Funkcje narzędziowe: nagłówek `<stdlib.h>`

Plik nagłówkowy `<stdlib.h>` zawiera deklaracje funkcji służących do przekształcania liczb, przydzielania pamięci i innych zadań.

```
double atof(const char *s)
```

rownowazna z

```
strtod(s,(char **)NULL)
```

```
int atoi(const char *s)
```

rownowazna z

```
(int) strtol(s, (char **)NULL, 10)
```

```
long atol(const char *s)
```

rownowazna z

```
strtol(s, (char **)NULL, 10)
```

```
double strtod(const char *s, char **endp)
```

Przekształca początkowe znaki s na wartość typu *double* (wiodące znaki białe są ignorowane).

Jeżeli **endp != NULL** to funkcja wstawia wskaźnik do nie przekształconej części tekstu **s**.

Zwraca:

`HUGE_VAL`	dla nadmiaru
0	dla niedomiaru

W obu przypadkach wstawia **ERANGE** do zmiennej **errno**.

```
long strtol(const char *s, char **endp, int base)
```

Działanie podobne jak wyżej, lecz zamiana na **long**. Argument **base** określa sposób transformacji:

od 2 do 36	liczba zapisana przy tej podstawie	
0	decyduje postać liczby zapisanej w s:	
	0 na początku,	to podstawa 8;
	0X lub 0X	podstawa 16
	w pozostałych przypadkach	podstawa 10

Zwraca: `LONG_MAX` lub `LONG_MIN` w przypadku nadmiaru, a **errno** otrzymuje wartość **ERANGE**.

```
unsigned long strtoul
    (const char *s, char **endp, int base)
```

Działa tak samo jak **strtol**, wynik typu **unsigned long**, a w przypadku błędu zwraca `ULONG_MAX`.

Generowanie liczb pseudolosowych

```
int rand(void)
```

Zwraca pseudolosową liczbę całkowitą z przedziału między 0 i `RAND_MAX`, która wynosi co najmniej 32767.

```
void srand(unsigned int seed)
```

Argument **seed** jest zarodkiem dla nowego ciągu liczb pseudolosowych. Początkowy zarodek jest równy 1.

przydział pamięci

```
void *calloc(size_t nobj, size_t size)
```

Zwraca wskaźnik do obszaru pamięci przeznaczonego dla tablicy o **nobj** elementów, każdy o rozmiarze **size**. Obszar jest inicjowany zerami.

```
void *malloc(size_t size)
```

Obszar nie jest inicjowany.

```
void *realloc(void *p, sizee_t size)
```

```
void free(void *p)
```

```
void abort(void)
```

Powoduje nienormalne zakończenie programu, podobnie jak:

```
raise(SIGABRT).
```

```
void exit(status)
```

Powoduje normalne zakończenie programu. Funkcje zarejestrowane przez *atexit* są wywoływane w kolejności odwrotnej do rejestracji, otwarte pliki są aktualizowane, a otwarte strumienie zamykane.

Od implementacji zależy, jak *status* jest przekazywany do środowiska.

```
int atexit(void (*fcn)(void))
```

Rejestruje fukcję fcn, którą należy wykonać przy normalnym zakończeniu programu.

```
int system(const char *)
```

Przykład:

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

int main(void)
{
   printf("Za chwile uruchomimy interpreter polecen"
          " i polecenie systemu!\n");
   system("ls -lt");
   return 0;
}
```

```
char *getenv(const char *name)
```

Zwraca tekst zawarty w zmiennej środowiskowej **name** lub NULL jeśli zmienna nie istnieje.

Przykład:

```
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <string.h>
#include <dos.h>

int main(void)
{
   char *path, *ptr;
   int i = 0;

   /* get the current path environment */
   ptr = getenv("PATH");

   /* set up new path */
   path = (char *) malloc(strlen(ptr)+15);
   strcpy(path,"PATH=");
   strcat(path,ptr);
   strcat(path,";c:\\temp");

   /* replace the current path and
      display current environment */
   putenv(path);
   while (environ[i])
       printf("%s\n",environ[i++]);

   return 0;
}
```

```
void *bsearch(const void *key, const void *base,
   size_t n, size_t size,
   int (*cmp)(const void *keyval, const void *datum))
```

Przegląda uporządkowaną tablicę **base[0] . . . base[n-1]** złożoną z elementów o rozmiarze **size**, w poszukiwaniu elementu odpowiadającemu obiektowi ***key**.

Funkcja wskazywana przez cmp musi zwracać wartość ujemną, gdy pierwszy argument (klucz szukania) jest mniejszy od drugiego (element tablicy), 0 , gdy równy, wartość dodatnią, gdy większy.

Funkcja zwraca wskaźnik do znalezionego elementu, albo NULL.

```
void qsort(void *base, size_t n, size_t size,
           int (*comp) (const void *, const void *))
```

Porządkuje tablicę **base** w porządku rosnącym.

```
int abs(int)

long labs(int)

div_t div(int num, int denom)
```

Oblicza część całkowitą i resztę z dzielenia **num/denom**. Wyniki wstawia do składowych **quot** i **rem** struktury typu `div_t`; składowe są typu **int**.

```
ldiv_t ldiv(long num, long denom)
```

J.w. dla typu long.

Diagnostyka: nagłówek `<assert.h>`

```
void assert(int wyrazenie)
```

Makro służące do diagnozowania programów. Jeśli przy wykonaniu

```
assert(wyrazenie)
```

wartość *wyrazenia* jest równa zeru, to makro **assert** wypisze do **stderr** komunikat postaci:

```
Assertion failed: wyrazenie, file Nazwa_pliku, line nn
```

i wywoła funkcję abort w celu zakończenia programu.

Jeśli przed włączeniem nagłówka `assert.h`do pliku źródłowego zdefiniowano makro **NDEBUG**, to makro **assert** jest pomijane.

Przykład:

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

struct ITEM {
   int key;
   int value;
};

/* add item to list, make sure list is not  null */
void additem(struct ITEM *itemptr) {
   assert(itemptr != NULL);
   /* add item to list */
}

int main(void)
{
   additem(NULL);
   return 0;
}
```

Zmienne listy argumentów: nagłówek `<stdarg.h>`

Skoki odległe: nagłówek `<setjmp.h>`

```
int setjmp(jmp_buf env)

void longjmp(jmp_buf env, int val)
```

Sygnały: nagłówek `<signal.h>`

Mechanizmy obsługi zdarzeń wyjątkowych, np. sygnałów przerwania nadesłanych z zewnątrz programu lub błędu w wykonaniu programu.

```
void (*signal(int sig, void (*handler)(int))(int)
```

Określa, w jaki sposób są obsługiwane nadchodzące sygnały. Jeśli argument **handler** Ma wartość `SIG_DFL`, to działanie domyślne zależne od implementacji, jeśli `SIGIGN`, to sygnał będzie zigonorowany, w pozostałych przypadkach wywołuje się wskazaną fukcję z podanym rodzajem sygnału.

Poniżej możliwe sygnały:

SIGABRT	nienormalne zakończenie programu
SIGFPE	błąd arytmetyczny
SIGILL	zła postać funkcji
SIGINT	ingerencja użytkownika (przerwanie)
SIGSEGV	nielegalne odwołanie do pamięci
SIGTERM	żadanie zakończenia wykonywania

Funkcja **signal** zwraca poprzednią wartość argumentu **handler** dla danego sygnału lub **SIGERR** w przypadku błędu.

Jeśli teraz wystąpi sygnał **sig**, to najpierw przywraca się domyślną akcję dla tego sygnału, a następnie wywołuje wskazaną funkcję obsługi tak, jakby nastąpiło wywołanie **(*handler)(sig)**. Po powrocie z funkcji obsługi program będzie wznowiony w miejscu, w którym nastąpiło przerwanie sygnałem.

```
int raise(int sig)
```

Funkcja **raise** wysyła do programu sygnał **sig**. W razie niepowodzenia zwraca wartość różną od zera.

Obsługa daty i czasu: nagłówek `<time.h>`

Składowe struktury **tm**:

int `tm_sec`;	sekundy, które upłynęły po minucie (0,61)
int `tm_min`;	minuty, które upłynęły po godzinie (0,59)
int `tm_hour`;	godziny, które upłynęły od północy (0,23)
int `tm_mday`;	dzień miesiąca (1,31)
int `tm_mon`;	miesiące, które upłynęły od stycznia (0,11)
int `tm_year`;	lata, ktŕe upłynęły od 1900 r.
int `tm_wday`;	dni, które upłynęły od niedzieli (0,6)
int `tm_yday`;	dni, które upłynęły od pierwszego stycznia (0,365)
int `tm_isdst`;	znacznik letniej zmiany czasu

```
clock_t clock(void)
```

Zwraca czas procesora wykorzystany przez program lub -1, gdy nie może tego wykonać.
Wyrażenie `clock()/CLOCKS_PER_SEC` daje czas w sekundach.

```
time_t time(time_t *tp)
```

Zwraca aktualny czas kalendarzowy lub -1. Jeżeli **tp** jest różne od NULL, to zwracana wartość jest wstawiana również do ***tp**.

```
double difftime(time_t time2, time_t time1)
```

Zwraca różnicę czasów wyrażoną w sekundach.

```
time_t mktime(struct tm *tp)
```

Przekształca czas lokalny zawarty w strukturze ***tp** na czas kalendarzowy w formacie używanym przez **time**.

Następne cztery funkcje zwracają wskaźnik do statycznego obiektu, który może ulec zmianie na skutek innych wywołań tych funkcji.

```
char *asctime(const struct tm *tp)
```

Przekształca czas zapisany w strukturze *tp na tekst:

```
Sun Jan 3 15:14:13 1988\n\0
```

```
char *ctime(const time_t *tp)
```

Przekształca czas kalendarzowy *tp na czas lokalny; równoważne wywołaniu: `asctime(localtime(tp))`.

```
struct tm *gmtime(const time_t *tp)
```

Przekształca czas kalendarzowy *tp na czas południka zerowego.

```
struct tm *localtime(const time_t *tp)
```

Przekształca czas kalendarzowy *tp na czas lokalny.

```
size_t strftime(char *s, size_t *smax,
        const char *fmt, const struct tm *tp)
```

Przekształca datę i czas zawarte w strukturze *tp w tekst i zapisuje w miejsce wskazane przez s. Przekształcenie odbywa się na podstawie formatu **fmt**, analogicznego do formatu **fprintf**.

Przykład:

```
 #include <stdio.h>
 #include <time.h>
#include <dos.h>
int main(void)
 {
    struct tm *time_now;
    time_t secs_now;
    char str[80];
    tzset();
    time(&secs_now);
    time_now = localtime(&secs_now);
    strftime(str, 80,"It is %M minutes after %I"
    " o'clock (%Z)  %A, %B %d 19%y",time_now);
    printf("%s\n",str);
    return 0;
 }
```

Ograniczenia implementacji: nagłówki `<limits.h>` **i** `<float.h>`

Nagłówek `<limits.h>` zawiera definicje stałych określających minimalne rozmiary typów całkowitych, a `<float.h>` minimalne rozmiary typów rzeczywistych.