

Лекция 4.

ПОСЛЕДОВАТЕЛЬНАЯ ОБРАБОТКА ДАННЫХ



Последовательная обработка данных

- **Задача 4.1.** Задана непустая числовая последовательность. Определить, является ли последовательность знакопередающей.
- Тест1. **Вход:** -5 3.1 -2 1 -7
Выход: Знаки чередуются.
- Тест2. **Вход:** -5 3.1 2 -1 7
Выход: Знаки не чередуются.
- **Используем алгоритм 3.3.** для обработки последовательности.



Последовательная обработка данных

Одного текущего элемента последовательности недостаточно, будем сохранять два соседних элемента:

xpred – предыдущий элемент последовательности,

x – текущий элемент последовательности.

При обработке последовательности необходимо **искать нарушение знаочередования.**

Используем дополнительную переменную

flag – **признак знаочередования.**

flag = 1 - знаки чередуются, **0** – нет,

инициализация: **flag = 1.**



Последовательная обработка данных

```
/* Программа 4.1. Знакопереживание последовательности чисел */  
#include <stdio.h>  
void main(void)  
{ float  xpred, x;          /* предыдущее и текущее числа      */  
  int  flag = 1;          /* flag = 1 - знаки чередуются, 0 - нет */  
  scanf("%f", &xpred) ;  
  while( scanf("%f", &x)>0)  
  {  if (xpred * x >= 0) flag = 0;  
    xpred = x;  
  }  
  if (flag) printf ("\n Знаки чередуются. ");  
  else printf ("\n Знаки не чередуются. ");  
}
```



Введение в программирование

ПОСЛЕДОВАТЕЛЬНАЯ ОБРАБОТКА СИМВОЛОВ



Последовательная обработка символов

Значением символьного типа является **одиночный символ**.

В языке С символьные данные рассматриваются как **разновидность целых чисел**. Числовым значением символа является его код.

В языке С над символами **разрешаются** не только операции присваивания и сравнения, но и **арифметические операции**.

В международном стандарте **ASCII** код символа обычно занимает **один байт**, но иногда и два байта (в международном коде UNICODE).



Последовательная обработка символов

- Примеры символьных констант:

'*' 'a' '5' 'n'

- Специальные (управляющие) символьные константы:

'\n' Новая строка (new line),

'\t' '\v' Табуляция горизонтальная,
вертикальная,

'\b' Возврат на шаг (backspace),

'\\' - \ (обратный слэш), '\'' - ' (апостроф), '\"' - " (кавычка),

'\0' Нулевой символ (байт с нулевым кодом).

- Объявление символьных переменных

char <имя> [,<имя>]...; или **int** <имя> [,<имя>]...;

Например:

```
char s, sim = 'Z', c;
```



Последовательная обработка символов

- Кодировка цифровых символов:

$$'0' = 48$$

$$'1' = '0' + 1 = 49$$

$$'2' = '0' + 2 = 50$$

$$'9' = '0' + 9 = 57$$

- Отсюда соотношения:

Код цифры = '0' + Значение цифры

Значение цифры = Код цифры - '0'

Условие

"значение символьной переменной s является цифрой"

на языке C запишется так:

`s >= '0' && s <= '9'`



Последовательная обработка символов

- Коды заглавных латинских букв возрастают по алфавиту:

'A' < 'B' < ... < 'Z' ,

- Коды строчных латинских букв возрастают:

'a' < 'b' < ... < 'z'.

Условие

"значение символьной переменной s является латинской буквой"

можно записать так:

`(s>='A' && s<='Z') || (s>='a' && s<='z')`



Последовательная обработка символов

char s; (или int s;)

- **Ввод символа** из *стандартного входного файла* (клавиатуры) в переменную s:

```
scanf ("%c", &s);
```

можно заменить присваиванием

```
s = getchar ();
```

Функция `getchar()` вводит очередной символ из стандартного входного файла и возвращает в виде значения код этого символа.



Последовательная обработка символов

Ввод символа часто пишется внутри условия в операторах **if**, **while**, **do-while** и **for**.

- Например, цикл ввода символов до конца файла

```
Ввод s;  
while(s!= конец файла)  
{  Обработка s;  
    Ввод s;  
}
```

МОЖЕТ ИМЕТЬ ВИД

```
while ((s=getchar()) != EOF)  
    Обработка s;
```



Последовательная обработка символов

- Символическая константа EOF - код конца файла (после нажатия клавиш Ctrl-Z или Ctrl-z, затем Enter).
- **Вывод символа** *s* в *стандартный выходной файл* (на экран)

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

эквивалентен оператору

```
putchar (s);
```

- Стандартный входной и выходной файлы, вместо клавиатуры и экрана, можно переадресовать на любой файл магнитного диска.



- **Задача 4.2.** Вывести коды введенных с клавиатуры символов. Последовательность завершается нажатием клавиши Esc.
- **Тест.** Вход: Kazan 2007 <Esc>

Выход: K=82, a=97, z=122, a=97, n= 110, =32, 2=50, 0=48, 0=48, 7=55, □ = 27



```
• /* Программа 4.2. Коды символов */
#include <stdio.h>
#include <conio.h>

main()
{ int sim;
  printf("\n Введи текст, завершив клавишей Esc\n");
  do
  { sim=getch(); putchar(sim);
    printf("=%d, ",sim);
  }
  while( sim!=27);
  puts(" Нажми любую клавишу"); getch();
  return 0;
}
```



Последовательная обработка символов

Подсчет строк, слов и символов

- **Задача 4.3.** Составить программу подсчета во входном тексте количества строк, слов и символов.

Словом считается любая последовательность символов, не содержащая пробелов, символов табуляции и новой строки. Строка заканчивается символом новой строки.

- **Тест.** **Вход:**
Если друг оказался вдруг
И не друг, и не враг, а так.

Выход:
Строк: 2, слов: 12, символов: 53.

Количество символов считаем сразу после ввода, количество строк – по количеству символов '\n', а количество слов – при вводе символа не разделителя, перед которым был разделитель. Используем для этого флаг разделителя.



Пояснения к программе

- **switch** (переключить) - оператор переключателя для организации **многовариантного ветвления**.

switch (выражение)

```
{ [ case цел-конст-выраж: [оператор...]]...  
  [ default: оператор...]  
  [ case цел-конст-выраж: [оператор...]]...  
}
```

- **case** (случай) - **вариант ветвления**, можно пометить целой или символьной константой (или константным выражением).

Вычисляется значение выражения. Переключатель осуществляет переход к одному из вариантов **case**, константное выражение которого совпадает с вычисленным значением, или к метке **default** (умолчание), если не совпадает ни с одной из констант вариантов ветвления.

Каждый вариант обычно заканчивается оператором **break**.



Последовательная обработка символов

```
/* Программа 4.3. Подсчет строк, слов и символов */
/* текст ::= символ... */
/* символ ::= разделитель | символ-слова */
/* разделитель ::= пробел | новая-строка | табуляция */
/* | конец-файла */
/* символ-слова - любой символ, кроме разделителей */
#include <stdio.h>
#define DA 1
#define NET 0
void main ()
{ int sim; /* Текущий символ (int для EOF) */
  int kstr, ksl, ksim; /* Кол-во строк, слов и символов */
  int razdel; /* Флаг символа - разделитель */
  razdel = DA; /* 1-й символ текста - начальный */
```



Подсчет строк, слов и символов

```
kstr = ksl = ksim = 0;
while ((sim = getchar()) != EOF)
{
    ksim++;
    switch (sim)
    {
    case '\n':
        kstr++;
        razdel = DA; break;      /* Эту строчку можно убрать! */
    case ' ':
    case '\t':
        razdel = DA; break;
    default:
        /* Символ слова не разделитель */
        if (razdel)             /* Предыдущий символ - разделитель */
            { ksl++; razdel = NET; }
    }
}
printf ("Строк: %d, слов: %d, символов %d. \n", kstr, ksl, ksim);
}
```

