

Podstawy Programowania

Wykład XI

Przetwarzanie napisów, drzewa binarne

Robert Muszyński

Katedra Cybernetyki i Robotyki, PWr

Zagadnienia: reprezentacja napisów znakowych, operowanie na napisach: porównywanie, kopiowanie, łączenie, wyszukiwanie, konwersja; częste błędy, drzewa binarne.

Copyright © 2007–2021 Robert Muszyński

Niniejszy dokument zawiera materiały do wykładu na temat podstaw programowania w językach wysokiego poziomu. Jest on udostępniony pod warunkiem wykorzystania wyłącznie do własnych, prywatnych potrzeb i może być kopiowany wyłącznie w całości, razem ze stroną tytułową.

– Skład Foil \TeX –

Reprezentacja napisów znakowych

- Tablice niepełne

```
#define MAXSTRING 20
typedef char String1[MAXSTRING];
String1 Napis1;
int Dlug_Napis1;
```

```
typedef struct {
    char Znaki[MAXSTRING];
    int Dlugosc;
} String2;
```

- Tablice ze znacznikiem (np. \0)

```
#define MAXSTRING 20
typedef char String1[MAXSTRING];
String1 Napis1 = "Ala ma kota"; /* to nie to samo co */
String1 Napis2 = {'A','l','a',' ','m','a',' ','k','o','t','a'};
```

Napisy w języku C

OSTRZEŻENIE!

Napisy w języku C mogą być przyczyną wielu trudnych do wykrycia błędów w programach. Warto dobrze zrozumieć, jak należy operować na łańcuchach znaków i zachować szczególną ostrożność w tych miejscach, gdzie napisów używamy.

```
printf("Napis w jezyku C");
```

```
char *tekst = "Jakis tam tekst";
printf("%c\n", tekst[2]); /* wypisze k */
printf("%c\n", "przyklad"[0]); /* wypisze p */
printf("%d", "test"[4]); /* wypisze 0 */
```

Na napisach operują standardowe funkcje `printf`, `scanf` z nagłówka `stdio.h`:

```
printf("%s", tekst);
```

Ale większość z nich znajduje się w pliku nagłówkowym `string.h`.

Napisy w języku C cd.

```
char *tekst = "Jakis tam tekst"; /* Napis w obszarze danych programu */
char tekst[] = "Jakis tam tekst"; /* Napis w tablicy o automatycznie */
/* dobranym rozmiarze */
char tekst[80] = "Tekst krotszy niz 80 znakow"; /* Tablica 80-znakowa */
```

```
printf("Ten napis zajmuje \
wiecej niz jedna linie");

printf("Ten napis\nna wyjsciunzajmie wiecej niz jedna linie.");
```

UWAGA!

Warto zaznaczyć, że znak nowej linii (`'\n'`) jest w różny sposób przechowywany w różnych systemach operacyjnych. W niektórych systemach używa się do tego jednego znaku (systemy z rodziny Unix: Linux, *BSD, Mac OS, Commodore, Apple II); drugą konwencją jest zapisywanie `'\n'` za pomocą dwóch znaków (CP/M, DOS, OS/2, Microsoft Windows).

Porównywanie napisów

Tak się nie da:

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

int main(void) {
    char str1[80], str2[80];

    puts("Podaj dwa ciagi znakow: ");
    fgets(str1, sizeof str1, stdin);
    fgets(str2, sizeof str2, stdin);

    if (str1<str2) {
        puts("Pierwszy napis jest mniejszy.");
    } else if (str1>str2) {
        puts("Pierwszy napis jest wiekszy.");
    } else {
        puts("Napisy sa takie same.");
    }
    return 0;
}
```

Trzeba tak:

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

int main(void) {
    char str1[80], str2[80];
    int cmp;
    puts("Podaj dwa ciagi znakow: ");
    fgets(str1, sizeof str1, stdin);
    fgets(str2, sizeof str2, stdin);

    cmp = strcmp(str1, str2);

    if (cmp<0) {
        puts("Pierwszy napis jest mniejszy.");
    } else if (cmp>0) {
        puts("Pierwszy napis jest wiekszy.");
    } else {
        puts("Napisy sa takie same.");
    }
    return 0;
}
```

Wczytywanie napisów – komentarz

- operacje na znakach (to już znamy :)

```
int fgetc(FILE *stream); /* to jest na pewno funkcja */
int getc(FILE *stream); /* moze byc zaimplementowane jako makro */
/* wiec dziala raczej szybciej, ale moze wyliczac argument wielokrotnie */
int getchar(void); /* rownowazna getc(stdin) */

int ungetc(int c, FILE *stream); /* gwarantowane tylko jednokrotnie! */
/****** wiecej: man getc *****/
```

- operacje na napisach

```
char *fgets(char *s, int size, FILE *stream);
char *gets(char *s); /* ze standardowego wejscia */
/* usunieta w standardzie C11!!! */
char *gets_s( char *str, rsize_t n ); /* od normy C11 */
/****** wiecej: man gets *****/
```

Biblioteka string

```
#include <string.h>

char *strcpy(char *dst, const char *src);
char *strncpy(char *dst, const char *src, size_t n);
char *strdup(const char *s1);
size_t strlen(const char *s);
char *strcat(char *dst, const char *src);
char *strncat(char *dst, const char *src, size_t n);
char *strchr(const char *s, int c);
char *strrchr(const char *s, int c);
int strcmp(const char *s1, const char *s2);
int strncmp(const char *s1, const char *s2, size_t n);
int strcasecmp(const char *s1, const char *s2);
int strncasecmp(const char *s1, const char *s2, int n);
size_t strcspn(const char *s1, const char *s2);
size_t strspn(const char *s1, const char *s2);
char *strpbrk(const char *s1, const char *s2);
char *strtok(char *s1, const char *s2);
char *strstr(const char *s1, const char *s2);
```

Inne operacje na napisach

- porównanie fragmentu napisów

```
if (!strncmp(str, "foo", 3)) {
    puts("Podany ciąg zaczyna sie od 'foo'.");
}
```

- kopiowanie napisów

```
strcpy(napis, "Oto pies Oli."); /* napis musi byc wystarczajaco duzy */
```

```
strncpy(napis, "Oto pies Oli.", sizeof napis - 1);
napis[sizeof napis - 1] = 0; /* teraz juz nie */
```

```
/* kopiowanie znak po znaku */
/* co z rozmiarem? */
for (i = 0; i < 80; ++i)
    s2[i] = s1[i];
```

```
/* to samo przy uzyciu wskaznikow */
s3 = s1; s4 = s2;
for (i = 0; i < 80; ++i, ++s3, ++s4)
    *s4 = *s3;
```

Inne operacje na napisach cd.

- kopiowanie napisów cd.

```
char s1[20], s2[20] = "Ala ma kota.";
char *s3, *s4;
s1 = s2; /* niedozwolone */
strcpy(s1, s2); /* tak mozna, funkcja biblioteczna */
s3 = s1; /* tez dozwolone */
strcpy(s4, s3); /* zle, s4 nie jest tablica */
s4 = s2; /* oczywiscie */
strcpy(s4, s3); /* teraz dobrze, kopiuja sie s2 do s1 */
```

- łączenie napisów

```
char napis1[80] = "Witaj, ";
char *napis2 = "Swiecie";
strcat(napis1, napis2);          /* znow musimy dbac o wymiar */
```

```
/* moze wiec podobnie jak poprzednio */
strncat(napis1, napis2, sizeof napis1 - 1); /* co ze znakiem \0? */
```

Wyszukiwanie wzorców

Zadanie: Napisać program do wyświetlania tych linii z `stdin`, które zawierają określony napis znakowy.

Schemat:

```
while ( jest jeszcze jedna linia danych )
  if ( linia zawiera zadany napis znakowy )
    wyswietl linie
```

```
/* getline: get line into s, return length */
int getline(char s[], int lim) {
  int c, i = 0;

  while (--lim > 0 && (c=getchar()) != EOF && c != '\n')
    s[i++] = c;
  if (c == '\n')
    s[i++] = c;
  s[i] = '\0';
  return i;
}
```

```
/* strindex: return index of t in s, -1 if none */
int strindex(char s[], char t[]) {
    int i, j, k;
    for (i = 0; s[i] != '\0'; i++) {
        for (j=i, k=0; t[k]!='\0' && s[j]==t[k]; j++, k++)
            ;
        if (k > 0 && t[k] == '\0')
            return i;
    }
    return -1;
}
```

Kompletujemy rozwiązanie naszego przykładowego problemu:

```
#include <stdio.h>
#define MAXLINE 1000 /* max dlugosc linii wejsciowej */
int getline(char line[], int max);
int strindex(char source[], char searchfor[]);

char pattern[] = "ould"; /* wzorzec do znalezienia */

/* wyszukaj wszystkie linie pasujace do wzorca */
main()
{
    char line[MAXLINE];
    int found = 0;
    while (getline(line, MAXLINE) > 0)
        if (strindex(line, pattern) >= 0) {
            printf("%s", line);
            found++;
        }
    return found;
}
```

Konwersje

Funkcje biblioteczne służące do konwersji napisów:

- `atoi` – zamienia łańcuch na liczbę całkowitą typu `int`,
- `atol`, `strtol` – zamienia łańcuch na liczbę całkowitą typu `long`,
- `atoll`, `strtoll` – zamienia łańcuch na liczbę całkowitą typu `long long`,
- `atof`, `strtod` – przekształca łańcuch na liczbę typu `double`

Czasami przydaje się też konwersja w drugą stronę, tzn. z liczby na łańcuch. Do tego celu może posłużyć funkcja `sprintf` lub `snprintf`. `sprintf` jest bardzo podobna do `printf`, tyle, że wyniki jej pracy zwracane są do wskazanego łańcucha, a nie wyświetlane na standardowym wyjściu.

Operacje na znakach – przykład konwersji

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>

int main()          /* zamiana male <-> duze */
{
    int znak;
    while ((znak = getchar())!=EOF) {
        if( islower(znak) ) {
            znak = toupper(znak);
        } else if( isupper(znak) ) {
            znak = tolower(znak);
        }
        putchar(znak);
    }
    return 0;
}
```

Częste błędy

- pisanie do niezaalokowanego miejsca

```
char *tekst;  
scanf("%s", tekst);
```

- zapominanie o kończącym napis „nullu”

```
char test[4] = "test"; /* nie zmiescil sie NULL konczacy napis */
```

- nieprawidłowe porównywanie łańcuchów

```
char tekst1[] = "jakis tekst";  
char tekst2[] = "jakis tekst";  
if( tekst1 == tekst2 ) { /* tu zawsze bedzie fałsz */  
    ...  
}
```

Drzewa binarne

drzewo – graf reprezentujący regularną strukturę wskaźnikową, gdzie każdy element zawiera dwa lub więcej wskaźników (ponumerowanych) do takich samych elementów; węzły (albo wierzchołki) grafu reprezentują elementy pamięciowe, a łuki reprezentują wskaźniki

drzewo binarne – każdy element zawiera dokładnie dwa wskaźniki

korzeń – element drzewa, od którego zaczynają się wskaźniki do pozostałych elementów

liście – elementy drzewa, których oba wskaźniki są puste (NULL)

wewnętrzne węzły drzewa – węzły, które nie są ani liśćmi ani korzeniem

ścieżka – ciąg węzłów biegnący zgodnie ze wskaźnikami od korzenia do jakiegoś liścia

poddzewo – drzewo, które jest całkowicie zawarte w innym drzewie

rzęd drzewa – liczba wskaźników (≥ 2) w jednym węźle drzewa (niektóre z nich są NULL)

wysokość drzewa – długość najdłuższej ścieżki drzewa

waga drzewa – całkowita liczba węzłów w drzewie

Podsumowanie

• Zagadnienia podstawowe

1. Wymień sposoby reprezentowania napisów w pamięci komputera.
2. Jakie są wady reprezentacji napisu w postaci ze znacznikiem końca?
3. Jakie są potencjalne źródła błędów przy korzystaniu z napisów w języku C?
4. Jakich funkcji można użyć do porównywania napisów w języku C? Czym one się różnią?
5. Co robi funkcja `strstr`?
6. W jaki sposób można przekształcić napis reprezentujący wartość liczbową w tę wartość i vice versa?
7. Czym jest drzewo binarne i jakie elementy można w nim wyróżnić?
8. Czym różni się drzewo od listy?
9. Czy każda ścieżka w drzewie jest listą?
10. Czy wysokość drzewa może być większa od jego wagi?

• Zagadnienia rozszerzające

1. Czym różni się zamiana ciągu znaków na liczbę przy pomocy funkcji `atoi()` od `sscanf()`?
2. Jaka jest różnica w poziomie bezpieczeństwa pomiędzy funkcją `strncpy` a `strncpy`?
3. Jaka jest rola modyfikatora `const` w funkcji
`char *strstr(const char *s1, const char *s2)?`

4. Na czym polega problem z kodowaniem narodowych znaków diakrytycznych w napisach? Jakie są popularne sposoby ich kodowania? Do czego służy polecenie `iconv`?
5. Jakie są zastosowania drzew binarnych?
6. Jakie inne rodzaje drzew (oprócz binarnych) stosuje się w programach?
7. Czym są drzewa zrównoważone? Podaj przykładowe ich rodzaje i zastosowania.

• Zadania

1. Napisz funkcję zwracającą indeks ostatniego znaku w napisie.
2. Napisz program, który w podanym na jego wejście tekście znajdzie wszystkie palindromy.
3. Napisz program, który dla podanego na jego wejściu tekstu zwróci liczbę występujących w nim słów o kolejnych długościach.
4. Napisz program, który w podanym pliku będzie wyszukiwał linie zawierające zadane słowo. Dodatkowo można uwzględnić następujące opcje: tylko zliczanie linii (bez wyświetlania), linie nie zawierające słowa, brak rozróżniania wielkich i małych liter. Zwróć uwagę na zachowanie programu przy różnych ustawieniach zmiennych lokalizacyjnych – w jakim przypadku można zaobserwować różnicę?
5. Zdefiniuj strukturę danych i podstawowe operacje potrzebne do stworzenia programu słownika wykorzystującego drzewa binarne.