Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Волгоградский государственный технический университет»

Факультет электроники и вычислительной техники

Кафедра «Программное обеспечение автоматизированных систем»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовой работе**

по дисциплине «Программирование»

на тему: «Реализация программы с применением процедурно-ориентированного подхода»

(индивидуальное задание – вариант №12)

Студент: Ковалева Анна Алексеевна

Группа: ПрИн-267

Работа зачтена с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

« » 2022 г.

Руководитель работы, нормоконтролер \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Литовкин Д.В.

Волгоград 2021 г.

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Волгоградский государственный технический университет»

Факультет электроники и вычислительной техники

Направление 09.03.04 «Программная инженерия»

Кафедра «Программное обеспечение автоматизированных систем»

Дисциплина «Программирование»

Утверждаю

Зав. кафедрой \_\_\_\_\_\_\_\_\_\_ Орлова Ю.А.

**ЗАДАНИЕ**

**на курсовую работу**

Студент: Ковалева Анна Алексеевна

Группа: ПрИн-267

1. Тема: «Реализация программы с применением процедурно-ориентированного подхода» (индивидуальное задание – вариант №12)

Утверждена приказом ректора № 1171-ст от « 29 » сентября 2021 г.

2. Срок представления работы к защите « 23 » декабря 2021 г.

3. Содержание пояснительной записки: формулировка задания, иерархия вызова функций, проектирование функций.

4. Перечень графического материала: иерархия вызова функций.

5. Дата выдачи задания « 30 » сентября 2021 г.

Руководитель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Литовкин Д.В.

Задание приняла к исполнению: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ковалева А.А.

« 30 » сентября 2021 г.

**Содержание**

1. Формулировка задания………………………………...……….………………4

2. Иерархия вызова функций………………..……………………………………5

3. Проектирование функции generatePrototip, решающей главную задачу….…6

3.1 Прототип…………………………………………………………………….…6

3.2 Модульные тесты………………………………...……………………………6

3.3 Структура данных…………………………………………………………....12

3.4 Псевдокод………………………………………………………………...…..12

4. Проектирование функции findEntryName…………..……………………….14

4.1 Прототип……………………………………………………………………...14

4.2 Модульные тесты………………………………………………………….....14

4.3 Структура данных……………………………………………………………16

4.4 Псевдокод……………………………………………………………….……16

5. Проектирование функции removeCommentsFromFunctionText.……………18

5.1 Прототип……………………………………………………………...………18

5.2 Модульные тесты………………………………………………………….…18

5.3 Структура данных……………………………………………………………18

5.4 Псевдокод………………………………………………………….…………18

6. Проектирование функции cutPartStr……...………………………..…………19

6.1 Прототип……………………………………………………………...………19

6.2 Модульные тесты……………………………………………………….……19

6.3 Структура данных……………………………………………………………21

6.4 Псевдокод……………………………………………………………….……21

7. Список использованной литературы и других источников…………………22

Приложение А. Исходный код программы……………………………………23

Приложение Б. Трасса тестирования и отладки…………………………….…27

**1. Формулировка задания**

Дан текст программы на языке Си (многострочный текст).

Текст программы синтаксически верен за исключением одного случая (см. ниже) и имеет следующее форматирование: каждый элементарный оператор языка записывается в отдельной строке, а составные операторы заключаются в фигурные скобки, фигурные скобки располагаются на отдельных строках.

В программе не используются глобальные переменные, структурированные типы данных, типы данных «ссылка» и «указатель», символьные и строковые константы, но могут встречаться однострочные комментарии (“//”).

Сгенерируйте для заданной функции ее прототип (прототип в тексте программы может, как присутствовать, так и отсутствовать, в последнем случае может быть синтаксическая ошибка). Если в программе уже присутствует прототип, то использовать его.

**Формат входных данных.**

[имя функции]

[M]

[1-я строка]

[2-я строка] и т.д.

M – кол-во строк в тексте, целое число в диапазоне [1, 20].

Считать, что пользователь может указать только допустимый идентификатор. Ошибочность такой ситуации не проверять.

**Формат выходных данных.** [прототип функции] или «no solution», если указанная функция не имеет описания.

**2. Иерархия вызова функций**

На рисунке 1 показана иерархия вызова функций с учетом библиотечных функций работы со строками.

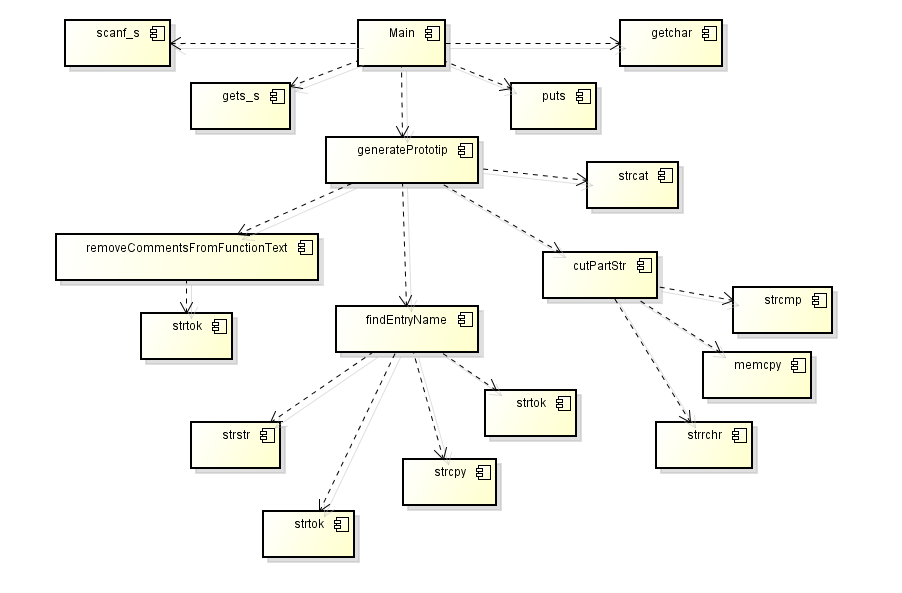


Рисунок 1. Иерархия вызовов функций с учетом библиотечных функций работы со строками

**3. Проектирование функции generatePrototip, решающей главную задачу**

**3.1 Прототип**

/\*!Главная функция, которая генерирует прототип

\ param [in] textFunction - текст функции

\ param [in] nameFunction - название функции

\ param [in] numberRows - число строк в тексте функции

\ param [out] strPrototip - строка с прототипом

\*/

int generatePrototip (char textFunction[][MAX\_NUMBER\_SYMBOLS], int numberRows, char nameFunction[], char strPrototip[]);

/\*Возвращает 1, прототип сгенерирован\найден;

Возвращает 0, прототип не найден \*/

**3.2 Модульные тесты**

**3.2.1 Аспекты тестирования**

|  |  |
| --- | --- |
| **Аспекты тестирования** | **Варианты значений** |
| Существование прототипа в тексте | есть\***нет*** |
| Место прототипа в тексте функции | ***вверху(правильно)\***внизу(синтаксическая ошибка) |
| Использование искомой функции | ***есть***\нет |
| Присутствие комментариев в тексте | есть\***нет*** |
| Наличие пробелов перед комментарием | ***есть***\нет |
| Место комментариев | после прототипа\после заголовка\***другое*** |
| Белые разделители перед прототипом | есть\***нет*** |
| Существование функции в тексте | ***есть***\нет |
| Количество строк текста функции | 1\***несколько***\20 |

**3.2.2 "Типовой" тест**

*Входные данные:*

summa

12

int main()

{

int a = 1;

int b = 2;

int sum = summa (a, b);

return 0;

}

int summa (int one\_number, int two\_number)

{

int sum = one\_number + two\_number;

return sum;

}

*Выходные данные:*

int summa (int one\_number, int two\_number);

**3.2.3 Реализация модульных тестов**

TEST\_METHOD(no\_prototip)

{

char prototipForTest[MAX\_NUMBER\_SYMBOLS] = "int summa(int one\_number, int two\_number);";

char nameFunction[10] = "summa";

char textFunction[MAX\_NUMBER\_ROWS][MAX\_NUMBER\_SYMBOLS] =

{

"int main()",

"{",

"int a = 1;",

"int b = 2;",

"int sum = summa (a, b);",

"return 0;",

"}",

"int summa(int one\_number, int two\_number)",

"{",

"int sum = one\_number + two\_number;",

"return sum;",

"}"

};

int numberSymbols = 12;

char strPrototip[MAX\_NUMBER\_SYMBOLS];

int exemple = generatePrototip(textFunction, numberSymbols, nameFunction, strPrototip);

Assert::AreEqual(prototipForTest, strPrototip);

}

TEST\_METHOD(have\_prototip)

{

char prototipForTest[MAX\_NUMBER\_SYMBOLS] = "int summa (int, int);";

char nameFunction[10] = "summa";

char textFunction[MAX\_NUMBER\_ROWS][MAX\_NUMBER\_SYMBOLS] =

{

"int summa (int, int);",

"int main()",

"{",

"int a = 1;",

"int b = 2;",

"int sum = summa (a, b);",

"return 0;",

"}",

"int summa (int one\_number, int two\_number)",

"{",

"int sum = one\_number + two\_number;",

"return sum;",

"}"

};

int numberSymbols = 13;

char strPrototip[MAX\_NUMBER\_SYMBOLS];

int exemple = generatePrototip(textFunction, numberSymbols, nameFunction, strPrototip);

Assert::AreEqual(prototipForTest, strPrototip);

}

TEST\_METHOD(prototip\_with\_comment\_without\_spaces)

{

char prototipForTest[MAX\_NUMBER\_SYMBOLS] = "int summa (int, int);";

char nameFunction[10] = "summa";

char textFunction[MAX\_NUMBER\_ROWS][MAX\_NUMBER\_SYMBOLS] =

{

"int summa (int, int);//прототип",

"int main()",

"{",

"int a = 1;",

"int b = 2;",

"int sum = summa (a, b);",

"return 0;",

"}",

"int summa(int one\_number, int two\_number)",

"{",

"int sum = one\_number + two\_number;",

"return sum;",

"}"

};

int numberSymbols = 13;

char strPrototip[MAX\_NUMBER\_SYMBOLS];

int exemple = generatePrototip(textFunction, numberSymbols, nameFunction, strPrototip);

Assert::AreEqual(prototipForTest, strPrototip);

}

TEST\_METHOD(prototip\_with\_comment\_with\_spaces)

{

char prototipForTest[MAX\_NUMBER\_SYMBOLS] = "int summa (int, int);";

char nameFunction[10] = "summa";

char textFunction[MAX\_NUMBER\_ROWS][MAX\_NUMBER\_SYMBOLS] =

{

"int summa (int, int); //прототип",

"int main()",

"{",

"int a = 1;",

"int b = 2;",

"int sum = summa (a, b);",

"return 0;",

"}",

"int summa(int one\_number, int two\_number)",

"{",

"int sum = one\_number + two\_number;",

"return sum;",

"}"

};

int numberSymbols = 13;

char strPrototip[MAX\_NUMBER\_SYMBOLS];

int exemple = generatePrototip(textFunction, numberSymbols, nameFunction, strPrototip);

Assert::AreEqual(prototipForTest, strPrototip);

}

TEST\_METHOD(header\_with\_comment\_without\_spaces)

{

char prototipForTest[MAX\_NUMBER\_SYMBOLS] = "int summa(int one\_number, int two\_number);";

char nameFunction[10] = "summa";

char textFunction[MAX\_NUMBER\_ROWS][MAX\_NUMBER\_SYMBOLS] =

{

"int main()",

"{",

"int a = 1;",

"int b = 2;",

"int sum = summa (a, b);",

"return 0;",

"}",

"int summa(int one\_number, int two\_number)//заголовок",

"{",

"int sum = one\_number + two\_number;",

"return sum;",

"}"

};

int numberSymbols = 12;

char strPrototip[MAX\_NUMBER\_SYMBOLS];

int exemple = generatePrototip(textFunction, numberSymbols, nameFunction, strPrototip);

Assert::AreEqual(prototipForTest, strPrototip);

}

TEST\_METHOD(header\_with\_comment\_with\_spaces)

{

char prototipForTest[MAX\_NUMBER\_SYMBOLS] = "int summa(int one\_number, int two\_number);";

char nameFunction[10] = "summa";

char textFunction[MAX\_NUMBER\_ROWS][MAX\_NUMBER\_SYMBOLS] =

{

"int main()",

"{",

"int a = 1;",

"int b = 2;",

"int sum = summa (a, b);",

"return 0;",

"}",

"int summa(int one\_number, int two\_number) //заголовок",

"{",

"int sum = one\_number + two\_number;",

"return sum;",

"}"

};

int numberSymbols = 12;

char strPrototip[MAX\_NUMBER\_SYMBOLS];

int exemple = generatePrototip(textFunction, numberSymbols, nameFunction, strPrototip);

Assert::AreEqual(prototipForTest, strPrototip);

}

TEST\_METHOD(no\_function)

{

char nameFunction[10] = "summa";

char textFunction[MAX\_NUMBER\_ROWS][MAX\_NUMBER\_SYMBOLS] =

{

"int main()",

"{",

"int a = 1;",

"int b = 2;",

"int sum = summaAB (a, b);",

"return 0;",

"}",

"int summaAB(int one\_number, int two\_number)",

"{",

"int sum = one\_number + two\_number;",

"return sum;",

"}"

};

int numberSymbols = 12;

char strPrototip[MAX\_NUMBER\_SYMBOLS];

int exemple = generatePrototip(textFunction, numberSymbols, nameFunction, strPrototip);

Assert::AreEqual(0, exemple);

}

TEST\_METHOD(prototip\_located\_in\_end)

{

char prototipForTest[MAX\_NUMBER\_SYMBOLS] = "int summa(int, int);";

char nameFunction[10] = "summa";

char textFunction[MAX\_NUMBER\_ROWS][MAX\_NUMBER\_SYMBOLS] =

{

"int main()",

"{",

"int a = 1;",

"int b = 2;",

"int sum = summa (a, b);",

"return 0;",

"}",

"int summa(int one\_number, int two\_number)",

"{",

"int sum = one\_number + two\_number;",

"return sum;",

"}",

"int summa(int, int);"

};

int numberSymbols = 13;

char strPrototip[MAX\_NUMBER\_SYMBOLS];

int exemple = generatePrototip(textFunction, numberSymbols, nameFunction, strPrototip);

Assert::AreEqual(prototipForTest, strPrototip);

}

TEST\_METHOD(spaces\_before\_prototip)

{

char prototipForTest[MAX\_NUMBER\_SYMBOLS] = "int summa(int, int);";

char nameFunction[10] = "summa";

char textFunction[MAX\_NUMBER\_ROWS][MAX\_NUMBER\_SYMBOLS] =

{

"int main()",

"{",

"int a = 1;",

"int b = 2;",

"int sum = summa (a, b);",

"return 0;",

"}",

"int summa(int one\_number, int two\_number)",

"{",

"int sum = one\_number + two\_number;",

"return sum;",

"}",

" int summa(int, int);"

};

int numberSymbols = 13;

char strPrototip[MAX\_NUMBER\_SYMBOLS];

int exemple = generatePrototip(textFunction, numberSymbols, nameFunction, strPrototip);

Assert::AreEqual(prototipForTest, strPrototip);

}

TEST\_METHOD(tab\_before\_prototip)

{

char prototipForTest[MAX\_NUMBER\_SYMBOLS] = "int summa(int, int);";

char nameFunction[10] = "summa";

char textFunction[MAX\_NUMBER\_ROWS][MAX\_NUMBER\_SYMBOLS] =

{

"int main()",

"{",

"int a = 1;",

"int b = 2;",

"int sum = summa (a, b);",

"return 0;",

"}",

"int summa(int one\_number, int two\_number)",

"{",

"int sum = one\_number + two\_number;",

"return sum;",

"}",

"\t\tint summa(int, int);"

};

int numberSymbols = 13;

char strPrototip[MAX\_NUMBER\_SYMBOLS];

int exemple = generatePrototip(textFunction, numberSymbols, nameFunction, strPrototip);

Assert::AreEqual(prototipForTest, strPrototip);

}

TEST\_METHOD(max\_size\_rows)

{

char prototipForTest[MAX\_NUMBER\_SYMBOLS] = "int summa(int, int);";

char nameFunction[10] = "summa";

char textFunction[MAX\_NUMBER\_ROWS][MAX\_NUMBER\_SYMBOLS] =

{

"int main()",

"{",

"int a = 1;",

"int b = 2;",

"int c = 3;",

"int d = 4;",

"int e = 5;",

"int j = 6;",

"int z = 7;",

"int i = 8;",

"int k = 9;",

"int sum = summa (a, b);",

"return 0;",

"}",

"int summa(int one\_number, int two\_number)",

"{",

"int sum = one\_number + two\_number;",

"return sum;",

"}",

"int summa(int, int);"

};

int numberSymbols = 20;

char strPrototip[MAX\_NUMBER\_SYMBOLS];

int exemple = generatePrototip(textFunction, numberSymbols, nameFunction, strPrototip);

Assert::AreEqual(prototipForTest, strPrototip);

}

TEST\_METHOD(min\_size\_rows)

{

char prototipForTest[MAX\_NUMBER\_SYMBOLS] = "int summa(int, int);";

char nameFunction[10] = "summa";

char textFunction[MAX\_NUMBER\_ROWS][MAX\_NUMBER\_SYMBOLS] =

{

"int summa(int, int);"

};

int numberSymbols = 1;

char strPrototip[MAX\_NUMBER\_SYMBOLS];

int exemple = generatePrototip(textFunction, numberSymbols, nameFunction, strPrototip);

Assert::AreEqual(prototipForTest, strPrototip);

}

**3.3 Структура данных**

int indexForFindEntryName; //Переменная для функции, которая определяет тип вхождения заданного имени.

int header; //Флаг заголовка.

char desiredWhiteSpacers; //Допустимые белые разделители.

**3.4 Псевдокод**

/\*!Главная функция, которая генерирует прототип

\ param [in] textFunction - текст функции

\ param [in] nameFunction - название функции

\ param [in] numberRows - число строк в тексте функции

\ param [out] strPrototip - строка с прототипом

\*/

int generatePrototip (char textFunction[][MAX\_NUMBER\_SYMBOLS], int numberRows, char nameFunction[], char strPrototip[])

{

//Убрать комментарии из текста

//Сгенерировать прототип, пройдясь по всему тексту...

{

//Определить, является ли строчка прототипом или заголовком

//Если найден прототип, то обрезаем лишние пробелы и помещаем его в результирующий массив

{

}

//Иначе если найден заголовок, то обрезаем лишние пробелы и помещаем его в результирующий массив

{

}

}

//Если истинный прототип в тексте не найден, возвращаем сгенерированный прототип

{

}

//Иначе ничего не возвращаем

{

}

}

**4. Проектирование функции findEntryName**

**4.1 Прототип**

/\*!Функция, которая определяет вид строки

\ param [in] textFunction - строка

\ param [in] nameFunction - название функции

\*/

int findEntryName (char textFunction[], char nameFunction[]);

/\*Возвращает 1 - прототип;

Возвращает 0 - заголовок;

Возвращает -1 - не найдено\*/

**4.2 Модульные тесты**

**4.2.1 Аспекты тестирования**

|  |  |
| --- | --- |
| **Аспекты тестирования** | **Варианты значений** |
| Присутствие в строке имени функции | ***есть***\нет |
| Чем является строка с именем функции | ***прототип***\заголовок\объявление\использование |
| Чем является вхождение имени функции | ***истинным именем функции***\частью другого слова |
| Длина возвращаемого значения | ***одно слово***\два слова\три слова |
| Начальное слово возвращаемого значения | ***базовое возвращаемое значение***\модификатор |
| Имя функции совпадает с именем переменной | да\***нет*** |

**4.2.2 "Типовой" тест**

Входное значение:

summa

int summa(int one\_number, int two\_number);

Выходное значение:

1

**4.2.3 Реализация модульных тестов**

TEST\_METHOD(no\_name)

{

char strForTest[MAX\_NUMBER\_SYMBOLS] = "int summa(int one\_number, int two\_number);";

char nameFunction[10] = "add";

int exemple = findEntryName(strForTest, nameFunction);

int trueExemple = -1;

Assert::AreEqual(exemple, trueExemple);

}

TEST\_METHOD(prototip)

{

char strForTest[MAX\_NUMBER\_SYMBOLS] = "int summa(int one\_number, int two\_number);";

char nameFunction[10] = "summa";

int exemple = findEntryName(strForTest, nameFunction);

int trueExemple = 1;

Assert::AreEqual(exemple, trueExemple);

}

TEST\_METHOD(header)

{

char strForTest[MAX\_NUMBER\_SYMBOLS] = "int summa(int one\_number, int two\_number)";

char nameFunction[10] = "summa";

int exemple = findEntryName(strForTest, nameFunction);

int trueExemple = 0;

Assert::AreEqual(exemple, trueExemple);

}

TEST\_METHOD(no\_true\_name)

{

char strForTest[MAX\_NUMBER\_SYMBOLS] = "int summator(int one\_number, int two\_number);";

char nameFunction[10] = "summa";

int exemple = findEntryName(strForTest, nameFunction);

int trueExemple = -1;

Assert::AreEqual(exemple, trueExemple);

}

TEST\_METHOD(use\_function)

{

char strForTest[MAX\_NUMBER\_SYMBOLS] = "int a = summa(one\_number, two\_number);";

char nameFunction[10] = "summa";

int exemple = findEntryName(strForTest, nameFunction);

int trueExemple = -1;

Assert::AreEqual(exemple, trueExemple);

}

TEST\_METHOD(advert\_variable)

{

char strForTest[MAX\_NUMBER\_SYMBOLS] = "int summa = 0;";

char nameFunction[10] = "summa";

int exemple = findEntryName(strForTest, nameFunction);

int trueExemple = -1;

Assert::AreEqual(exemple, trueExemple);

}

TEST\_METHOD(longest\_prototip)

{

char strForTest[MAX\_NUMBER\_SYMBOLS] = "long long int summa(int one\_number, int two\_number);";

char nameFunction[10] = "summa";

int exemple = findEntryName(strForTest, nameFunction);

int trueExemple = 1;

Assert::AreEqual(exemple, trueExemple);

}

TEST\_METHOD(mini\_prototip)

{

char strForTest[MAX\_NUMBER\_SYMBOLS] = "int summa();";

char nameFunction[10] = "summa";

int exemple = findEntryName(strForTest, nameFunction);

int trueExemple = 1;

Assert::AreEqual(exemple, trueExemple);

}

TEST\_METHOD(prototip\_with\_modifiers)

{

char strForTest[MAX\_NUMBER\_SYMBOLS] = "long summa();";

char nameFunction[10] = "summa";

int exemple = findEntryName(strForTest, nameFunction);

int trueExemple = 1;

Assert::AreEqual(exemple, trueExemple);

}

TEST\_METHOD(function\_name\_same\_variable\_name)

{

char strForTest[MAX\_NUMBER\_SYMBOLS] = "int summa = summa();";

char nameFunction[10] = "summa";

int exemple = findEntryName(strForTest, nameFunction);

int trueExemple = -1;

Assert::AreEqual(exemple, trueExemple);

}

**4.3 Структура данных**

char\* pointerFirstWord; //Четыре переменные – указатели на слова в строке по порядку.

char\* pointerSecondWord;

char\* pointerThirdWord;

char\* pointerFourthWord;

char wordBreakingStr; //Строка, созданная, чтобы разделять её на слова.

char strSearchSpecialChars; //Строка для нахождения определенных символов.

bool indexFirstSpecialWord; //Переменная для определения, есть ли в строке базовые возвращаемые значения или\и модификаторы.

char\* pointerNameFunction; //Указатель на предполагаемое имя функции.

**4.4 Псевдокод**

/\*!Функция, которая определяет вид строки

\ param [in] textFunction - строка

\ param [in] nameFunction - название функции

\*/

int findEntryName(char textFunction[], char nameFunction[])

{

//Проверка на наличие имени функции в строке...

{

//Наличие базовых типов возвращаемого значения...

{

//Проверка формата строки...

{

//Если в строке есть имя функции...

{

//Если в строке есть открывающая и закрывающая круглые скобки...

{

//Если в строке присутствует точка с запятой...

{

}

}

}

}

}

//Наличие модификаторов для базовых типов...

{

//Проверка формата строки...

{

//Если в строке есть имя функции...

{

//Если в строке есть открывающая и закрывающая круглые скобки...

{

//Если в строке присутствует точка с запятой...

{

}

}

}

}

}

}

//Иначе не найдено

{

}

}

**5. Проектирование функции removeCommentsFromFunctionText**

**5.1 Прототип**

/\*!Функция, которая убирает комментарии из всего текста

\ param[in] textFunction - текст функции

\ param[in] numberRows - число строк в тексте функции \*/

void removeCommentsFromFunctionText(char textFunction[][MAX\_NUMBER\_SYMBOLS], int numberRows);

**5.2 Модульные тесты**

Функция слишком проста, чтобы создавать модульные тесты.

**5.3 Структура данных**

**-**

**5.4 Псевдокод**

Функция слишком проста, чтобы иметь псевдокод.

**6. Проектирование функции cutPartStr**

**6.1 Прототип**

/\*!Функция, которая вырезает часть строки

\ param [in] str - строка, из которой происходит вырезка

\ param [out] partStr - строка с вырезкой

\ param [in] startStr - строка с символами, которые нужно игнорировать при вырезке перед нужной частью

\ param [in] finishSymbol - символ, до которого нужно вырезать часть строки

\*/

void cutPartStr(char str[], char partStr[], char finishSymbol, char startStr[]);

**6.2 Модульные тесты**

**6.2.1 Аспекты тестирования**

|  |  |
| --- | --- |
| **Аспекты тестирования** | **Варианты значений** |
| Место неужных символов | ***спереди***\сзади\ и спереди, и сзади |
| Количество игнорируемых символов | ***один***\несколько |
| Символы по составу | ***белые разделители***\знаки препинания\ буквы\другое |
| Длина вырезанной строки | ***несколько символов***\один символ\ноль символов |

**6.2.2 "Типовой" тест**

*Входные данные:*

" int summa(int a, int b);"

(строка, куда помещается вырезанная часть строки)

';'

" "

*Выходные данные:*

"int summa(int a, int b);"

**6.2.3 Реализация модульных тестов**

TEST\_METHOD(spaces\_before\_str)

{

char str[MAX\_NUMBER\_SYMBOLS] = " int summa(int a, int b);";

char partStr[MAX\_NUMBER\_SYMBOLS];

char finishSymbol = ';';

char startStr[MAX\_NUMBER\_SYMBOLS] = " ";

char truePartStr[MAX\_NUMBER\_SYMBOLS] = "int summa(int a, int b);";

cutPartStr(str, partStr,finishSymbol, startStr);

Assert::AreEqual(partStr, truePartStr);

}

TEST\_METHOD(spaces\_after\_str)

{

char str[MAX\_NUMBER\_SYMBOLS] = "int summa(int a, int b); ";

char partStr[MAX\_NUMBER\_SYMBOLS];

char finishSymbol = ';';

char startStr[MAX\_NUMBER\_SYMBOLS] = " ";

char truePartStr[MAX\_NUMBER\_SYMBOLS] = "int summa(int a, int b);";

cutPartStr(str, partStr, finishSymbol, startStr);

Assert::AreEqual(partStr, truePartStr);

}

TEST\_METHOD(spaces\_after\_and\_before\_str)

{

char str[MAX\_NUMBER\_SYMBOLS] = " int summa(int a, int b); ";

char partStr[MAX\_NUMBER\_SYMBOLS];

char finishSymbol = ';';

char startStr[MAX\_NUMBER\_SYMBOLS] = " ";

char truePartStr[MAX\_NUMBER\_SYMBOLS] = "int summa(int a, int b);";

cutPartStr(str, partStr, finishSymbol, startStr);

Assert::AreEqual(partStr, truePartStr);

}

TEST\_METHOD(random\_chars\_before\_str)

{

char str[MAX\_NUMBER\_SYMBOLS] = "..//,, int summa(int a, int b);";

char partStr[MAX\_NUMBER\_SYMBOLS];

char finishSymbol = ';';

char startStr[MAX\_NUMBER\_SYMBOLS] = " ,./";

char truePartStr[MAX\_NUMBER\_SYMBOLS] = "int summa(int a, int b);";

cutPartStr(str, partStr, finishSymbol, startStr);

Assert::AreEqual(partStr, truePartStr);

}

TEST\_METHOD(random\_chars\_after\_str)

{

char str[MAX\_NUMBER\_SYMBOLS] = "int summa(int a, int b);//,,.. ";

char partStr[MAX\_NUMBER\_SYMBOLS];

char finishSymbol = ';';

char startStr[MAX\_NUMBER\_SYMBOLS] = " ";

char truePartStr[MAX\_NUMBER\_SYMBOLS] = "int summa(int a, int b);";

cutPartStr(str, partStr, finishSymbol, startStr);

Assert::AreEqual(partStr, truePartStr);

}

TEST\_METHOD(one\_char)

{

char str[MAX\_NUMBER\_SYMBOLS] = "a;";

char partStr[MAX\_NUMBER\_SYMBOLS];

char finishSymbol = ';';

char startStr[MAX\_NUMBER\_SYMBOLS] = "a";

char truePartStr[MAX\_NUMBER\_SYMBOLS] = ";";

cutPartStr(str, partStr, finishSymbol, startStr);

Assert::AreEqual(partStr, truePartStr);

}

TEST\_METHOD(finish\_empty\_str)

{

char str[MAX\_NUMBER\_SYMBOLS] = "a;";

char partStr[MAX\_NUMBER\_SYMBOLS];

char finishSymbol = 'a';

char startStr[MAX\_NUMBER\_SYMBOLS] = "a";

char truePartStr[MAX\_NUMBER\_SYMBOLS] = "";

cutPartStr(str, partStr, finishSymbol, startStr);

Assert::AreEqual(partStr, truePartStr);

}

**6.3 Структура данных**

char\* finishPrototiр; //Указатель на последний символ части строки.

int indexStart; //Индекс начального символа в строке.

char\* startPrototip; //Указатель на первый символ части строки.

int lengthPrototip; //Длина результирующей строки.

**6.4 Псевдокод**

/\*!Функция, которая вырезает часть строки

\ param [in] str - строка, из которой происходит вырезка

\ param [out] partStr - строка с вырезкой

\ param [in] startStr - строка с символами, которые нужно игнорировать при вырезке перед нужной частью

\ param [in] finishSymbol - символ, до которого нужно вырезать часть строки

\*/

void cutPartStr(char str[], char partStr[], char finishSymbol, char startStr[])

{

//Находим указатель на последний символ части строки

//Находим указатель на первый символ части строки

//Помещаем часть строки в результирующий массив

}

**7. Список использованной литературы и других источников**

1)Трофимов, В. В. Алгоритмизация и программирование: учебник для академического бакалавриата / В. В. Трофимов, Т. А. Павловская; под ред. В. В. Трофимова. — М.: Издательство Юрайт, 2017. — 137 с. — (Серия: Бакалавр. Академический курс. Модуль.). — ISBN 978-5-9916-9866-5. — Режим доступа: www.biblio-online.ru/book/B08DB966-3F96-4B5A-B030-E3CD9085CED4

2)Справочная служба MSDN Library – [Электронный ресурс].

3)Литовкин Д.В., Комплект презентаций к теоретическим занятиям по дисциплине «Программирование» – [Электронный ресурс].

4)Литовкин Д.В., Сычев О.А. Комплект презентаций к теоретическим занятиям по дисциплине «Основы программирования» – [Электронный ресурс].

**Приложение А. Исходный код программы**

12\_Kovaleva\_AA.cpp

#include <iostream>

#include <string.h>

#include <stdio.h>

#include "Header.h"

int main()

{

//Имя функции, для которой нужно сгенерировать прототип

char nameFunction[MAX\_NUMBER\_SYMBOLS];

//Количество строк в функции

int numberRows;

//Текст функции

char textFunction[MAX\_NUMBER\_ROWS][MAX\_NUMBER\_SYMBOLS];

//Строка, где хранится прототип

char strPrototip[MAX\_NUMBER\_SYMBOLS];

//Ввести данные...

//Ввести название функции

gets\_s(nameFunction);

//Ввести количество строк

scanf\_s("%d", &numberRows);

if (numberRows<21 && numberRows>0)

{

//Ввести текст функции

getchar();

for (int i = 0; i < numberRows; i++)

{

gets\_s(textFunction[i]);

}

//Сгенерировать прототип

int hitIndex = generatePrototip(textFunction, numberRows, nameFunction, strPrototip);

//Вывести данные

if (hitIndex == 1)

{

puts(strPrototip);

}

else

{

puts("no solution");

}

}

else

{

puts("invalid input data");

}

return 0;

}

int generatePrototip(char textFunction[][MAX\_NUMBER\_SYMBOLS], int numberRows, char nameFunction[], char strPrototip[])

{

//Убрать комментарии из текста

removeCommentsFromFunctionText;

int indexForFindEntryName = -1;

int header = 0;

char desiredWhiteSpacers[5] = " \t";

//Сгенерировать прототип, пройдясь по всему тексту...

for (int i = 0; i < numberRows; i++)

{

//Определить, является ли строчка прототипом или заголовком

indexForFindEntryName = findEntryName(textFunction[i], nameFunction);

//Если найден прототип, то обрезаем лишние пробелы и помещаем его в результирующий массив

if (indexForFindEntryName == 1)

{

cutPartStr(textFunction[i], strPrototip, ';', desiredWhiteSpacers);

return 1;

}

//Иначе если найден заголовок, то обрезаем лишние пробелы и помещаем его в результирующий массив

else if (indexForFindEntryName == 0)

{

cutPartStr(textFunction[i], strPrototip, ')', desiredWhiteSpacers);

strcat(strPrototip, ";");

header = 1;

}

}

//Если истинный прототип в тексте не найден, возвращаем сгенерированный прототип

if (header == 1)

{

return 1;

}

//Иначе ничего не возвращаем

else

{

return 0;

}

}

int findEntryName(char textFunction[], char nameFunction[])

{

//Проверка на наличие имени функции в строке...

if ((strstr(textFunction, nameFunction) != 0) && (strstr(textFunction, "=") == 0))

{

char\* pointerFirstWord;

char wordBreakingStr[MAX\_NUMBER\_SYMBOLS];

char strSearchSpecialChars[MAX\_NUMBER\_SYMBOLS];

strcpy(wordBreakingStr, textFunction);

strcpy(strSearchSpecialChars, textFunction);

pointerFirstWord = strtok(wordBreakingStr, " \t");

//Наличие базовых типов возвращаемого значения...

for (int i = 0; i < 5; i++)

{

bool indexFirstSpecialWord = strcmp(pointerFirstWord, returnTypes[i]);

//Проверка формата строки...

if (!indexFirstSpecialWord)

{

char\* pointerNameFunction = strtok(NULL, " (\t");

//Если в строке есть имя функции...

if (strcmp(pointerNameFunction, nameFunction) == 0)

{

//Если в строке есть открывающая и закрывающая круглые скобки...

if (strstr(strSearchSpecialChars, "(") && strstr(strSearchSpecialChars, ")"))

{

//Если в строке присутствует точка с запятой...

if (strstr(strSearchSpecialChars, ";"))

{

return 1;

}

return 0;

}

}

}

}

//Наличие модификаторов для базовых типов...

for (int i = 5; i < 9; i++)

{

bool indexFirstSpecialWord = strcmp(pointerFirstWord, returnTypes[i]);

//Проверка формата строки...

if (!indexFirstSpecialWord)

{

char\* pointerSecondWord = strtok(NULL, " (\t");

char\* pointerThirdWord = strtok(NULL, " (\t");

char\* pointerFourthWord = strtok(NULL, " (\t");

//Если в строке есть имя функции...

if (strcmp(pointerSecondWord, nameFunction) == 0 || strcmp(pointerThirdWord, nameFunction) == 0 || strcmp(pointerFourthWord, nameFunction) == 0)

{

//Если в строке есть открывающая и закрывающая круглые скобки...

if (strstr(strSearchSpecialChars, "(") && strstr(strSearchSpecialChars, ")"))

{

//Если в строке присутствует точка с запятой...

if (strstr(strSearchSpecialChars, ";"))

{

return 1;

}

return 0;

}

}

}

}

return -1;

}

//Иначе не найдено

else

{

return -1;

}

}

void removeCommentsFromFunctionText(char textFunction[][MAX\_NUMBER\_SYMBOLS], int numberRows) //Текст функции слишком тривиальный, чтобы создавать тесты

{

for (int a = 0; a < numberRows; a++)

{

strtok(textFunction[a], "//");

}

}

void cutPartStr(char str[], char partStr[], char finishSymbol, char startStr[])

{

//Находим указатель на последний символ части строки

char\* finishPrototip = strrchr(str, finishSymbol);

//Находим указатель на первый символ части строки

int indexStart = strspn(str, startStr);

char\* startPrototip = &str[indexStart];

//Помещаем часть строки в результирующий массив

int lengthPrototip = finishPrototip - startPrototip + 1;

memcpy(partStr, startPrototip, lengthPrototip);

partStr[lengthPrototip] = NULL;

}

Header.h

#pragma once

//!Максимальное количество строк в функции

int const MAX\_NUMBER\_ROWS = 20;

//!Максимальное количество символов в строке функции

int const MAX\_NUMBER\_SYMBOLS = 500;

//!Список допустимых возвращаемых значений

char const returnTypes[10][10] = { "int", "void", "char", "float", "double", "unsigned", "signed", "short", "long"};

/\*!Главная функция, которая генерирует прототип

\ param [in] textFunction - текст функции

\ param [in] nameFunction - название функции

\ param [in] numberRows - число строк в тексте функции

\ param [out] strPrototip - строка с прототипом

\*/

int generatePrototip (char textFunction[][MAX\_NUMBER\_SYMBOLS], int numberRows, char nameFunction[], char strPrototip[]);

/\*Возвращает 1, прототип сгенерирован\найден;

Возвращает 0, прототип не найден \*/

/\*!Функция, которая определяет вид строки

\ param [in] textFunction - строка

\ param [in] nameFunction - название функции

\*/

int findEntryName (char textFunction[], char nameFunction[]);

/\*Возвращает 1 - прототип;

Возвращает 0 - заголовок;

Возвращает -1 - не найдено\*/

/\*!Функция, которая убирает комментарии из всего текста

\ param[in] textFunction - текст функции

\ param[in] numberRows - число строк в тексте функции \*/

void removeCommentsFromFunctionText(char textFunction[][MAX\_NUMBER\_SYMBOLS], int numberRows);

/\*!Функция, которая вырезает часть строки

\ param [in] str - строка, из которой происходит вырезка

\ param [out] partStr - строка с вырезкой

\ param [in] startStr - строка с символами, которые нужно игнорировать при вырезке перед нужной частью

\ param [in] finishSymbol - символ, до которого нужно вырезать часть строки

\*/

void cutPartStr(char str[], char partStr[], char finishSymbol, char startStr[]);

**Приложение Б. Трасса тестирования и отладки**

Реализованы функции:

generatePrototip, findEntryName, removeCommentsFromFunctionText, cutPartStr

Использованы заглушки:

// Пустая заглушка

int generatePrototip(char textFunction[][MAX\_NUMBER\_SYMBOLS], int numberRows, char nameFunction[], char strPrototip[])

{

return 1;

}

// Пустая заглушка

int findEntryName (char textFunction[], char nameFunction[])

{

return 1;

}

// Пустая заглушка

void removeCommentsFromFunctionText(char textFunction[][MAX\_NUMBER\_SYMBOLS], int numberRows)

{

}

// Пустая заглушка

void cutPartStr(char str[], char partStr[], char finishSymbol, char startStr[])

{

}

***Синтаксические ошибки НЕ обнаружены.***

Реализованы функции:

generatePrototip, findEntryName, removeCommentsFromFunctionText, cutPartStr

Использованы заглушки:

// Заглушка, формирующая выходные данные вручную

int generatePrototip(char textFunction[][MAX\_NUMBER\_SYMBOLS], int numberRows, char nameFunction[], char strPrototip[])

{

int index;

return index;

}

// Заглушка, формирующая выходные данные вручную

int findEntryName (char textFunction[], char nameFunction[])

{

int index;

return index;

}

// Заглушка, формирующая выходные данные вручную

void removeCommentsFromFunctionText(char textFunction[][MAX\_NUMBER\_SYMBOLS], int numberRows)

{

}

// Заглушка, формирующая выходные данные вручную

void cutPartStr(char str[], char partStr[], char finishSymbol, char startStr[])

{

}

***Синтаксические ошибки НЕ обнаружены.***

Реализованы функции:

generatePrototip, findEntryName, removeCommentsFromFunctionText, cutPartStr

Обнаружены ошибки в generatePrototip:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Способ обнаружения/проявления ошибки | Предполагаемая причина ошибки | Способ локализации ошибки | Причина ошибки | Решение |
| Сообщение компилятора: аргумент типа "const char \*" несовместим с параметром типа "char", строка 77. | Неверный параметр передан в функцию. | Посмотреть на прототип функции. | Передан неверный параметр. Неправильные скобки | ";" > ';' |
| Тест tab\_before\_prototip не пройден. | - | Отладка программы по шагам. | Нет в белых разделителях знака табуляции. | char desiredWhiteSpacers[5] = " \t"; |
| Тесты не max\_size\_rows и min\_size\_rows пройдены | - | Отладка теста min\_size\_rows | Неверное сравнение в цикле. | for (int i = 0; i <= numberRows; i++) |
| Все тесты пройдены. | | | | |

Обнаружены ошибки в findEntryName:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Способ обнаружения/проявления ошибки | Предполагаемая причина ошибки | Способ локализации ошибки | Причина ошибки | Решение |
| Сообщение компилятора: "indexFirstSpecialWord" не определен, 120 строка. | - | Внимательно посмотреть код выше. | Нет открывающей фигурной скобки. | for (int i=0; i < 5; i++)  { |
| Функция выдает только -1. | - | Отладка программы по шагам. | Функция strtok разрушает строку. Нужно создать еще один массив, чтобы разрушать уже его, а не настоящую строку. | char wordBreakingStr[MAX\_NUMBER\_SYMBOLS];  strcpy(wordBreakingStr, textFunction); |
| Тест prototip не пройден. | Ошибка в тесте. | Отладка теста. | Неверное нужное возвращаемое значение. | int trueExemple = 1; |
| Тест longest\_prototip не пройден. | - | Отладка программы по шагам. | Не учитывается, что есть возвращаемое значение, состоящее из 3х слов. | if(strcmp(pointerSecondWord, nameFunction) == 0 || strcmp(pointerThirdWord, nameFunction) == 0 || strcmp(pointerFourthWord, nameFunction) == 0) |
| Все тесты пройдены. | | | | |

Обнаружены ошибки в cutPartStr:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Способ обнаружения/проявления ошибки | Предполагаемая причина ошибки | Способ локализации ошибки | Причина ошибки | Решение |
| Сообщение компилятора: значение типа "char" нельзя использовать для инициализации сущности типа "char \*, 194 строка. | - | Посмотреть внимательно на строчку. | Указателю присвоили значение символа, а не адреса. | char\* startPrototip = &str[indexStart]; |
| Функция некорректно работает. Длина не соответствует действительности. | - | Отладка по шагам. | Неверное вычисление длины. | int lengthPrototip = finishPrototip - startPrototip + 1; |
| Тест spaces\_before\_str не пройден. | - | Отладка теста по шагам. | Поменяны местами входящие параметры. | cutPartStr(str, partStr, finishSymbol, startStr); |
| Все тесты пройдены. | | | | |