

# Header time.h

Header	Opis zawartych funkcji	Przykładowe funkcje
time.h	Makro, struktury, definicje typu i funkcje daty i czasu	CLOCKS_PER_SEC tm{}, size_t, time_t, clock_t, asctime(), ctime(), gmtime(), clock(), difftime(), time(), localtime(), mktime(), strftime()

W uruchamianym programie warto podać, oprócz nazwy programu, datę i czas uruchamiania programu tworzących dane. Co prawda robią to już predefiniowane makra (`__FILE__`, `__DATE__`, `__TIME__`) ale **time.h** daje pod tym względem znacznie więcej możliwości.

## #include<time.h>

**time.h** definiuje cztery podstawowe typy danych: struct **tm**, typedef **size\_t**, **time\_t** i **clock\_t** w następujący sposób:

```
struct tm                // struktura definiująca zmienne daty i czasu jak w kalendarzu
{
int tm_sec;              // liczba sekund [0; 59]
int tm_min;              // liczba minut [0; 59]
int tm_hour;             // liczba godzin [0; 23]
int tm_mday;             // liczba dnia miesiąca [1; 31]
int tm_mon;              // liczba miesiąca [0 (styczeń); 11 (grudzień)]
int tm_year;             // liczba lat od roku 1900
int tm_wday;             // liczba dnia tygodnia [0 (niedziela); 6 (sobota)]
int tm_yday;             // liczba dnia roku (kalendarz Juliański) [1; 365]
int tm_isdst;            // liczba określająca letnie/zimowe przesunięcie czasowe
};
```

Powyższa struktura jest wypełniana danymi aktualnej daty i czasu. Dane te można powielić innymi danymi tak, że wymuszają one wzajemną spójność (np. podanie daty wymusi poprawny dzień tygodnia).

```
typedef unsigned size_t;    // typ całkowity bez znaku i jest wynikiem słowa kluczowego sizeof
typedef long time_t;        // typ odpowiedni do przechowywania daty i czasu kalendarzowego
typedef long clock_t;       // typ odpowiedni do przechowywania czasu procesora
```

Na bazie wyżej wymienionych obiektów zdefiniowane są następujące funkcje:

```
char *    _Cdecl asctime(const struct tm *__tblock);
char *    _Cdecl ctime(const time_t *__time);
double    _Cdecl difftime(time_t __time2, time_t __time1);
struct tm *_Cdecl gmtime(const time_t *__timer);
struct tm *_Cdecl localtime(const time_t *__timer);
time_t     _Cdecl time(time_t *__timer);
time_t     _Cdecl mktime(struct tm *__timeptr);
clock_t    _Cdecl clock(void);
size_t     _Cdecl strftime(char *__s, size_t __maxsize,
                           const char *__fmt, const struct tm *__t);
```

Stała makro:

- **CLOCKS\_PER\_SEC** – Liczba taktów zegara procesora (systemowego) na sekundę

Funkcje:

- **asctime(struct tm \*tp)** – wyszczególnia aktualne wartości elementów struktury **tm** z góry przyjętej kolejności wykorzystując pointer (wskaźnik) do tej struktury
- **ctime(time\_t \*tp)** – działa podobnie jak **asctime()**, ale nie odwołuje się do struktury **tm** lecz do **time\_t**
- **gmtime(time\_t \*tp)** – działa podobnie jak **asctime()** i **ctime()** ale podaje czas skoordynowany z Greenwich (UTC, Greenwich Mean Time - GMT)

```
Aktualna data i czas według...
asctime() : Thu May 09 00:22:13 2024
ctime()   : Thu May 09 00:22:13 2024
gmtime()  : Wed May 08 22:22:13 2024
```

- **clock()** – podaje czas zegara procesora zużyty od początku uruchomienia programu w liczbie taktów tego zegara. Ta liczba podzielona przez **CLOCKS\_PER\_SEC** dałaby czas w sekundach.
- **difftime(time\_t punkt2, time\_t punkt1)** – oblicza różnicę czasową w sekundach pomiędzy dwoma punktami, w których odmierzany jest czas
- **time(time\_t \*t)** – podaje ilość sekund w odniesieniu do 1 stycznia 1970 roku i wprowadza go pod adres **t** pointera (wskaźnika)
- **localtime(struct tm \*tp)** – wartości struktury **tm** są wyrażane w lokalnej strefie czasowej
- **mktime(time\_t \*tp)** – ma za zadanie przyswoić wprowadzone do zmiennych struktury **tm** przez użytkownika dane na zsynchronizowane dane kalendarzowe (np. dzień tygodnia będzie zgodny z wprowadzoną przez użytkownika datą)
- **strftime(bufor łańcucha znakowego utworzonego zgodnie z 'formatem wydruku' gotowy do wydruku, rozmiar bufora do wydruku (może on nie być całym), format wydruku i zgodnie z nim będzie wyglądał łańcuch znakowy 'bufora', struct tm)** – pozwala formatować wydruk, korzystając ze specyfikatorów, które określają zarówno typ pojedynczej danej jak i jej format (np. %H to godzina formatu 24 godzinowego, %I - 12 godzinowego itd.) co pozwala na wzbogacenie środków przekazu danych kalendarzowych

---

## CLOCKS\_PER\_SEC

Plik nagłówkowy **time.h** Dev C++ sprawia przykrą niespodziankę ustalając makro **CLOCKS\_PER\_SEC** na 1000

```
#define CLOCKS_PER_SEC 1000
```

wobec czego wszelkie operacje na **CLOCKS\_PER\_SEC** tracą sens.

Można tylko odczytać prawdziwą częstotliwość taktowania zegara systemowego tak:

- 'Wyszukaj' **system**



Odczytaj prawdziwą wartość **CLOCKS\_PER\_SEC**

Procesor Intel(R) Core(TM) i7-3630QM CPU  
@ 2.40GHz 2.40 GHz

A więc u mnie 2.40 GHz = 2 400 000 000 CLOCKS\_PER\_SEC

---

```
struct tm, time_t,  
time(), localtime(), asctime(),  
time(), difftime(), clock()
```

Niemal każde użycie powyższych funkcji wymaga odwołania się do jednych z czterech podstawowych typów danych: struct **tm**, typedef **size\_t**, **time\_t** i **clock\_t**. Wobec czego, nie dziw się, że niemal każdy tu przedstawiony program zaczyna się od tych deklaracji:

```
#include<stdio.h>  
#include<time.h>           // ten temat  
#include<locale.h>         // dla 'setlocale()'  
  
time_t data_i_czas; // deklaruj zmienną o nazwie 'data_i_czas' typu czasowego 'time_t'  
time(&data_i_czas); // wprowadź pod adres zmiennej 'data_i_czas' bieżący czas  
                    // odczytany w ten sposób z systemu (time); zarówno  
                    // 'data_i_czas' jak i 'time' są tego samego typu (time_t)  
struct tm *tp;      // deklaracja pointera - ma mieć długość struktury tm  
                    // tp sugeruje nazwę 'time pointer' ('time' od nazwy header'a)  
tp = localtime(&data_i_czas); // przypisanie pointerowi funkcji 'localtime' na  
                               // bieżących dacie i czasie systemu  
                               // zarówno 'tp' jak i 'localtime' są tego samego typu (struct tm)  
                               // a więc występuje wymagana, przy podstawieniu, zgodność typu
```

```
// Data i czas  
  
#include <stdio.h>  
#include <time.h>           // ten temat  
  
int main()  
{  
printf("\n ----- struct tm, time_t -----");  
printf("\n ----- funkcje time() i localtime() -----");  
    time_t data_i_czas;  
    time( &data_i_czas );  
    struct tm *tp;  
    tp = localtime( &data_i_czas );  
  
printf("\n ----- asctime() ----- \n");  
    printf("Aktualna data i czas: %s \n", asctime( tp ));  
    return 0;  
}
```

## Wynik działania programu:

```
Aktualna data i czas: Wed May 08 17:59:30 2024
```

A są to elementy struktury **tm** w kolejności nieco chaotycznej:

- |   |              |                                   |   |
|---|--------------|-----------------------------------|---|
| 1 | int tm_wday; | // Dzień tygodnia. Zakres [0; 6]. | 0 - Sunday, 6 - Saturday <b>ale słownie</b>   |
| 2 | int tm_mon;  | // Miesiąc. Zakres [0; 11].       | 0 - January, 11 - December <b>ale słownie</b> |

```

3  int tm_mday;    // Dzień miesiąca. Zakres [1..31]
4  int tm_hour;    // Godziny. Zakres [0; 23]
5  int tm_min;     // Minuty. Zakres [0; 59]
6  int tm_sec;     // Sekundy. Zakres [0; 59]
7  int tm_year;    // Obecny rok. Lata zaczynają się liczyć od roku 1900, czyli: wartość 0 to 1900 rok.

```

Zróbmy asctime() po naszymu.

```

// Data i czas

#include <stdio.h>
#include <time.h>           // ten temat
#include <locale.h>         // dla 'setLocale()'

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

    const char *dzien_slownie[] = {"Niedziela", "Poniedziałek", "Wtorek",
                                    "Środa", "Czwartek", "Piątek", "Sobota"};

    const char *miesiac_slownie[] = {"stycznia", "lutego", "marca", "kwietnia",
                                      "maja", "czerwca", "lipca", "sierpnia",
                                      "września", "października", "listopada", "grudnia"};

    time_t data_i_czas;
    time( &data_i_czas );
    struct tm *tp;
    tp = localtime( &data_i_czas );

    printf("\n\n ----- Zamiana asctime() na coś lepszego ----- \n");
    // mktime( tp ); <--- tu jest zbyteczne bo weryfikacja jest natychmiastowa
    printf("\n Aktualna data i czas: %s, %d %s %d, godzina %2d:%2d:%2d \n"
           , dzien_slownie[tp -> tm_wday]
           , tp -> tm_mday
           , miesiac_slownie[tp -> tm_mon]
           , tp -> tm_year + 1900
           , tp -> tm_hour
           , tp -> tm_min
           , tp -> tm_sec );

    return 0;
}

```

**Wynik działania programu:**

```

----- Zamiana asctime() na coś lepszego -----

Aktualna data i czas: Piątek, 10 maja 2024, godzina 18: 7:18

-----

Process exited after 10.92 seconds with return value 0
Press any key to continue . . . _

```

A gdyby się odwołać do przeszłości lub przyszłości (np. czy następny Nowy Rok będzie w niedzielę?).

Wystarczy powielić **tm\_mday**, **tm\_mon** i **tm\_year** własnymi danymi i uzyskać **tm\_wday** a tym samym **dzien\_slownie**. Resztę danych można zignorować.

W praktyce **mktime()** działa od 1 stycznia 1970 roku czyli od kiedy pracuje funkcja **time()** mierząca ilość sekund od tego momentu do teraz.

```
// Data i czas

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

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

    int sprawdzam;

    const char *dzien_slownie[] = {"Niedziela", "Poniedziałek", "Wtorek",
                                    "Środa", "Czwartek", "Piątek", "Sobota"};

    const char *miesiac_slownie[] = {"stycznia", "lutego", "marca", "kwietnia",
                                       "maja", "czerwca", "lipca", "sierpnia",
                                       "września", "października", "listopada", "grudnia"};

    int dzien, miesiac, rok;
    time_t poczatek, koniec; // dla zmierzenia czasu procesu w sekundach

    time_t data_i_czas;
    time( &data_i_czas );
    struct tm *tp;
    tp = localtime( &data_i_czas );

    printf("\n\n ----- Znajdywanie dnia tygodnia ----- \n");
    printf(" ----- Przy okazji: Pomiar czasu procesu time() i difftime() ----- \n");
    printf(" ----- Liczba taktów zegara systemowego - clock() ----- \n");

    time( &poczatek );

    printf("\nPodaj dzień, miesiąc i rok oddzielając liczby spacją: ");
    scanf("%d %d %d", &dzien, &miesiac, &rok);

    // Tu muszą się znaleźć wartości dla systemu:
    // miesiac_slownie to tablica: styczeń=0, ... grudzień=11 więc trzeba odjąć 1
    // rok 1900 to rok nr 0, więc np. rok 2024 to dla systemu ma być rok 124
    tp -> tm_mday = dzien;
    tp -> tm_mon = miesiac - 1; // 'tm_mon' to tablica: styczeń = 0, itd.
    tp -> tm_year = rok - 1900; // bo rok 1900 to rok nr 0 dla 'tm_year'

    sprawdzam = mktime( tp );
    if( sprawdzam == -1 )
    { printf("\nBłąd: Nieudana konwersja danych daty i czasu przez funkcję \
```

```

mktime(). \n");
} else
    // 'tm_year' daje liczbę dwucyfrową, stąd wybór zmiennej 'rok'
    printf("\n %d %s %d roku to %. \n", tp -> tm_mday,
        miesiac_slownie[tp -> tm_mon], rok, dzien_slownie[tp -> tm_wday]);

time( &koniec );

printf("\nCzas procesu wyniósł %.2f sekund.\n", difftime(koniec, poczatek));
printf("\nLiczba taktów zegara systemowego od początku procesu wyniosła %ld.\n",
clock());

return 0;
}

```

### Przykładowe wyniki działania programu:

```

----- Znajdywanie dnia tygodnia -----
----- Przy okazji: Pomiar czasu procesu time() i difftime() -----
----- Liczba taktów zegara systemowego - clock() -----

Podaj dzień, miesiąc i rok oddzielając liczby spacją: 16 10 1978

16 października 1978 roku to Poniedziałek.

Czas procesu wyniósł 8.00 sekund.

Liczba taktów zegara systemowego od początku procesu wyniosła 8324.

-----
Process exited after 7.914 seconds with return value 0
Press any key to continue . . . _

Powyżej jest dzień wybrania kardynała Karola Józefa Wojtyły na papieża

1 maja 2004 roku to Sobota.
Wejście Polski do Unii Europejskiej.

2 kwietnia 2005 roku to Sobota.
Śmierć papieża Jana Pawła II.

10 kwietnia 2010 roku to Sobota.
Katastrofa lotnicza pod Smoleńskiem.

1 stycznia 2025 roku to Środa.
Nowy Rok 2025.

```

Weryfikacja powyższych wyników z programu zbudowanego na bazie definicji kalendarza gregoriańskiego:

Kalendarz Gregoriański dowolnego miesiąca	Kalendarz Gregoriański dowolnego miesiąca	Kalendarz Gregoriański dowolnego miesiąca	Kalendarz Gregoriański dowolnego miesiąca
Miesiąc (1..12) : 10 Rok (od 1582) : 1978	Miesiąc (1..12) : 5 Rok (od 1582) : 2004	Miesiąc (1..12) : 4 Rok (od 1582) : 2005	Miesiąc (1..12) : 4 Rok (od 1582) : 2010
10 1978	5 2004	4 2005	4 2010
Po Wt Śr Cz Pi So Ni	Po Wt Śr Cz Pi So Ni	Po Wt Śr Cz Pi So Ni	Po Wt Śr Cz Pi So Ni
1	1 2	1 2 3	1 2 3 4
2 3 4 5 6 7 8	3 4 5 6 7 8 9	4 5 6 7 8 9 10	5 6 7 8 9 10 11
9 10 11 12 13 14 15	10 11 12 13 14 15 16	11 12 13 14 15 16 17	12 13 14 15 16 17 18
16 17 18 19 20 21 22	17 18 19 20 21 22 23	18 19 20 21 22 23 24	19 20 21 22 23 24 25
23 24 25 26 27 28 29	24 25 26 27 28 29 30	25 26 27 28 29 30	26 27 28 29 30
30 31	31		

2025			2025		
<b>Styczeń</b>	<b>Luty</b>	<b>Marzec</b>	<b>Lipiec</b>	<b>Sierpień</b>	<b>Wrzesień</b>
Po Wt Śr Cz Pi So Ni	Po Wt Śr Cz Pi So Ni	Po Wt Śr Cz Pi So Ni	Po Wt Śr Cz Pi So Ni	Po Wt Śr Cz Pi So Ni	Po Wt Śr Cz Pi So Ni
1 2 3 4 5	1 2	1 2	1 2 3 4 5 6	1 2 3	1 2 3 4 5 6 7
6 7 8 9 10 11 12	3 4 5 6 7 8 9	3 4 5 6 7 8 9	7 8 9 10 11 12 13	4 5 6 7 8 9 10	8 9 10 11 12 13 14
13 14 15 16 17 18 19	10 11 12 13 14 15 16	10 11 12 13 14 15 16	14 15 16 17 18 19 20	11 12 13 14 15 16 17	15 16 17 18 19 20 21
20 21 22 23 24 25 26	17 18 19 20 21 22 23	17 18 19 20 21 22 23	21 22 23 24 25 26 27	18 19 20 21 22 23 24	22 23 24 25 26 27 28
27 28 29 30 31	24 25 26 27 28	24 25 26 27 28 29 30 31	28 29 30 31	25 26 27 28 29 30 31	29 30
<b>Kwiecień</b>	<b>Maj</b>	<b>Czerwiec</b>	<b>Październik</b>	<b>Listopad</b>	<b>Grudzień</b>
Po Wt Śr Cz Pi So Ni	Po Wt Śr Cz Pi So Ni	Po Wt Śr Cz Pi So Ni	Po Wt Śr Cz Pi So Ni	Po Wt Śr Cz Pi So Ni	Po Wt Śr Cz Pi So Ni
1 2 3 4 5 6	1 2 3 4	1	1 2 3 4 5	1 2	1 2 3 4 5 6 7
7 8 9 10 11 12 13	5 6 7 8 9 10 11	2 3 4 5 6 7 8	6 7 8 9 10 11 12	3 4 5 6 7 8 9	8 9 10 11 12 13 14
14 15 16 17 18 19 20	12 13 14 15 16 17 18	9 10 11 12 13 14 15	13 14 15 16 17 18 19	10 11 12 13 14 15 16	15 16 17 18 19 20 21
21 22 23 24 25 26 27	19 20 21 22 23 24 25	16 17 18 19 20 21 22	20 21 22 23 24 25 26	17 18 19 20 21 22 23	22 23 24 25 26 27 28
28 29 30	26 27 28 29 30 31	23 24 25 26 27 28 29 30	27 28 29 30 31	24 25 26 27 28 29 30	29 30 31
Przyciśnij dowolny klawisz aby zobaczyć następne półrocze			[Q] - wyjście, [SPACJA] - pierwsza strona, [inny klawisz] - rozpocznij od nowa		

**strftime()**  
i specyfikatory tej funkcji

```
// Data i czas

#include <stdio.h>
#include <time.h>
#include <string.h>
#include <locale.h>

// ten temat
// dla memset() i strlen()
// dla 'setlocale()'

char bufor[100];

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

    time_t data_i_czas;
    time( &data_i_czas );
    struct tm *tp;
    tp = localtime( &data_i_czas );

    printf("\n ----- strftime() ----- \n");
    strftime( bufor, 60, "Sformatowane domyślnie data i czas %c", tp );
}
```



```

printf(" Przykład nr 1 : %s \n", bufor);

memset(bufor,0,strlen(bufor));           // wyczyść bufor
strftime( bufor, 60, "Dzisiaj jest %A, %Y-%m-%d, godzina %X", tp );
printf(" Przykład nr 2 : %s \n", bufor);

memset(bufor,0,strlen(bufor));           // wyczyść bufor
strftime( bufor, 100, "Dzisiaj jest kolejny %j dzień, %m miesiąc, %W tydzień, \
%w dzień tygodnia %Y roku", tp );
printf(" Przykład nr 3 : %s \n", bufor);

return 0;
}

```

Wszystkie specyfikatory funkcji `strftime()` zaczynają się od znaku %.

Jeżeli ma być wydrukowany znak %, to należy podwoić ten znak (%%) co jest niemal standardem w różnych językach programowania.

Większość specyfikatorów nie działa w Dev C++. Brak takiego działania dyskwalifikuje wydruk całej linii (u nas wartości łańcucha znakowego 'bufor').

Poniżej wyszczególnione są tylko te specyfikatory, które rozpoznaje Dev C++.

Specyfikator	Co oznacza specyfikator	Przykład
%a	Słowna nazwa dnia tygodnia w skrócie	Sun
%A	Pełna słowna nazwa dnia tygodnia	Sunday
%b	Słowna nazwa miesiąca w skrócie	Sep
%B	Pełna słowna nazwa miesiąca	September
%c	Sformatowana domyślnie data i czas	02/14/21 02:52:02
%d	Numer dnia miesiąca z wiodącym zerem [01; 31]	07
%H	Godzina formatu 24-godzinnego [00; 23]	16
%I	Godzina formatu 12-godzinnego [01; 12]	04
%j	Numer dnia roku ( data juliańska, [001; 366] )	124
%m	Numer miesiąca [01; 12]	06
%M	Minuty [00; 59]	36
%p	Symbole AM lub PM (dla formatu 12-godzinnego)	PM
%S	Sekundy [00; 61]	04
%U	Numer tygodnia z pierwszą niedzielą jako pierwszym dniem tygodnia [00; 53]	21
%w	Numer dnia tygodnia z 0 (zerem) dla niedzieli [0; 6]	4
%W	Numer tygodnia z pierwszym poniedziałkiem jako pierwszym dniem tygodnia [00; 53]	23
%X	Sformatowane domyślnie przedstawienie czasu	05:45:07



%y	Rok - jego ostatnie dwie cyfry [00; 99]	09
%Y	Rok - pełna liczba	2009

## Przykładowe wyniki działania programu:

```
----- strftime() -----
Przykład nr 1 : Sformatowane domyślnie data i czas 05/13/24 00:52:01
Przykład nr 2 : Dzisiaj jest Monday, 2024-05-13, godzina 00:52:01
Przykład nr 3 : Dzisiaj jest kolejny 134 dzień, 05 miesiąc, 20 tydzień, 1 dzień tygodnia 2024 roku

-----
Process exited after 0.5024 seconds with return value 0
Press any key to continue . . . _
```

Plik nagłówkowy `locale.h` posiada jedną z 'kategorii' o nazwie `LC_TIME`, która współpracując z `strftime()` pozwala na uzyskanie polskich nazw, np. dla powyższego przykładu nr 2, gdzie zamiast *Monday* będzie *poniedziałek*. Patrz: Plik nagłówkowy **locale.h**.

## Podsumowanie

```
// Data i czas

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

char bufor[100];

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

    time_t data_i_czas;
    time( &data_i_czas );
    struct tm *tp;
    tp = localtime( &data_i_czas );

    printf("\n ----- asctime(), ctime(), gmtime() ----- \n");

    printf("Aktualna data i czas według...\n");
    printf("    asctime()    : %s", asctime( tp ) );
    printf("    ctime()      : %s", ctime( &data_i_czas ) );
    tp = gmtime( &data_i_czas );
    printf("    gmtime()     : %s", asctime( tp ) );

    printf("\n ----- time(), localtime(), clock() ----- \n");

    printf("Tyle sekund upłynęło od 1 stycznia 1970 roku \n");
    printf("    time() : %ld \n", time(NULL) );
    printf("\nZ całej struktury typu tm wybrano godzinę - ma być taka, jak na Twoim \
zegarku \n");
    tp = localtime( &data_i_czas ); // trzeba odnowić pointer tp bo był gmtime()
    printf("    localtime() : %2d:%2d:%2d \n", tp->tm_hour, tp->tm_min, tp->tm_sec );
    printf("\nLiczba taktów zegara systemowego od początku procesu \n");
    printf("    clock()      : %ld \n", clock() );
```

```

printf("\n ----- strftime() ----- \n");
    strftime( bufor, 60, "Podaję aktualny czas w stylu angielskim %I:%M:%S %p", tp );
    printf("    strftime() : %s \n", bufor);

    return 0;
}

```

### Przykładowe wyniki działania programu:

```

----- asctime(), ctime(), gmtime() -----
Aktualna data i czas według...
    asctime()   : Sat May 11 14:46:58 2024
    ctime()     : Sat May 11 14:46:58 2024
    gmtime()    : Sat May 11 12:46:58 2024

----- time(), localtime(), clock() -----
Tyle sekund upłynęło od 1 stycznia 1970 roku
    time()      : 1715431618

Z całej struktury typu tm wybrano godzinę - ma być taka, jak na Twoim zegarku
    localtime() : 14:46:58

Liczba taktów zegara systemowego od początku procesu
    clock()     : 41

----- strftime() -----
    strftime()  : Podaję aktualny czas w stylu angielskim 02:46:58 PM

-----
Process exited after 10.38 seconds with return value 0
Press any key to continue . . . _

```