МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский политехнический университет Петра Великого»

Институт компьютерных наук и кибербезопасности

Высшая школа технологий и искусственного интеллекта

Направление: 02.03.01 Математика и компьютерные науки

Курсовая работа по дисциплине Математические основы программирования «Блок-схема алгоритма и реализация программы»

Студент,
группа 5130201/40001 _____ **Ови Мд Шамин Ясир**

Преподаватель,
_____ **Востров Алексей В.**

«_____»_____ **2024г.**

Санкт-Петербург, 2024

Содержание

Введение

Математические основы программирования играют важную роль в проектировании алгоритмов и решении задач программирования. В данной лабораторной работе рассматривается задача манипуляции строками, которая является ключевой во многих приложениях.

Задача заключается в реализации программы, которая принимает строковый ввод пользователя и отображает его в формате "облачка речи"с использованием ASCII-графики. Алгоритм должен обрабатывать введенный текст таким образом, чтобы:

- Текст не превышал 40 символов в ширину, при этом слова не должны разрываться.
- Текст должен быть корректно разбит на строки, не нарушая целостности слов.
- Вывод должен содержать границы облачка речи, а также нарисованное ASCII-искусство с хвостиком, символизирующее облачко.

Программа использует динамическое выделение памяти, алгоритмы переноса текста и форматирования вывода с учетом всех ограничений задачи. В дополнение к программному решению, будет представлена блок-схема, иллюстрирующая структуру алгоритма.

1 Математическое описание

На рис.1 представлена блок-схема, демонстрирующая работу алгоритма:

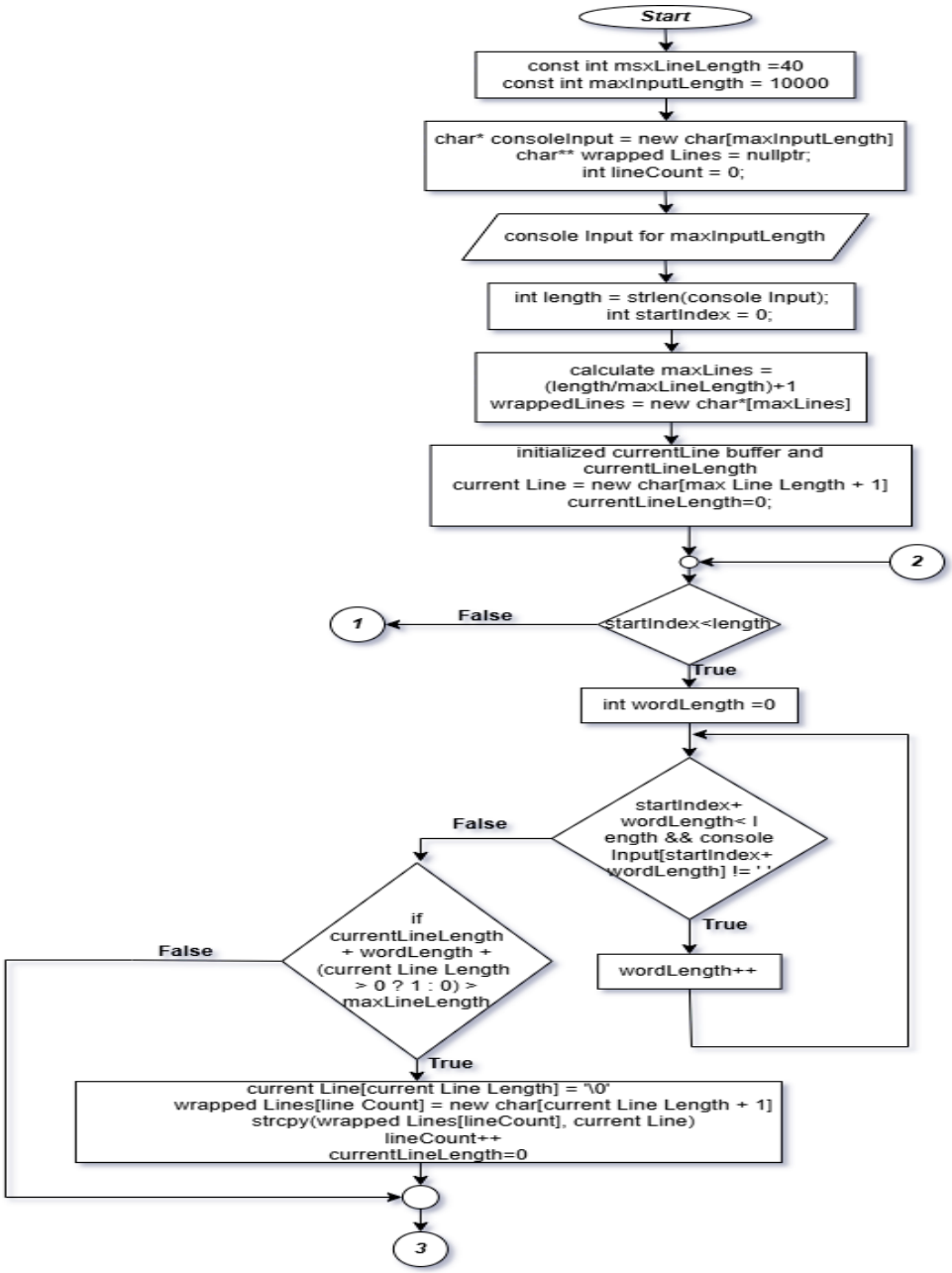

Рис. 1 Блок-схема алгоритма

На рис.2 представлена блок-схема, демонстрирующая работу алгоритма:

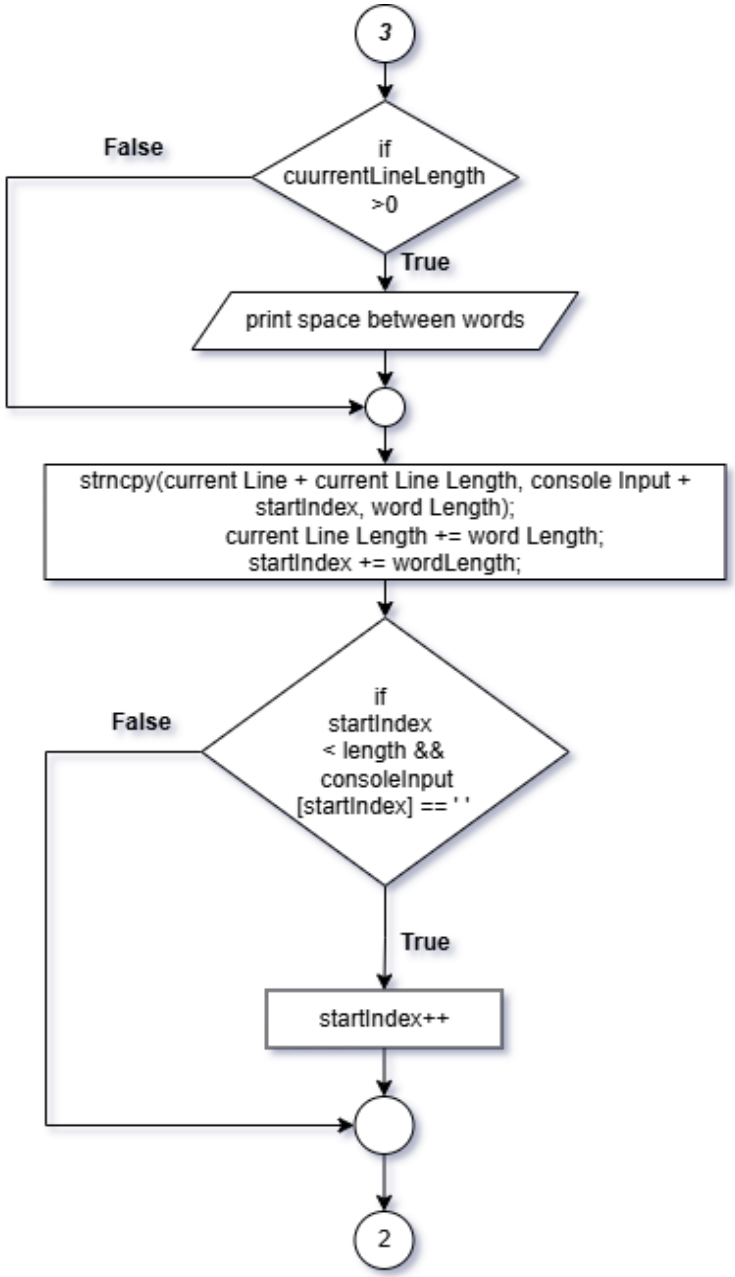

Рис. 2 Блок-схема алгоритма

На рис.3 представлена блок-схема, демонстрирующая работу алгоритма:

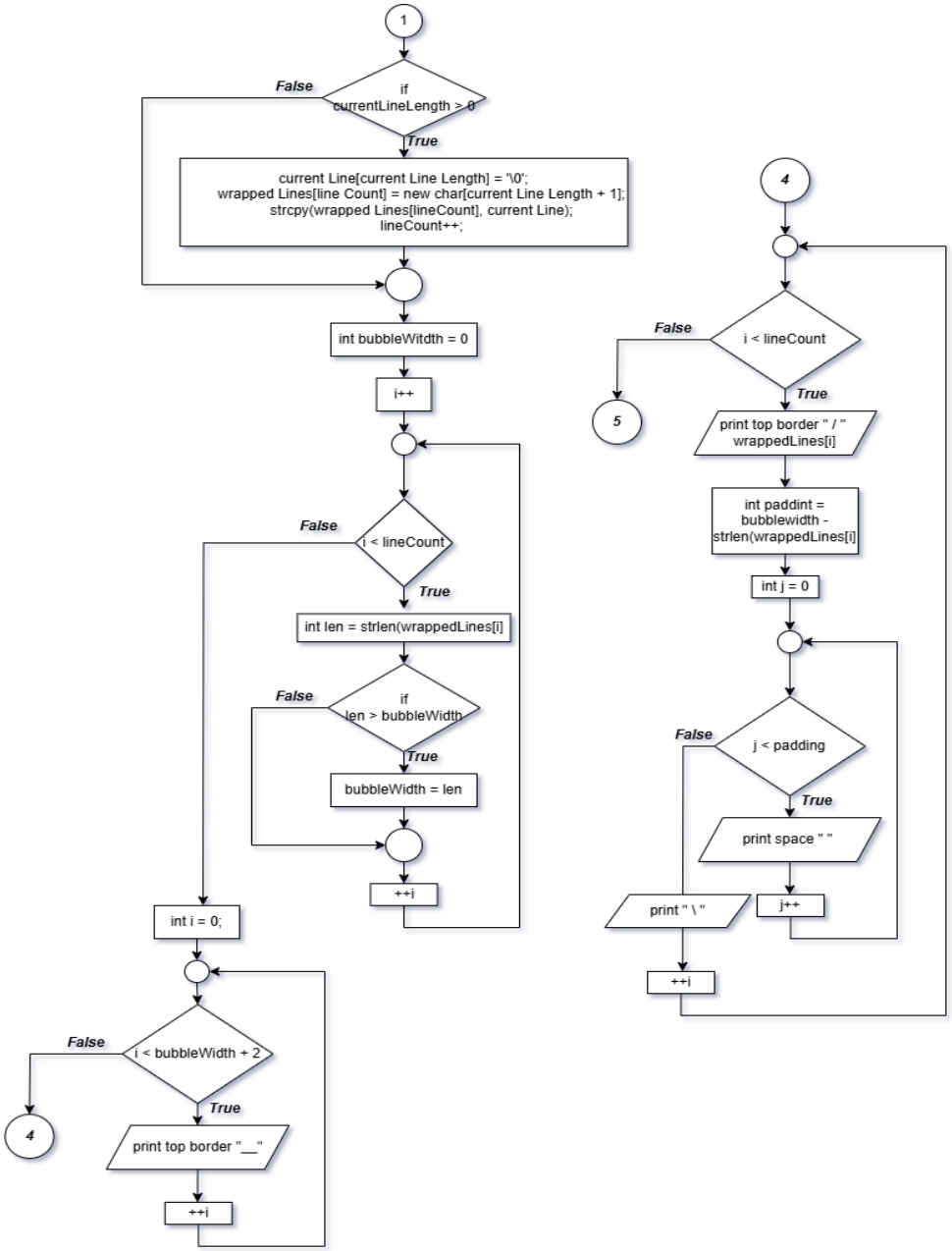

Рис. 3 Блок-схема алгоритма

На рис.4 представлена блок-схема, демонстрирующая работу алгоритма:

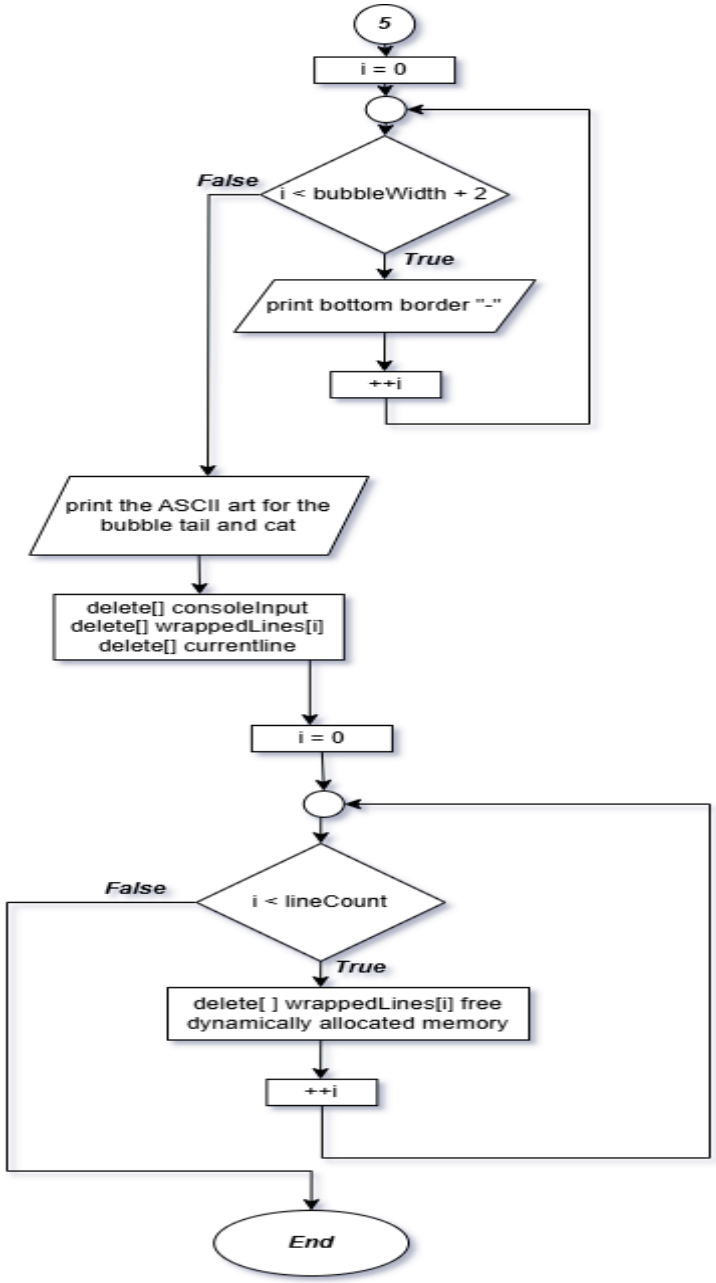

Рис. 4 Блок-схема алгоритма

2 Особенности реализации

Этот код на C++ предназначен для того, чтобы принимать ввод пользователя и разбивать его на несколько строк, гарантируя, что ни одна строка не будет превышать максимальную длину в 40 символов, при этом не разделяя слова. Программа начинается с определения констант для максимальной длины ввода и строки, после чего динамически выделяется память для хранения ввода пользователя и обернутых строк. Программа запрашивает у пользователя ввод строки, вычисляет, сколько строк нужно для того, чтобы вписать ввод в лимит 40 символов, и выделяет память для этих строк. Временный буфер используется для формирования каждой строки, и отслеживается длина текущей строки. Однако логика разбивки текста на строки без разбиения слов не была полностью реализована в предоставленном фрагменте кода.

```
#include <iostream>
#include <cstring>//for strcmp
using namespace std;
int main()
{
  const int maxLineLength = 40;//max length for each line inside the speech bubble

  // Dynamically allocate memory for user input
  const int maxInputLength = 10000;//maximum length of input
  char* consoleInput = new char[maxInputLength];// pointer for dynamic user input
  char** wrappedLines = nullptr;// pointer to dynamically allocated array of lines
  int lineCount = 0;//variable to track the number of lines

  cout << "enter words: ";
  cin.getline(consoleInput, maxInputLength);

  int length = strlen(consoleInput);
  int startIndex = 0;

  //without breaking words
  int maxLines = (length / maxLineLength) + 1;
  wrappedLines = new char*[maxLines];//dynamically allocate memory for each line

  //max lineint
  char* currentLine = new char[maxLineLength + 1]; // Temporary buffer for each line
  int currentLineLength = 0; // Keep track of the current line length
```

Эта часть кода обрабатывает ввод текста, добавляя слова в строки, не разделяя их. Он вычисляет длину каждого слова и проверяет, не превышает ли его добавление в текущую строку максимальную длину. Если превышает, текущая строка сохраняется, и начинается новая строка. Слово добавляется в текущую строку с пробелом, если это необходимо. Код использует ‘strncpy‘ для копирования слова в строку, обновляет длину строки и переходит к следующему слову. Если после слова есть пробел, он пропускается.

```
      while (startIndex < length) {
  int wordLength = 0;

  //finding the length of next word
  while (startIndex + wordLength < length && consoleInput[startIndex +
   wordLength] != ' ') {
    wordLength++;
  }

  //if console word exceeds the line length, to next line
  if (currentLineLength + wordLength + (currentLineLength > 0 ? 1 : 0) >
    maxLineLength) {
    // save current line
    currentLine[currentLineLength] = '\0';
    wrappedLines[lineCount] = new char[currentLineLength + 1];
    strcpy(wrappedLines[lineCount], currentLine);
    lineCount++;

    //starting a new line
    currentLineLength = 0;
  }

  //adding the word to the current line
  if (currentLineLength > 0) {
    currentLine[currentLineLength++] = ' ';//space between words
  }

  strncpy(currentLine + currentLineLength, consoleInput + startIndex,
   wordLength);
  currentLineLength += wordLength;

  //move to the next word
  startIndex += wordLength;

  //skip over the space if it exists
  if (startIndex < length && consoleInput[startIndex] == ' ') {
    startIndex++;
  }
}


```

Этот код находит самую длинную строку в ‘wrappedLines‘, чтобы определить ширину пузыря (‘bubbleWidth‘). Он проходит по каждой строке, вычисляет её длину и обновляет ‘bubbleWidth‘, если текущая строка длиннее.

```
    //add the last line
    if (currentLineLength > 0) {
      currentLine[currentLineLength] = '\0';
      wrappedLines[lineCount] = new char[currentLineLength + 1];
      strcpy(wrappedLines[lineCount], currentLine);
      lineCount++;
    }

```

Этот код генерирует ASCII-стиль пузыря речи, сначала определяя его ширину на основе самой длинной строки текста. Затем он выводит верхнюю и нижнюю границы с использованием подчеркиваний и дефисов соответственно. Каждая строка текста отображается внутри пузыря с соответствующими отступами для обеспечения одинаковой ширины, а слева и справа добавляются границы. В конце код выводит ASCII-арт кошки с хвостом под пузырем.

```
    //find the longest line length (for bubble width calculation)
  int bubbleWidth = 0;
  for (int i = 0; i < lineCount; ++i) {
    int len = strlen(wrappedLines[i]);
    if (len > bubbleWidth) {
      bubbleWidth = len;
    }
  }

  cout << " ";
  for (int i = 0; i < bubbleWidth + 2; ++i) {//print the top border of the bubble

    cout << "_";
  }
  cout << endl;

  for (int i = 0; i < lineCount; ++i) {//print each line inside the bubble

    cout << "/ "; //left border
    cout << wrappedLines[i];

    int padding = bubbleWidth - strlen(wrappedLines[i]);//ensure that bubbleWidth characters before the closing border
    for (int j = 0; j < padding; ++j) {
      cout << " "; // padding with spaces
    }
    cout << " \\" << endl; //right border
  }

  //print the bottom border of the bubble
  cout <<" ";
  for (int i = 0; i< bubbleWidth + 2; ++i) {
    cout << "-";
  }
  cout << endl;

  //print the ASCII art for the bubble tail and cat
  cout << "     \\ " << endl;
  cout << "      \\ " << endl;
  cout << "        /\\_/\\  (" << endl;
  cout << "       ( ^.^ ) _)" << endl;
  cout << "         \\\"/  (" << endl;
  cout << "       (   )" << endl;
  cout << "      (__d b__)" << endl;

```

Эта часть кода освобождает всю динамически выделенную память, включая ввод пользователя (‘consoleInput‘), строки в ‘wrappedLines‘, массив ‘wrappedLines‘ и временный буфер (‘currentLine‘), предотвращая утечки памяти.

```
      delete[] consoleInput;//free dynamically allocated memory
for (int i = 0; i < lineCount; ++i) {
  delete[] wrappedLines[i];
}
delete[] wrappedLines; //free the wrappedLines array
delete[] currentLine; //free the temporary buffer for lines


}

```

3 Результаты работы

Программа была протестирована с различными сценариями ввода, включая правильный ввод, вывод, превышающий максимальную длину, пустой ввод и специальные символы. В отчете приведены скриншоты, демонстрирующие, как программа обрабатывает эти сценарии и выдает ожидаемые результаты.Ниже приведены некоторые результаты работы программы (рис. 1, рис. 2, рис. 3).

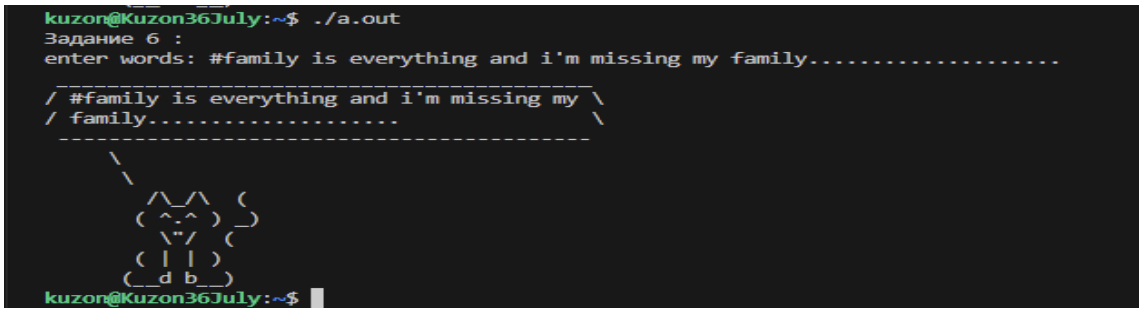

Рис. 1 нормальное предложение

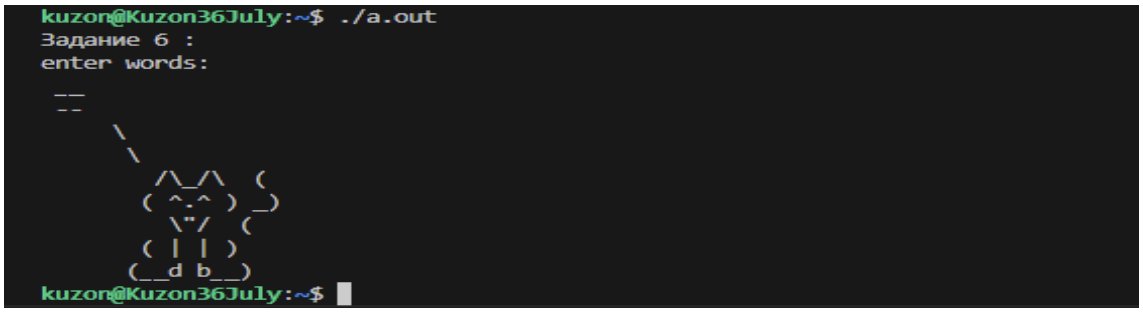

Рис. 2 космический выход

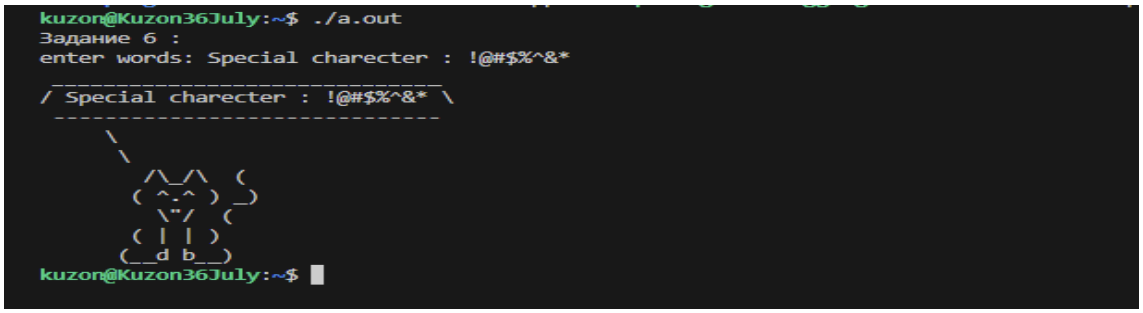

Рис. 3 Вывод специальных символов

Заключение

В данной лабораторной работе была успешно реализована программа, отображающая ввод пользователя в виде «облачка речи» с использованием ASCII-графики. Программа обеспечивает соблюдение ограничений форматирования, таких как ограничение длины строки до 40 символов без разрыва слов и корректное разбиение текста с сохранением целостности слов. Вывод включает хорошо структурированное облачко речи с границами и хвостиком, представленным в ASCII-графике. Программа эффективно управляет разбиением текста, использует динамическое выделение памяти для гибкости и позволяет легко настраивать параметры, такие как длина строки и размер ввода. Однако у программы есть некоторые ограничения. Её сложность, обусловленная использованием множества циклов и динамическим управлением памятью, усложняет обслуживание. Хотя программа достигает своих основных целей и демонстрирует практическое применение методов обработки текста, устранение этих ограничений повысит её надёжность, удобство использования и масштабируемость.

Список использованной литературы

[1] Полубенцева, М.И. C/C++. Процедурное программирование./ М.И.Полубенцева. Санкт-Петербург : БХВ-Петербург, 2017. - 417 с.

[2] Дейтел, Х.М. Как программировать на C++. / Х. М. Дейтел, П. Дж. Дейтел; перевод. с английского В.В. Тимофеев. Москва: Бином, 2008.- 1454 с.

[3] Павловская, Т.А. C/C++ Программирование на языке высокого уровня. / Т.А. Павловская. Санкт-Петербург : Питер, 2003. 460 с.

[4] Лекции преподавателя, доступные по ссылке: http://13633.mooo.com/.