Header locale.h

Header	Opis zawartych funkcji	Podstawowa struktura i funkcje
locale.h	Funkcje umożliwiające modyfikację programu dla bieżących ustawień regionalnych (krajowych), na których jest uruchomiony.	lconv{}, setlocale(), localeconv()

#include<locale.h>

Pierwszą deklaracją przy wprowadzaniu atrybutów pliku nagłówkowego **locale.h** jest jej funkcja **setlocale()**

- **`setlocale(`***kategoria*, *lokalizacja***`)`** – pobiera i ustawia bieżące parametry regionalne programu

'Kategorię' poddaną zmianom na ustawienia lokalne wraz z podaniem lokalizacji (u nas będzie "Polish") deklaruje się wraz z uruchomieniem funkcji **setlocale()**, np.:

```
setlocale(LC_ALL, "Polish");
setlocale(LC_CTYPE, "Polish");
```

locale.h rozpoznaje sześć **kategorii** danych zawartych w nastepujących makrach:

`LC_ALL` – zmienia wszystkie poniższe kategorie na ustawienia lokalne (danego kraju)

Wszystkie pozostałe kategorie wprowadzają lokalne (krajowe) ustawienia tylko dla swojej kategorii:

`LC_COLLATE` – wpływa na działanie funkcji *strcoll* i *strxfrm* pliku nagłówkowego **string.h**, których to funkcji nie omawiałem
`LC_CTYPE` – powoduje akceptacje znaków narodowych, np. polskich (ą,ę, ł...)
`LC_MONETARY` – rewiduje wartości z dziedziny finansów dając im narodowy charakter przy formatowaniu związanym z funkcją ***localeconv()***
`LC_NUMERIC` – działanie podobne do LC_MONETARY tylko dotyczące liczb, np, w liczbach będzie przecinek dziesiętny zamiast anglosaskiej kropki dziesiętnej
`LC_TIME` – dla funkcji *strftime()* z pliku nagłówkowego **time.h** - tu nazwy angielskie będą zastąpione polskimi, np. 'poniedziałek' zamiast 'Monday'

Środowiskiem pracy funkcji **setlocale()** są struktura **lconv{}** i funkcja **localeconv()**, która jest operacyjnym klonem **lconv{}** .

- **`lconv{}`** – szczegóły formatowania, podawane przez **localeconv()**
- **`localeconv()`** – wykonuje zadania dotyczące szczegółów formatowania liczbowego i monetarnego bieżących ustawień regionalnych/krajowych

```
struct lconv
   {
   char *decimal_point;       // Kropka dziesiętna lub przecinek dziesiętny
   char *thousands_sep;       // Separator grup co tysiąc dla niefinansowych wartości
   char *grouping;            // Liczba cyfr w grupie
   char *int_curr_symbol;     // Trzy znaki międzynarodowego symbolu waluty, np. PLN
   char *currency_symbol;     // Krajowy symbol waluty, np. zł
   char *mon_decimal_point;   // (to samo co: *decimal_point lecz dla finansowych wartości)
   char *mon_thousands_sep;   // (to samo co: *thousands_sep lecz dla finansowych wartości)
   char *mon_grouping;        // (to samo co: *grouping lecz dla finansowych wartości)
```

```
    char *positive_sign;       // Znak wartości dodatnich (+)
    char *negative_sign;       // Znak wartości ujemnych (–)
    char int_frac_digits;      // Ilość cyfr części ułamkowej w zapisie międzynarodowym
    char frac_digits;          // Ilość cyfr części ułamkowej w zapisie krajowym
    char p_cs_precedes;        // 0 – Symbol waluty występuje po liczbie dodatniej
                               // 1 – Symbol waluty występuje przed liczbą dodatnią
    char p_sep_by_space;       // 0 – Nie ma spacji pomiędzy symbolem waluty a liczbą dodatnią
                               // 1 – Spacja oddziela symbol waluty od liczby dodatniej
    char n_cs_precedes;        // 0 – Symbol waluty występuje po liczbie ujemnej
                               // 1 – Symbol waluty występuje przed liczbą ujemną
    char n_sep_by_space;       // 0 – Nie ma spacji pomiędzy symbolem waluty a liczbą ujemną
                               // 1 – Spacja oddziela symbol waluty od liczby ujemnej
    char p_sign_posn;          // Pozycja znaku +
    char n_sign_posn;          // Pozycja znaku –
                               //   0 – Symbol waluty i wartość liczbowa otoczone nawiasami
                               //   1 – Znak przed symbolem waluty i wartością
                               //   2 – Znak po symbolem waluty i wartością
                               //   3 – Znak bezpośrednio przed symbolem waluty
                               //   4 – Znak bezpośrednio po symbolem waluty
    };
```

Zapis deklaracji **localeconv()**

```
struct lconv *localeconv(void);
```

albo

```
struct lconv *lc = localeconv();
```

co jest równoważne zapisowi:

```
struct lconv *lc;
lc = localeconv();
```

Stąd łatwo odwołać się do skróconego zapisu `'lc'`, np.:

```
// Ustawienia lokalne pisma, wartości liczbowych i monetarnych
#include <stdio.h>
#include <locale.h>                // ten temat

int main(void)
{   setlocale(LC_ALL, "Polish");

    struct lconv *lc = localeconv();

    printf("lc->decimal_point = %s\n", lc->decimal_point );
    printf("lc->thousands_sep = |%s|\n", lc->thousands_sep );
    printf("lc->grouping = %d\n", lc->grouping );
    printf("lc->int_curr_symbol = %s\n", lc->int_curr_symbol );
    printf("lc->currency_symbol = %s\n", lc->currency_symbol );
    printf("lc->mon_decimal_point = %s\n", lc->mon_decimal_point );
    printf("lc->mon_thousands_sep = |%s|\n", lc->mon_thousands_sep );
    printf("lc->mon_grouping = %d\n", lc->mon_grouping );
    printf("lc->positive_sign = %s\n", lc->positive_sign );
    printf("lc->negative_sign = %s\n", lc->negative_sign );
```

```
    printf("lc->int_frac_digits = %d \n", lc->int_frac_digits );
    printf("lc->frac_digits = %d \n", lc->frac_digits );
    printf("lc->p_cs_precedes = %d \n", lc->p_cs_precedes );
    printf("lc->p_sep_by_space = %d \n", lc->p_sep_by_space );
    printf("lc->n_cs_precedes = %d \n", lc->n_cs_precedes );
    printf("lc->n_sep_by_space = %d \n", lc->n_sep_by_space );
    printf("lc->p_sign_posn = %d \n", lc->p_sign_posn );
    printf("lc->n_sign_posn = %d \n", lc->n_sign_posn );

    return 0;
}
```

Wynik działania programu:

```
lc->decimal_point = ,
lc->thousands_sep = | |      // <-- tajemniczy znak, inny za każdym razem
lc->grouping = 2057120       // <-- wartość inna wraz z nowym uruchomieniem programu
lc->int_curr_symbol = PLN
lc->currency_symbol = zł
lc->mon_decimal_point = ,
lc->mon_thousands_sep = | |  // <-- tajemniczy znak, inny za każdym razem
lc->mon_grouping = 2039168   // <-- wartość inna wraz z nowym uruchomieniem programu
lc->positive_sign = +
lc->negative_sign = -
lc->int_frac_digits = 2
lc->frac_digits = 2
lc->p_cs_precedes = 0
lc->p_sep_by_space = 1
lc->n_cs_precedes = 0
lc->n_sep_by_space = 1
lc->p_sign_posn = 1
lc->n_sign_posn = 1

--------------------------------
Process exited after 0.1056 seconds with return value 0
Press any key to continue . . . _
```

Wszystkie powyższe zmienne struktury **lconv{}** powinny dać wyniki typu **`char.`** Aby otrzymać sensowne wartości, niektóre pola musiałem zamienić na **`int.`**

Uwagi dotyczące wyświetlenia wartości zmiennej.

- Zalety zastosowania jakiegoś specjalnego znaku jako separatora przy wydruku wartości zmiennej:
  - Wyświetlenie wartości zmiennej może prowadzić w zakłopotanie, gdy rezultatu nie widzimy na ekranie. Przykładem tego w powyższym programie są zmienne 'thousands_sep' i 'mon_thousands_sep'. Stosuję wtedy separator tuż przed wyświetleniem wartości zmiennej i tuż za nim. Znakiem separatora, jaki tu zastosowałem jest '|', znak nad odwrotnym ukośnikiem (*backslash*'em) na klawiaturze . Widzę wtedy, że w wyniku otrzymuję jeden znak spacji w obydwu przypadkach. Ale czy naprawdę jest to spacja? Potrzebna jest lupa Sherlocka Holmesa a coś takiego mamy - patrz: pierwszy kod poniżej.
  - Taki separator obwarujący wartość zmiennej przydaje się, gdy chcemy sprawdzić faktyczną długość zmiennej, np. spodziewamy się wartości |BARTEK| a uzyskujemy |BARTEK |.

- Zalety wyświetlania znaków jako numerów kodu ASCII:
  - Kiedy otrzymujemy dane do programu, może okazać się, że one dziwnie się zachowują. Wtedy wyświetlenie 'podejrzanych' znaków jako ich numerów kodowych (np. kodu ASCII) da nam pewność, że to, co wydaje się nam spacją, faktycznie nią jest a nie jest np. znakiem sterującym 'udającym' spację.

    Może być odwrotnie: znak 'udający' spację nie jest spacją, jak np. w naszym przypadku:

```c
// Odkrywanie faktycznego znaku
#include <stdio.h>
#include <locale.h>                 // ten temat

int main(void)
{   setlocale(LC_ALL, "Polish");
    struct lconv *lc = localeconv();

    // Każde uruchomienie tego programu daje inną wartość 'lc->thousands_sep'
    // Każda wartość dla %s 'udaje spację'
    char *znak = lc->thousands_sep;
    printf("lc->thousands_sep = |%s| \n",   znak );  // udaje spację
    printf("lc->thousands_sep = |%c| \n",   znak );  // może udawać spację
    printf("lc->thousands_sep = |%d| \n",   znak );  // tu na pewno nie jest numer spacji
    printf("lc->thousands_sep = |%x| \n\n", znak );  // tu też nie będzie numer spacji

    char zmienna = ' ';                     // tu wpisana jest prawdziwa spacja
    printf("zmienna = |%c| \n", zmienna );  // czy znak tylko udaje spację?
    printf("zmienna = |%d| \n", zmienna );  // 32 to ASCII dla spacji w systemie dziesiętnym
    printf("zmienna = |%o| \n", zmienna );  // 40 to ASCII dla spacji w systemie ósemkowym
                                            //    bo  4 * 8^1 + 0 * 8^0 = 32
    printf("zmienna = |%x| \n", zmienna );  // 20 to ASCII dla spacji w systemie 16-tkowym
                                            //    bo  2 * 16^1 + 0 * 16^0 = 32

    return 0;
}
```

Przykładowy wynik działania programu:

```
lc->thousands_sep = | |
lc->thousands_sep = |0|
lc->thousands_sep = |12280368|
lc->thousands_sep = |bb6230|

zmienna = | |
zmienna = |32|
zmienna = |40|
zmienna = |20|

--------------------------------
Process exited after 8.199 seconds with return value 0
Press any key to continue . . . _
```

  - Czasem w pliku danych tekstowych może być ukryty znak, na który nie reaguje klawisz strzałki → , tak jakby tam nic nie było. Numer kodu ASCII potwierdzi obecność takiego znaku: wycinek tekstu można poddać pętli 'for' drukując [funkcją printf("%d\n", znak)] każdy znak jako liczbę.

Próba wykorzystania tych ustawień:

```c
// Ustawienia lokalne pisma, wartości liczbowych i monetarnych
#include <stdio.h>
#include <locale.h>                 // ten temat
```

```c
int main()
{   double liczba = 12345678.16789;

//    setlocale (LC_MONETARY, "Polish");
    setlocale (LC_ALL, "Polish");
    struct lconv *lc;    // lub lconv *lc; bo wiadomo, że to struktura
    lc = localeconv();

    printf("liczba = %lf %s\n", liczba, lc->int_curr_symbol);
    printf("liczba = %lf %s\n", liczba, lc->currency_symbol);

    return 0;
}
```

Wynik działania programu:

```
liczba = 12345678,167890 PLN
liczba = 12345678,167890 zł

--------------------------------
Process exited after 1.0754 seconds with return value 0
Press any key to continue . . . _
```

W samej liczbie oprócz przecinka zamiast kropki dziesiętnej, nie uzyskałem spodziewanych wyników! Sugerowane, w pewnej znakomicie opracowanej stronie internetowej, podstawienie `lc->grouping = "\3"` nie działa tak łatwo w **Dev C++**.

Modyfikacja funkcji **`strftime()`** z pliku nagłówkowego **time.h** przy pomocy zalet **locale.h**

```c
// Ustawienia lokalne pisma, wartości liczbowych i monetarnych
#include <stdio.h>
#include <locale.h>                 // ten temat
#include <time.h>                   // dla strftime()
#include <string.h>                 // dla memset() i strlen()

char bufor[100];

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

    setlocale(LC_CTYPE, "Polish");

    printf("Lokalne ustawienia: %s\n", setlocale(LC_TIME, "Polish"));

    memset(bufor,0,strlen(bufor));          // wyczyść bufor
    strftime(bufor,80,"Dzisiaj jest %A, %Y-%m-%d, godzina %X", tp );
    printf("Przykład nr 2 z 'time.h': %s \n", bufor);

    return 0;
```

```
}
```

Przykładowy wynik działania programu:

```
Lokalne ustawienia: Polish_Poland.1250
Przykład nr 2 z 'time.h': Dzisiaj jest wtorek, 2024-05-14, godzina 17:01:51

--------------------------------
Process exited after 10.96 seconds with return value 0
Press any key to continue . . . _
```

Patrz: Plik nagłówkowy '**time.h**', gdzie znajdziesz opis funkcji **`strftime()`** i jej dostępne w **Dev C++** specyfikatory.