Информатика (основы программирования)

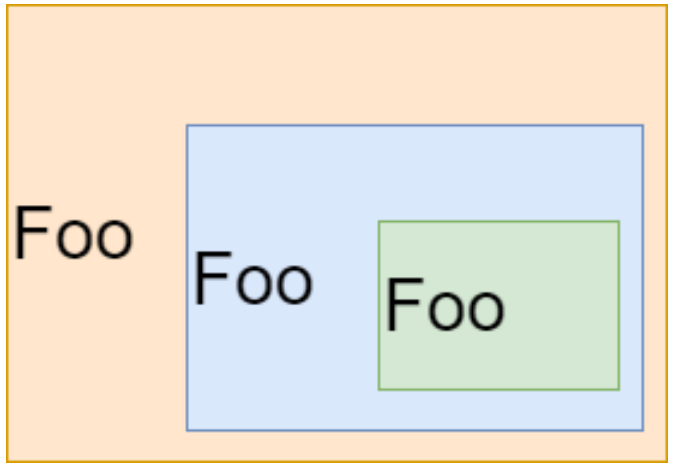
Лекция 8

Алгоритмизация обработки строк в языке С

Автор: Бабалова И.Ф.
Доцент, каф.12

Рекурсивные функции (1)

- Объект называется рекурсивным, если он содержит сам себя или определен с помощью самого себя.
- **Функция** в языке программирования C может быть рекурсивной, т.е. может вызывать сама себя. Каждый раз, когда функция вызывает себя, на стеке выделяется новый кадр вызова (локальные переменные, промежуточные результаты и т.д.).
- Функция должна вызывать себя условно, например, внутри оператора `if`, иначе бесконечная рекурсия приведёт к переполнению стека.
- Рекурсивный вызов функции может привести к потере инварианта цикла.



Рекурсивные функции (2)

- Рекурсии в математике:

1. а) 0! б) n!=n*(n-1)!
2. а) 1 – натуральное число
   б) целое число,

следующее за натуральным, есть натуральное число

```
// Реализация на языке
// Выделить цифры целого числа
void printd (int n) {
  int i;
  if (n < 0) {
      putchar ('-');
      n = -n;
  }
  if ((i = n/10) != 0) {
      printd(i);
  }
  putchar(n % 10 + '0');
}
```

Рекурсивная функция бинарного поиска

```
int binarySearch(int val, int a[], int left, int right) {
    if (left >= right) {                    // конец поиска: число найдено (a[left])
        return (a[left] == val)? left : -1;// или нет искомого числа
    }
    int mid = (left + right) / 2; //поиск среднего ключа
    if (a[mid] == val) {
        return mid;
    }
    if (a[mid] > val) { // рекурсивный вызов функции поиска
        return binarySearch(val, a, left, mid-1);
    }
    else {
        return binarySearch(val, a, mid+1, right);
    }
}
```

Константы в языке С (1)

- **Константа** — определенная в программе и неизменяемая величина.

В программе константа может использоваться как поименованная или как литерал.

Поименованная константа – константа, обращение к которой выполняется по имени. Она описывается в разделе **объявлений переменных** и поименованных констант.

Для объявления поименованной константы используется **ключевое слово** `const`.

Значение константы или литерал представляет собой значение , записанное непосредственно в тексте программы.

Константы в языке С (2)

- Константы делятся на пять групп:
  1. Целые константы
  2. Вещественные
  3. Константы перечисления
  4. Символьные константы
  5. Строковые константы
- **Компилятор**, выделив константу, относит ее к тому или другому **типу данных** по "внешнему виду" (по форме записи) в исходном тексте и по числовому значению.
- **Константу нельзя изменить в теле программы, где она была объявлена**

Примеры записи констант (1)

- Целочисленные константы:

```
const a = 200;  // int
const a = 200L; // long int
const a = 200U; // unsigned int
```

- Вещественные константы:

```
const a  = 200.56;  // float
const a  = 200.78L; // long double
const w  = 2.1E5;
const zz = .123E3;
const mm = 4037e-5;
const pi = 3.14159F; // константа типа float, занимающая 4
байта
const vv = .14L // константа типа long double, занимающая 10
байт
```

Примеры записи констант (2)

- Константы перечисления:

```
enum boolean {NO, YES};
// Булевского типа данных в языке в явном виде нет

enum day {SUN, MON, TUES, FRI=5, SAT};
// Константы индексируются, начиная с нуля или с указанного значения: SUN=0, MON=1, TUES=2, FRI=5, SAT=6.
```

Символьная константа (1)

- **Символьная константа** – символ, заключенный в одиночные кавыки (апострофы). Для обычных символов в апострофах указывается сам символ (например, `'a', 'B', '+', '5'`).
- Все символьные константы имеют соответствующее им числовое значение согласно используемой таблице кодирования символов (ASCII, КОИ-8 и др.) и занимают 1 байт памяти.
- Символьные константы – это целые числа типа **`char`**.
- Они могут участвовать в операциях над числами точно так же, как и другие целые, хотя чаще они используются для сравнения с другими символами.
- Будьте внимательны и помните, что символьная константа и строка, содержащая один символ – не одно и то же: символ `'x'` не то же самое, что `"x"`.
- Запись `'x'` обозначает целое значение, равное коду буквы x из стандартного символьного набора, а запись `"x"` – массив символов, который содержит один символ (букву x) и `'\0'`.

Символьная константа (2)

- Символьная константа ‘\0’ это символ с нулевым значением, так называемый символ NULL. Вместо простой записи 0 часто используют ‘\0’ чтобы подчеркнуть символьную природу выражения, хотя и в другом случае запись обозначает нуль.

```
const char = ‘z’; // В памяти будет записан код буквы 01011010₂ или 5A₁₆
const char msg[] = “Предупреждение: ”; // это константа будет строкой символов с нулём в конце строки
```

Строковая константа

- Строковая константа определяется как набор символов, который заключен в двойные кавычки и неявно содержит завершающий символ с нулевым кодом (‘\0’).
- Следующий пример иллюстрируют два способа определения символьной строки в форме строковой константы.

```
#define OS “Linux" // Макроопределение строковой константы
char* os = “Linux”;// Указатель на строковую константу

#define A 280U  // unsigned int
#define M 10000 // задаём размер
```

Символьная строка

- **Символьная строка** — основная структура данных в программах обработки текстовой информации.
- В терминах языка программирования С под строкой символов понимается любая последовательность символов, которую завершает нулевой код `'\0'`.
- В прикладной программе символьная строка может быть задана как строковая константа или как одномерный массив символов.
- Строковая константа определяется как набор символов, который заключен в двойные кавычки и неявно содержит завершающий символ с нулевым кодом.
- Если символьная строка задается в форме одномерного массива, то все его элементы должны иметь тип char или (unsigned char), а последний символ должен иметь нулевой код.

```
char buf[81] = {0}; // Возможное объявление строки
```

Алгоритмы работы со строками (1)

```
char* str[] = “объявление строки ”;
```

- Алгоритмы для работы со строками нацелены на поиск заданного образца P в тексте T. Формально задача называется задачей поиска подстрок. Алгоритм должен определить позицию j подстроки P в тексте T.
- Позиция может быть допустимой, если:

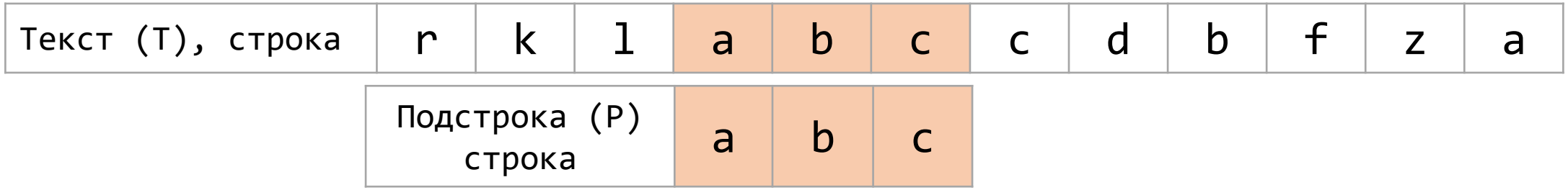
$$0 \le j \le n\text{-}m \text{ и } T[j+1..j+m] = P[1..m],$$

где n - длина строки, а m –длина подстроки.

- Остальные задачи типа упорядочивания по алфавиту, инверсии строки, преобразования отдельных слов в строке – все основаны на поиске элемента строки или подстроки.

Алгоритмы работы со строками (2)

- Задача. Найти подстроку в строке. Поиск подстроки сводится к определению позиции s подстроки P в тексте T.



- Длина строки n (12), длина подстроки m (3)
- Время поиска подстроки при просмотре всей строки оценивается формулой:

$$t \approx \theta((n - m + 1) * m)$$

- Основное требование для реализации цикла – длина строки и длина подстроки должны контролироваться дополнительно.

Алгоритмы для поиска подстрок (1)

1. **Рабина-Карпа** с $t \approx O(n) + O(m(\nu + n / q))$, где ν – количество допустимых сдвигов, q – число, позволяющее считать сравниваемые коды по модулю этого числа. Время, затрачиваемое на подготовку последовательности к поиску подстрок, уменьшается.

2. **Конечный автомат –** $\Theta(n)$. Из строки строится конечный автомат всех его состояний и дополняется записью состояний искомой подстроки. Анализ совпадений позволит выделить совпадение заданной подстроки с символами строки и количество этих совпадений.

3. **Кнута – Морриса- Пратта** $\Theta(n)$. Алгоритм анализирует не строку, а подстроку, выявляя необходимое количество сдвигов для просмотра строки.

Алгоритмы для поиска подстрок (2)

- Во всех этих алгоритмах предполагается предварительная подготовка данных, которая не будет более O(m), что меньше O(n).

- Сложные абстрактные структуры данных типа деревьев, таблиц позволяют сократить времена поиска подстрок, но всегда требуют предварительной подготовки входных данных.

Литература: Т. Кормен и др., Д. Кнут

Алгоритмы для преобразования строк

- Алгоритмы поиска символов в строке
- Алгоритмы кодирования строки
- Алгоритмы декодирования строки
- Инверсия строки на месте исходной строки
- Конкатенация строк
- Нахождение наибольшей общей последовательности символов в двух строках
- Удаление символа из строки.
- Добавление символа в строку
- Сортировка строк по требуемым признакам

Стандартные приемы обработки строк

- Большинство программ, обрабатывающих строки, используют последовательный просмотр символ за символом – посимвольный просмотр строки. Если же в процессе предполагается изменение её содержимого, то возможны два варианта:

1. редактировать строку «на месте», реализуя вставку и удаление символов или ее фрагментов
2. Организовать посимвольное приписывание входной строки в выходную с копированием нужных или преобразованных фрагментов.

Пример

Задача: удалить в строке все лишние пробелы.

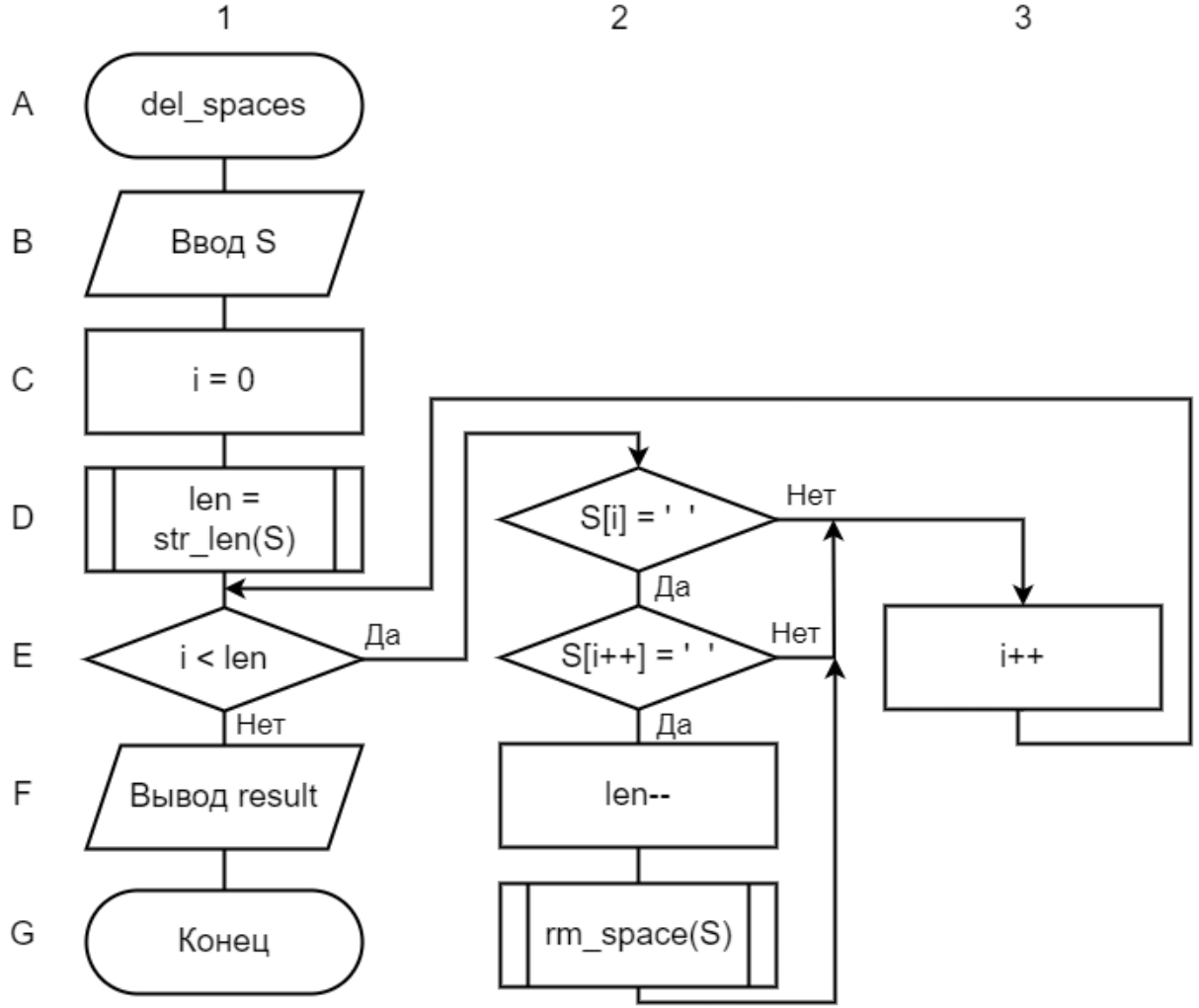
- Поиск пробелов в строке сводится к определению позиции пробела в строке s. Искомый символ можно интерпретировать как подстроку. В этом случае можно считать эту задачу, как самую распространённую для строк задачу: поиск подстроки в заданной строке.
- Алгоритмическая сложность решения подобной задачи оценивается величиной $t \approx \theta((n-m+1)*m)$, где n – это длина строки, а m – длина подстроки.

В этом алгоритме необходимые действия со строкой:

1. Ввод строки
2. Определение длины строки
3. Удаление лишних пробелов.
4. Вывод результата обработки

- Эти действия опишем собственными функциями. В основной программе main будут только вызовы этих функций и организация многократного повторения решения задачи для объёмного тестирования программы с чтением данных и соответствующим выводом.

Алгоритм удаления лишних пробелов в строке



Две встроенные функции:
Str_len и rm_space
введены для решения задачи

Это только удаление лишних пробелов.

Строка уже введена и определена её длина

Применение функций при работе со строками

Задача. Закодировать строку текста

```
#include <stdio.h>
  int lens(const char *str)
{ int i=0;
    while (str[i])
    {  i++;
    }
    return i;}
  void fcode(char *str){
    int i=0;
      while(str[i]){
        str[i++]++;}
}
 int   main(){
    char ss[]="Ab cde fghi jklm nop qrst";
    printf("    %s\n",ss);
    printf("  len_ss  =%d\n",lens(ss));
     fcode(ss);
   printf("     %s\n", ss);
 return 0;
}
```

Задача. Переставить каждые два символа в строке

```
Ab cde fghi jklm nop qrst

   len_ss  =25

Bc!def!ghij!klmn!opq!rstu
```

```
1. void change (char x, char y)
   { char k=x;
       x=y;
      y=k;
   }// Ошибочная запись!
 2. void change (char *x, char*y)
{  char k=*x;
      *x=*y;
      *y=k;
}//Правильная запись
```

```
 int i=0;int k=lens(ss);
   for (i=0;i<=k;i++)
 {change(&ss[i],&ss[i+1]);
   i++;
}
   printf(" New %s\n", ss);
 New   cBd!feg!ih!jlknmo!qpr!ts
```