanie bąbelkowe)
// Proces: Dane zostaną podzielone na pół (stąd nazwa: binarne/dwójkowe) i program
//         sprawdza, w którym z tych przedziałów może znaleźć się szukany element.
//         Czy element nie jest na granicy wybranego przedziału? Jeżeli nie, to...
//         ... znowu podzielony zostaje wybrany przedział na pół ... itd. aż...
//         ... szukany element znajdzie się na granicy przedziału (= granicznemu)
//         lub graniczne elementy się zcałq co znaczy, że elementu nie ma w danych.

#include<stdio.h>
#include <locale.h>           // dla 'setlocale()'

#define MAX 100

int a[MAX] = { 178, 224, 3, 254, 5, 6, 136, 9, 8, 10,
               127, 12, 348, 141, 15, 616, 17, 18, 169, 20,
               21, 22, 623, 24, 525, 26, 27, 456, 29, 30,
               128, 832, 733, 34, 35, 36, 37, 638, 39, 145,
               41, 75, 43, 44, 445, 46, 145, 48, 949, 150,
               51, 52, 53, 54, 55, 56, 57, 58, 59, 360,
               61, 124, 63, 64, 65, 66, 67, 376, 69, 740,
               129, 72, 73, 74, 42, 76, 77, 78, 79, 680,
               81, 282, 83, 384, 85, 122, 87, 88, 89, 690,
               191, 92, 493, 594, 97, 96, 98, 95, 123, 100
               };

int znalazlem = 0;
int i,j;

```

```

int x = 101;           // szukana liczba (do wymiany)
int ilosc_krokow = 0;
int temp;
int min, max, polowa;

int main()
{
    setlocale(LC_CTYPE, "Polish"); //polskie znaki

    for(i = 0; i < MAX-1; i++)      // sortowanie bąbelkowe
        for(j = i+1; j < MAX; j++)
            if(a[i] > a[j])
            {
                temp = a[j];
                a[j] = a[i];
                a[i] = temp;
            }

    printf("\n Tablica posortowana:\n");
    for (i = 0; i < MAX; i++)
    {
        printf("%5d", a[i]);
        if((i+1) % 10 == 0) printf("\n");
    }

    // przeszukiwanie binarne
    min = 0;
    max = MAX-1;

    do
    {
        ilosc_krokow++;
        polowa = (min + max) / 2;

        if(x == a[min])
        {
            znalazlem = 1; j = min; break; }           // znalazł

        if(x == a[max])
        {
            znalazlem = 1; j = max; break; }           // znalazł

        if(min + 1 == max || min == max) break;        // ostatecznie nie znalazł

        if(x > a[min] && x < a[polowa]) max = polowa; // szuka dalej
        else (min = polowa);

    } while (znalazlem == 0);

    if(znalazlem == 1)
    {
        printf("\n\n Liczba %d jest w tablicy", x);
        printf("\n\n Liczba została znaleziona na pozycji %d tablicy posortowanej",
j+1);
    }
    else
        printf("\n\n Liczby %d nie ma w tablicy", x);

    printf("\n Ilość kroków = %d \n", ilosc_krokow);
}

```

**Wynik działania programu:**

```

Tablica posortowana:
 3   5   6   8   9  10  12  15  17  18

```



20	21	22	24	26	27	29	30	34	35
36	37	39	41	42	43	44	46	48	51
52	53	54	55	56	57	58	59	61	63
64	65	66	67	69	72	73	74	75	76
77	78	79	81	83	85	87	88	89	92
95	96	97	98	100	122	123	124	127	128
129	136	141	145	145	150	169	178	191	224
254	282	348	360	376	384	445	456	493	525
594	616	623	638	680	690	733	740	832	949

Liczby 101 nie ma w tablicy  
Ilość kroków = 7

-----  
Process exited after 12.01 seconds with return value 0  
Press any key to continue . . .

## Matematyka

- **abs(int x)** - podaje wartość bezwzględną liczby całkowitej *x*
- **labs(long int x)** - podaje wartość bezwzględną liczby *long integer* *x*
- **div(int x, int y)** - dzielenie liczb całkowitych *x* przez *y* (oblicza iloraz i resztę)

Skopiowane z stdlib.h

```
typedef struct {
    int    quot;
    int    rem;
} div_t;
div_t     _Cdecl div(int __numer, int __denom);
```

Dzieli *numer* (dzielna, licznik) przez *denom* (dzielnik, mianownik) dając część całkowitą 'quot' i resztę 'rem'.

- **ldiv(long int x, long int y)** - dzielenie liczb *long integer* *x* przez *y* (oblicza iloraz i resztę)

Skopiowane z stdlib.h

```
typedef struct {
    long    quot;
    long    rem;
} ldiv_t;
ldiv_t     _Cdecl ldiv(long __numer, long __denom);
```

Dzieli *numer* (dzielna, licznik) przez *denom* (dzielnik, mianownik) dając część całkowitą 'quot' i resztę 'rem'.

// Matematyka

```
#include <stdio.h>
#include <stdlib.h>           // ten temat
#include <locale.h>           // dla 'setlocale()'

int main()
{  setlocale(LC_CTYPE, "Polish"); //polskie znaki

////////// wartość bezwzględna liczby //////////
    printf("\n\t Wartość bezwzględna liczb całkowitych\n");
// ***** dla liczb całkowitych *****
    int liczba = -33;
```

```

printf("Wartość bezwzględna liczby %d to %d.\n", liczba, abs(liczba) );

// ***** dla długich liczb całkowitych *****
long int liczba_long = -11112222;
printf("Wartość bezwzględna liczby %ld to %ld.\n", liczba_long, abs(liczba_long) );

////////// dzielenie liczb całkowitych //////////
printf("\n\tDzielenie liczb całkowitych\n");
// ***** dla liczb całkowitych *****
int dzielna = 121, dzielnik = 35; // zmienne do wymiany
div_t rezultat_dzielenia; // obiekt typu div_t, który jest strukturą
// o zmiennych 'quot' i 'rem'
rezultat_dzielenia = div (dzielna, dzielnik);
printf ("%d podzielone przez %d daje %d całe i %d reszty.\n", dzielna, dzielnik,
        rezultat_dzielenia.quot, rezultat_dzielenia.rem);

// ***** dla długich liczb całkowitych *****
long int dzielna_long = 11112222, dzielnik_long = 121;
ldiv_t rezultat_dzielenia_long; // obiekt typu ldiv_t, który jest strukturą
// o zmiennych 'quot' i 'rem'

rezultat_dzielenia_long = ldiv (dzielna_long, dzielnik_long);
printf ("%d podzielone przez %d daje %d całe i %d reszty.\n", dzielna_long,
        dzielnik_long, rezultat_dzielenia_long.quot, rezultat_dzielenia_long.rem);

return 0;
}

```

## Wynik działania programu:

```

        Wartość bezwzględna liczb całkowitych
Wartość bezwzględna liczby -33 to 33.
Wartość bezwzględna liczby -11112222 to 11112222.

        Dzielenie liczb całkowitych
121 podzielone przez 35 daje 3 całe i 16 reszty.
11112222 podzielone przez 121 daje 91836 całe i 66 reszty.

-----
Process exited after 13 seconds with return value 0
Press any key to continue . . .

```