1. Wstęp. Przedstawiany tu program po wczytaniu tekstu ze standardowego wejścia wypisuje uporządkowaną alfabetycznie listę słów wraz z liczbą ich wystąpień w podanym tekście. Przez *słowo* rozumiemy każdy napis złożony z liter oddzielony od innych napisów znakiem nie będącym literą. Wielkie i małe litery nie będą rozróżniane.

Program `fw` można wykorzystać do wyszukiwania najczęściej występujących słów. Wykonanie poniższego polecenia spowoduje wypisanie na ekranie monitora dziesięciu linii zawierających najczęściej występujące słowa w tekście tego programu:

```
fw < fw.w | sort +0nr | head -n10
```

2. Program czyta tekst, po jednym wierszu, ze *stdin* przy pomocy funkcji *getline*. Funkcja ta nie należy do standardowej biblioteki I/O (wejścia/wyjścia). Należy ona do biblioteki I/O kompilatora GCC. Używam tej funkcji ponieważ umożliwia ona wczytywanie dowolnie długich wierszy tekstu.

Napisy porównywane są przez funkcję *strcoll*. Funkcja ta, porównując napisy stosuje kolejność znaków języka określonego zmienną środowiskową `LANG`. Aby program `fw` zaliczył polskie znaki diakrytyczne do liter i słowa zostały wypisane według porządku alfabetycznego obowiązującego w Polsce, wystarczy przed uruchomieniem tego programu ustawić zmienną `LANG` tak, aby wskazywała język polski:

```
export LANG=pl_PL
```

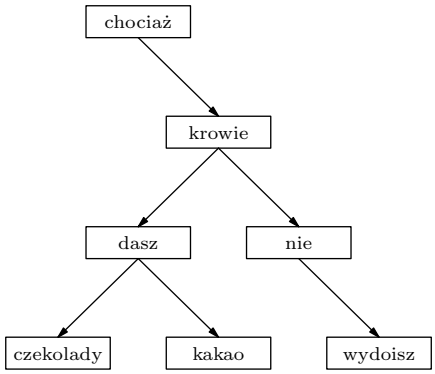
albo uruchamiamy program w taki oto sposób:

```
LANG=pl_PL  cat ⟨nazwa pliku⟩ | fw
```

W programie korzystam z kilkunastu funkcji, których prototypy zawarte są w standardowych plikach nagłówkowych.

```
#define _GNU_SOURCE      /* zdefiniuj getline */
#include <stdio.h>     /* zawiera deklarację funkcji getline */
#include <stdlib.h>     /* malloc, EXIT_SUCCESS */
#include <limits.h>     /* UCHAR_MAX */
#include <locale.h>     /* setlocale – wpływa na działanie poniższych funkcji */
#include <ctype.h>     /* isupper, islower, tolower */
#include <string.h>     /* strcoll, strtok */
```

3. Drzewa binarne. Wczytywane słowa są na bieżąco wstawiane do drzewa binarnego. Zanim powstanie wierzchołek z nowo wczytanego słowa zostanie ono porównane ze słowem uprzednio wstawionym do korzenia. Jeśli nowe słowo jest alfabetycznie mniejsze od porównywanego słowa, to operacja wstawiania zostanie zastosowana rekurencyjnie do lewego poddrzewa. Jeśli jest większe, to do prawego poddrzewa. Jeśli jest równe słowu porównywanemu to licznik jego wystąpień w wczytywanym tekście zostanie zwiększony o 1.



Obok przedstawiono drzewo binarne zdania z książki S. J. Leca, *Myśli nieuczesane*: „Chociaż krowie dasz kakao, nie wydoisz czekolady.”

Uwaga: W języku C nie dopuszcza się, aby struktura była definiowana rekurencyjnie inaczej niż za pomocą wskaźników do siebie samej.

```
struct wierzchołek {
  char *słowo;
  int liczba_wystąpień;
  struct wierzchołek *lewe_poddrzewo;
  struct wierzchołek *prawe_poddrzewo;
};
```

4. ⟨ Zmienne lokalne funkcji main 4 ⟩ ≡

```
struct wierzchołek *korzeń = Λ;
```

Kontynuacja: sekcje 11 i 14.

Ten kod jest użyty w sekcji 10.

5. Utworzenie i inicjalizacja nowego wierzchołka. Jeśli zabraknie pamięci na nowy wierzchołek to program `fw` kończy działanie. Jeśli miejsce jest, to kopiujemy wczytane słowo s z bufora w nowe miejsce w pamięci. Gdybyśmy tego nie zrobili, to następne wywołanie funkcji *getline* umieściłoby w buforze nową linię, zamazując tym samym poprzednią zawartość zawierającą słowo s.

```
struct wierzchołek *nowy_wierzchołek(char *s)
{
  struct wierzchołek *w = malloc(sizeof(struct wierzchołek));
  if (w ≡ Λ) ⟨ Zakończ działanie komunikatem o braku pamięci 9 ⟩
  w→słowo = strdup(s);
  if (w→słowo ≡ Λ) ⟨ Zakończ działanie komunikatem o braku pamięci 9 ⟩
  w→liczba_wystąpień = 1;
  w→lewe_poddrzewo = w→prawe_poddrzewo = Λ;
  return w;
}
```

6. Wstawianie wierzchołków do drzewa. Funkcja *wstaw_wierzchołek* jest rekurencyjna. Jeśli wczytywany ciąg słów jest już częściowo uporządkowany, to budowane drzewo ma mało rozgałęzień i rośnie w dół bardzo nisko (tak rosną drzewa w RAM). Również w takiej sytuacji, drzewo rośnie coraz wolniej, ponieważ funkcja *wstaw_wierzchołek* będzie przeglądać większość wierzchołków drzewa zanim wstawi nowy wierzchołek.

```
struct wierzchołek *wstaw_wierzchołek(struct wierzchołek *w, char *s)
{
  int wynik_porównania;
  if (w ≡ Λ) w = nowy_wierzchołek(s);
  else if ((wynik_porównania = strcoll(s, w→słowo)) ≡ 0) w→liczba_wystąpień++;
  else if (wynik_porównania < 0) w→lewe_poddrzewo = wstaw_wierzchołek(w→lewe_poddrzewo, s);
  else w→prawe_poddrzewo = wstaw_wierzchołek(w→prawe_poddrzewo, s);
  return w;
}
```

7. Drukowanie słów wstawionych do drzewa w porządku alfabetycznym. Funkcja rekurencyjna *inorder_print* wypisze słowa występujące w wierzchołkach w porządku alfabetycznym określonym przez kolejność liter alfabetu języka ustalonego przez zmienną środowiskową `LANG`. Jest tak, ponieważ dla każdego wierzchołka, słowa występujące w jego lewym poddrzewie są mniejsze alfabetycznie od słowa występującego w wierzchołku, a ono samo jest mniejsze od wszystkich słów występujących w jego prawym poddrzewie (zob. rysunek w sekcji 3).

```
void inorder_print(struct wierzchołek *w)
{
  if (w ≠ Λ) {
    inorder_print(w→lewe_poddrzewo);
    printf("%9d␣%s\n", w→liczba_wystąpień, w→słowo);
    inorder_print(w→prawe_poddrzewo);
  }
}
```

8. Przed umieszczeniem słowa w drzewie zamieniamy wielkie litery na małe.

```
char *lowercase(char *s)
{
  char *c = s;
  while (isupper(*c)) {
    *c = tolower(*c);
    c++;
  }
  return s;
}
```

9. Jest możliwe, by ten program zachowywał się sensowniej, niż tylko kończąc działanie w tym miejscu – patrz dokumentacja INFO do biblioteki `libc` węzeł: *Basic Allocation*.

⟨ Zakończ działanie komunikatem o braku pamięci 9 ⟩ ≡

```
{
  fprintf(stderr, "!␣Virtual␣memory␣exhausted.\n");
  exit(1);
}
```

Ten kod jest użyty w sekcji 5.

10. Funkcja główna. Główna część programu to pętla **while**, w której tekst ze *stdin* wczytywany jest przez funkcję *getline* do *bufora_p*. Z wczytanego wiersza kolejne słowa oddzielane są przez *strtok* i wstawiane na bieżąco do drzewa binarnego przez funkcję *wstaw_wierzchołek*. Po wczytaniu ostatniego wiersza ze *stdin*, funkcja *inorder_print* wypisze wszystkie przeczytane słowa wraz z liczbą ich wystąpień.

```
int main( )
{
  ⟨ Zmienne lokalne funkcji main 4 ⟩
  ⟨ Ustal język na podstawie wartości zmiennej środowiskowej LANG 12 ⟩
  ⟨ Utwórz ogranicznik dla funkcji strtok 13 ⟩
  while (getline(&bufor_p, &długość_bufora, stdin) ≠ −1) {
    while ((następne_słowo = strtok(bufor_p, ogranicznik)) ≠ Λ) {
      korzeń = wstaw_wierzchołek(korzeń, lowercase(następne_słowo));
      bufor_p = Λ;      /* dlaczego Λ – patrz strona podręcznika poświęcona strtok */
    }
  }
  inorder_print(korzeń);
  return EXIT_SUCCESS;
}
```

11. Jeśli przed wywołaniem *getline* nadamy zmiennym *bufor_p* i *długość_bufora* wartość 0, to funkcja sama przydzieli sobie pamięć na bufor i adres początku bufora umieści w *bufor_p*. Napotykając koniec pliku *getline* zwraca wartość -1.

```
⟨ Zmienne lokalne funkcji main 4 ⟩ +≡
  char *bufor_p = 0;
  size_t długość_bufora = 0;
```

12. Wywołanie *setlocale* z drugim argumentem będącym pustym napisem spowoduje, że funkcje: *isupper*, *islower*, *tolower* i *strcoll* będą korzystać z danych lokalnych (*locale data*) ustalonych przez zmienną środowiskową LANG.

```
⟨ Ustal język na podstawie wartości zmiennej środowiskowej LANG 12 ⟩ ≡
  setlocale(LC_CTYPE, "");      /* wczytaj kody wielkich i małych liter */
  setlocale(LC_COLLATE, "");      /* wczytaj dane dotyczące porządku alfabetycznego */
```

Ten kod jest użyty w sekcji 10.

13. Kolejne wywołania *strtok* oddzielają od początku bufora *bufor_p* napisy rozdzielone znakami umieszczonymi w tablicy *ogranicznik*. Tablica ta zawiera tylko znaki nie będące znakami alfanumerycznymi.

```
⟨ Utwórz ogranicznik dla funkcji strtok 13 ⟩ ≡
  for (i = 1, k = 0; i ≤ UCHAR_MAX; i++) if (¬isalnum(i)) ogranicznik[k++] = i;
  ogranicznik[k] = '\0';
```

Ten kod jest użyty w sekcji 10.

14. Pozostaje jeszcze tylko zadeklarować kilka zmiennych i program jest gotowy. „To by było na tyle” jak mawia prof. Stanisławski.

```
⟨ Zmienne lokalne funkcji main 4 ⟩ +≡
  unsigned int i, k;      /* liczniki pętli */
  unsigned char ogranicznik[UCHAR_MAX];      /* argument delim w strtok */
  char *następne_słowo;
```

15. Skorowidz. Poniżej znajdziesz listę identyfikatorów użytych w programie `hello.w`. Liczba wskazuje na numer sekcji, w której użyto identyfikatora, a liczba podkreślona — numer sekcji w której zdefiniowano identyfikator.

⟨ Ustal język na podstawie wartości zmiennej środowiskowej `LANG` 12 ⟩ Użyto w sekcji 10.
⟨ Utwórz *ogranicznik* dla funkcji *strtok* 13 ⟩ Użyto w sekcji 10.
⟨ Zakończ działanie komunikatem o braku pamięci 9 ⟩ Użyto w sekcji 5.
⟨ Zmienne lokalne funkcji main 4, 11, 14 ⟩ Użyto w sekcji 10.

ZLICZANIE SŁÓW

(wersja 1.1)

Jest to wersja programu autorstwa R. Haighta z książki B. W. Kernighana, D. M. Ritchiego, *Język ANSI C*. Program ten korzysta z *locales*. Oznacza to, że słowa zostaną wypisane w porządku alfabetycznym właściwym dla języka ustalonego przez zmienną środowiskową `LANG`.