Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет: “Информационные технологии и прикладная математика”
Кафедра: 806 “Вычислительная математика и программирование”

Курсовая работа
по курсу “Операционные системы”

Текстовый препроцессор Си/C++

Студент	Полей-Добронравова А.В.
Группа	М8О-307Б-18
Преподаватель	Миронов Е.С.
Дата	
Оценка	

Москва, 2020

Содержание

Постановка задачи

Разработать текстовый препроцессор Си/C++, используя memory-mapping, со следующими функциями:

1) Удаление комментариев во входном тексте программы (//, /**/).
2) Директива #include. Вставляет текст программы с названием заключенным между <>, “” в общий обрабатываемый файл.
3) Директива #define. Определяет замены макросов и заменяет их в последующем тексте входной программы.
4) Директива #undef. Отменяет определение макроса.
5) Директива #line. Изменяет нумерацию строк и имя файла.
6) Директива #error. Выводит ошибку и прекращает препроцессинг программы.
7) Директивы #if, #ifdef, #ifndef, #else, #elif, #endif. Устанавливают, какой код программы подвергается препроцессингу.
8) Предупреждения об ошибке в директивах с выводом строки с ошибкой, прекращение препроцессинга.

Общий метод и алгоритм решения

Имя файла, который будет редактироваться, указывается первым аргументом при запуске программы. При помощи функции **open** файл открывается, затем, получаем размер файла через функцию **getFileSize()** использующую **stat**, файл отображается в виртуальную память процесса при помощи системного вызова **mmap**. Создается файл, в который записывается результат препроцессинга **creat("lprework.cpp", 0666).**

Чтение входного файла посимвольное, есть счетчик текущей позиции в файле **pos.** Пока счетчик меньше размера файла, происходит анализ текущего символа. Вся работа программы построена на конечном автомате из 6 состояний. Вызов состояний по номерам:

-1. Обработка символов не для препроцессора, отслеживание начала команд для препроцессора начинающихся с “#”. Символы записываются в буфер до встречи пробела или переноса строки во входном файле, если длина слова превышает 30 символов, вызывается исключение с

завершением программы. Если слово является макросом, определенным с помощью #define, происходит его замена до тех пор, пока заменяемое слово не на что заменить. Если встретился символ “/”, подозрение на комментарий, переход в состояние 3.

-2. Ожидание директивы после “#”. Пропускаются пробелы; если встретился перенос строки, пустая директива, вернуться к состоянию -1. Если символ не пробел, не перенос строки, но буква английского алфавита, начать формировать последовательность символов команды **com,** перейти в состояние 0 формирования директивы. Если ничто из вышеперечисленного, ожидание конца строки в состоянии 1.

0. Если длина **com** строки команды больше 8, ошибка в написании директивы, состояние 1. Если встретился пробел, перейти в состояние 5 формирования слова после директивы (в моем препроцессоре все команды директив располагаются на одной строке кроме условных директив, иначе ошибка). Если встретился перенос строки, если сформированная команда **com** это else или endif, сделать обработку директивы, иначе перейти в состояние 5.

1. Ожидание символа переноса строки с пропуском остальных символов.

-3. После команд define и undef и слов после них, если встретился перенос строки, то правильно написанные директивы, выполняем их, переход в состояние -1. Если встретился пробел, то читаем дальше символы. Иначе неправильно написанная директива, очистка буферов и переход в состояние 1.

3. Если текущий символ “/”, значит встретилась подряд комбинация “//”, нужно пропустить все символы до конца строки. Выводим в выходной файл “\n”, переходим в состояние 1. Если текущий символ “*”, значит встретилась подряд комбинация “/*”, переходим в состояние 2 с ожиданием “*”.

2. Если текущий символ “*”, переходим в состояние 4, иначе ничего.

4. Если текущий символ “/”, конец комментария, переходим в состояние -1. Иначе возвращаемся к ожиданию “*” в состояние 2.

5. Чтение слова до переноса строки или пробела.

6. Состояние в котором происходит работа команд директив. **flag1** регулирует работу таким образом, что условная директива может его изменять при успешном или неуспешном выполнении условия. Если **flag1**

равен “+”, то выполняем всё кроме else, elif. Если **flag1** равен “-”, значит до этого была обработана директива if с неуспехом, и нужно проверить остальные проверки else и elif. При директиве endif в любом случае flag1 становится “+”. Кроме того, если **flag1** не равен “+”, не происходит запись в выходной файл символов (не для препроцессора).

Далее стоит описать реализацию работы директив препроцессора в состоянии 6. Большая часть работы заключается в считывании и анализе корректного вызова директивы.

Для директивы **define** создан класс словаря Define с необходимыми функциями: **define_insert(char* k1, char* k2)** вставляет в словарь новую пару, **define_check(char* k1)** проверяет наличие макроса k1 в словаре, **define_delete(char* k1)** удаляет из словаря макрос k1, **define_second (char* k1)** возвращает строку заменяющую макрос, **define_cout()** выводит словарь при необходимости дебага. Работа директивы заключается в использовании этого класса и функций класса в соответствии с классическим препроцессором.

Undef проверяет наличие макроса в словаре и при его обнаружении удаляет из словаря.

Include обрабатывает все случаи указания названий включаемых файлов: в <>, в “”, с различными отступами внутри этих кавычек. При указании файла в <> открывается директория с системными библиотеками на macOS (/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/include/c++/v1) для поиска нужной библиотеки с помощью **opendir("path");** Далее читается всё содержимое каталога **entry = readdir(dir);** Если имя считанного объекта **entry->d_name** совпадает с искомым, открывается этот файл и копируется в выходной, так же с использованием memory map. Каталог закрывается с помощью **closedir(dir);**
При указании файла в “” в начале идет поиск этого файла в текущем каталоге (**opendir(".")**), а затем так же в системном.

Директива **If** считывает далее файл до символа переноса строки и далее анализирует считанную строку. Она успешно обрабатывает команды:
if defined(MACROS) //без пробелов

if !defined(MACROS) //без пробелов

if defined MACROS

if !defined MACROS

if MACROS > int

if MACROS == int

if MACROS < int

И сообщает об ошибке при других вариантах обращения к препроцессору.

Elif работает совершенно так же, но при значении flag1 равном “-”.

Ifdef, Ifndef работают так же как **if defined(MACROS)** и **if !defined(MACROS)** соответственно.

Else и **Endif** меняет **flag1** на “+” для выполнения последующего кода.

Line меняет нумерацию строк в файле, нумерация строк записывается в переменную **_LINE_**. Имя файла так же можно поменять, оно во-первых изменится в переменной **_FILE_** , во вторых, изменится на самом компьютере с помощью **rename(_FILE_, newname);** Но до изменения имени файла происходит проверка текущего каталога на наличие файлов с таким же названием. Поиск идёт так же, как и в работе директивы **include**.

При любой ошибке происходит освобождение всей занятой памяти, выводится сообщение об ошибке с текущей строкой, закрываются все открытые файлы.

Листинг программы

```
// preprocessor for language C++
// Полей-Добронравова Амелия

#include <iostream>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/uio.h>
#include <sys/mman.h>
#include <unistd.h>
#include <map>
#include <string>
#include <dirent.h>
```

```
#include <sys/stat.h>

class Define { //хранит все пары для замены в тексте файла
   std::map<char*, char*> d;
public:
   void define_insert(char* k1, char* k2) {
      char* p1 = new char[sizeof(k1)];
      for (int j = 0; j <= sizeof(k1); j++) {
         p1[j] = k1[j];
      }
      char* p2 = new char[sizeof(k2)];
      for (int j = 0; j <= sizeof(k2); j++) {
         p2[j] = k2[j];
      }
      d[p1] = p2;
   }
   bool define_check(char* k1) {
      std::map<char*, char*> :: iterator it = d.begin();
      for (; it != d.end(); it++) {  // выводим их
         if (!strcmp(it->first, k1)) {
            return true;
         }
      }
      return false;
   }
   void define_delete(char* k1) {
      std::map<char*, char*> :: iterator it = d.begin();
      for (; it != d.end(); it++) {  // выводим их
         if (!strcmp(it->first, k1)) {
            break;
         }
      }
      if (it != d.end()) {
         delete [] it->first;
         delete [] it->second;
         d.erase(it);
      }
   }
   void define_change(char* k1, char* k2) {
      std::map<char*, char*> :: iterator it = d.begin();
      for (; it != d.end(); it++) {  // выводим их
         if (!strcmp(it->first, k1)) {
            it->second = k2;
            break;
         }
      }
      if (it == d.end()) {
         std::cout << "No pair to change\n";
      }
   }
   char* define_second (char* k1) {
      std::map<char*, char*> :: iterator it = d.begin();
      for (; it != d.end(); it++) {  // выводим их
         if (!strcmp(it->first, k1)) {
```

```
            return it->second;
        }
    }
    if (it == d.end()) {
        std::cout << "No pair to change\n";
    }
    return NULL;
}
void define_cout () {
    std::map<char*, char*> :: iterator it = d.begin();
    for (int k = 0; it != d.end(); it++, k++) {  // выводим их
        std::cout << k << ") Ключ " << it->first << ", значение " << it->second << std::endl;
    }
    std::cout << "\n";
}
~Define() {
    std::map<char*, char*> :: iterator it = d.begin();
    for (; it != d.end(); it++) {  // выводим их
        delete [] it->first;
        delete [] it->second;
    }
}

};

struct FileMapping {
    int fl;
    size_t fsize;
    char* dataPtr;
};

ssize_t getFileSize(const char * fileName)
{
    struct stat file_stat;
    stat(fileName, &file_stat);
    return file_stat.st_size;
}

int main(int argc, const char * argv[]) { //ввод имя файла
    Define d;
    int _LINE_ = 1;
    struct dirent* entry;
    int result;
    DIR *dir = NULL;
    char* _FILE_ = new char[sizeof(&argv[1])];
    char com[8] = {'\0'};
    com[0] = '\0';
    for (int j = 0; j <= strlen(argv[1]); j++) {
        _FILE_[j] = argv[1][j];
    }
    int fp, fl, lib, state = -1, i = 0, i1 = 0;
    ssize_t n = 1;
    char c = '\0', b = '\0';
    struct FileMapping g;
```

```
    struct FileMapping l; // for includes
    char buf[255] = {'\0'};
    int bu = 0;
    char path_include[] =
"/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/include/c++/v1";
    char dop[30] = {'\0'};
    char dop1[30] = {'\0'};
    char* uk = NULL;
    dop[0] = '\0';
    dop1[0] = '\0';
    char flag = ' ', flag1 = '+'; //for include
    if ((fp = open(argv[1], O_RDONLY)) < 0) {
        printf("Cannot open file.\n");
        exit(1);
    }
    g.fl = fp;
    g.fsize = getFileSize(argv[1]);
    g.dataPtr = (char*)mmap(NULL, g.fsize, PROT_READ, MAP_SHARED, g.fl, 0); //Создаем отображение файла в
память
    // PROT_READ страницы могут быть прочитаны
    // PROT_WRITE стр могут быть описаны
    //  MAP_SHARED стр могут сипользоваться совместно с др процессами, которые также проектируют этот
объект в память
    if (g.dataPtr == MAP_FAILED)
        // при ошибке возвращается значение MAP_FAILED
    {
        perror("Map");
        printf("FileMappingCreate - open failed, fname = %s \n", argv[1]);
        close(g.fl);
        exit(-1);
    }
    if ((fl = creat("lprework.cpp", 0666)) == -1) { //буфер для редактирования
        printf("Cannot create file.\n");
        exit(1);
    }
    /* список состояний:
     -3 после define + комбинация
     -2 пропускаю пробелы до директивы
     -1 обработка не символов для препроцессора
     0 обработка директивы
     1 удаляем комментарии пока встречаемый символ не \0
     2 удаляем комментарии пока не встретится * /
     3 подозрение на коментарий
     4 подозрение на закрытие комментария
     5 после директивы
     6 чтение лексем после директивы
     7 после include <
     8 после include "
     */
    c = ' ';
    ssize_t pos = 0, pos1;
    while (pos < g.fsize) {
        c = g.dataPtr[pos];
        pos++;
```

```
if (c == '\n') {
    _LINE_++;
}
switch (state)
{
    case -1:
        if (c == '#') {
            state = -2;
            i = 0;
        }
        else if (c == '/') {
            state = 3;
        }
        else if ((c == ' ' || c == '\n' || c == '\0') && flag1 == '+') {
            dop[i] = '\0';
            while (d.define_check(dop)) {
                strcpy(dop, d.define_second(dop)); //нашли окончательную замену макроса на число в строковом виде
            }
            for (i = 0; i < strlen(dop); i++) {
                if (write(fl, &dop[i], 1) != 1)
                    printf("Error in writing in. line: %d\n", _LINE_);
            }
            if (write(fl, &c, 1) != 1)
                printf("Error in writing in. line: %d\n", _LINE_);
            for (i = 0; i <= strlen(dop); i++) {
                dop[i] = '\0';
            }
            i = 0;
        }
        else if (flag1 == '+') {
            if (i < 30) {
                dop[i] = c;
                i++;
            }
            else {
                std::cout << "Lexem is too big. line: " << _LINE_ << "\n";
                munmap(g.dataPtr, g.fsize);
                d.~Define();
                delete [] _FILE_;
                close(fp);
                close(fl);
                //return 0;
                exit(0);
            }
        }
        break;
    case -2:
        if (c == '\n') {
            state = -1;
        }
        else if (c == ' ') {

        }
        else if ((c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z')) {
```

```
            com[i] = c;
            i++;
            state = 0;
        }
        else {
            state = 1;
        }
        break;
    case 0:
        if (i == 8) {
            printf("Error in directive. line: %d\n", _LINE_);
            for (int j = 0; j <= i;j++) {
                com[j] = '\0';
            }
            i = 0;
            state = 1;
        }
        if (c == ' ') {
            com[i] = '\0';
            i = 0;
            state = 5;
        }
        else if (c == '\n') {
            com[i] = '\0';
            i = 0;
            if (!strcmp(com,"else")) {
                if (flag1 == '-') {
                    flag1 = '+';
                }
                else {
                    flag1 = '-';
                }
                for (int j = 0; j <= 5;j++) {
                    com[j] = '\0';
                }
                state = -1;
            }
            else if (!strcmp(com,"endif")) {
                flag1 = '+';
                for (int j = 0; j <= 5;j++) {
                    com[j] = '\0';
                }
                state = -1;
            }
            else {
                state = 5;
            }
        }
        else if ((c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z')) {
            com[i] = c;
            i++;
        }
        else {
            printf("Error in directive. line: %d\n", _LINE_);
```

```
        for (int j = 0; j <= i;j++) {
            com[j] = '\0';
        }
        i = 0;
        state = 1; //ожидание конца строки
    }
    break;
case 1:
    if (c == '\n') {
        state = -1;
    }
    break;
case -3: //after define + combination
    if (c == '\n') {
        state = -1;
        if (!strcmp(com,"define")) {
            if (!d.define_check(dop1)) {
                d.define_insert(dop1, dop);
            }
            for (int j = 0; j <= i1;j++) {
                dop1[j] = '\0';
            }
            for (int j = 0; j <= i;j++) {
                dop[j] = '\0';
            }
            i = 0;
            i1 = 0;
            for (int j = 0; j <= 7;j++) {
                com[j] = '\0';
            }
        }
        else if (!strcmp(com,"undef")) {
            if (d.define_check(dop)) {
                d.define_delete(dop);
            }
            for (int j = 0; j <= i;j++) {
                dop[j] = '\0';
            }
            i = 0;
            for (int j = 0; j <= 6;j++) {
                com[j] = '\0';
            }
        }
    }
    else if (c == ' ') {

    }
    else {
        state = 1;
        if (!strcmp(com,"define")) {
            for (int j = 0; j <= i1;j++) {
                dop1[j] = '\0';
            }
            for (int j = 0; j <= i;j++) {
```

```
            dop[j] = '\0';
          }
          i = 0;
          i1 = 0;
          for (int j = 0; j <= 7;j++) {
            com[j] = '\0';
          }
        }
        else if (!strcmp(com,"undef")) {
          for (int j = 0; j <= strlen(dop);j++) {
            dop[j] = '\0';
          }
          i = 0;
          for (int j = 0; j <= 6;j++) {
            com[j] = '\0';
          }
        }
      }
      break;
    case 2:
      if (c == '*') {
        state = 4;
      }
      break;
    case 3:
      if (c == '/') {
        c = '\n';
        if (write(fl, &c, 1) != 1)
          printf("Error in writing in. line: %d\n", _LINE_);
        state = 1;
      }
      else if (c == '*') {
        state = 2;
      }
      else {
        b = '/';
        if (write(fl, &b, 1) != 1)
          printf("Error in writing in. line: %d\n", _LINE_);
        if (write(fl, &c, 1) != 1)
          printf("Error in writing in. line: %d\n", _LINE_);
        state = -1;
      }
      break;
    case 4:
      if (c == '/') {
        state = -1;
      }
      else {
        state = 2;
      }
      break;
    case 5: //after directive
      if (c == '\n') {
        if (!strcmp(com,"error")) {
```

```
    std::cout << "\n";
    munmap(g.dataPtr, g.fsize);
    delete [] _FILE_;
    close(fp);
    close(fl);
    break;
}
else if ((!strcmp(com,"line"))) {
    for (int y = 0; y < i1; y++) {
        if (dop1[y] < '0' || dop1[y] > '9') {
            flag = 'f';
            break;
        }
    }
    if (flag == 'f') {
        std::cout << "Error in #line. line: " << _LINE_ <<"\n\n";
        flag = ' ';
    }
    else {
        _LINE_ = 0;
        for (int y = 0; y < i1; y++) {
            _LINE_ = _LINE_ * 10 + dop1[y] - '0';
        }
        std::cout << "CHANGED LINE: " << _LINE_ << "\n";
        state = -1;
        for (int y = 0; y <= i1; y++) {
            dop1[y] ='\0';
        }
        i1 = 0;
    }
}
else if (!strcmp(com,"endif")) {
    flag1 = '+';
    for (int j = 0; j <= 6;j++) {
        com[j] = '\0';
    }
    for (int p = 0; p <= i; p++) {
        dop[p] = '\0';
    }
    i = 0;
}
state = -1;
for (int p = 0; p <= i; p++) {
    dop[p] = '\0';
}
for (int p = 0; p <= i1; p++) {
    dop1[p] = '\0';
}
i = 0;
i1 = 0;
for (int j = 0; j <= 8;j++) {
    com[j] = '\0';
}
flag = ' ';
```

```
    }
    else if (c == ' ') {

    }
    else {
        i = 0;
        dop[i] = c;
        i++;
        state = 6;
    }
    break;
case 6: //read lexeme, what next?
    if (c == '\n' || c == ' ') {
        dop[i] = '\0';
        if (!strcmp(com,"define") && flag1 == '+') { //без пробелов в именах лексем!!!!
            if (dop1[0] == '\0') {
                if (c == '\n') {
                    dop[0] = '\0';
                    state = -1;
                }
                else {
                    for (int j = 0; j <= i;j++) {
                        dop1[j] = dop[j];
                    }
                    for (int j = 0; j <= i;j++) {
                        dop[j] = '\0';
                    }
                    i1 = i;
                    i = 0;
                    state = 5;
                }
            }
            else {
                if (c == '\n') {
                    state = -1;
                    if (!d.define_check(dop1)) {
                        d.define_insert(dop1, dop);
                    }
                    for (int j = 0; j <= i1;j++) {
                        dop1[j] = '\0';
                    }
                    for (int j = 0; j <= i;j++) {
                        dop[j] = '\0';
                    }
                    for (int j = 0; j <= 7;j++) {
                        com[j] = '\0';
                    }
                    i = 0;
                    i1 = 0;
                }
                else {
                    state = -3;
                }
            }
```

```
}
else if (!strcmp(com,"undef") && flag1 == '+') {
   if (c == '\n') {
      if (d.define_check(dop)) {
         d.define_delete(dop);
      }
      for (int j = 0; j <= i;j++) {
         dop[j] = '\0';
      }
      i = 0;
      for (int j = 0; j <= 6;j++) {
         com[j] = '\0';
      }
      state = -1;
   }
   else {
      state = -3;
   }
}
else if (!strcmp(com,"include") && flag1 == '+') {
   if (flag == ' ') {
      if (!strcmp(dop,"<") || !strcmp(dop,"\"")) {
         flag = dop[0];
         state = 5;
         dop[0] = '\0';
         i = 0;
      }
      else if (dop[0] == '<') {
         if (dop[i - 1] == '>') {
            //поиск <>
            dop[i - 1] = '\0';
            for (i = 0; i < strlen(dop); i++) {
               dop1[i] = dop[i+1];
            }
            dir = opendir(path_include);
            if( dir == NULL ) {
               printf( "Error opening dir. line: %d\n", _LINE_);
            }
            entry = readdir(dir);
            while (entry != NULL) {
               if (!strcmp(entry->d_name, dop1)) {
                  char* path = new char[strlen(path_include) + 30 + 1];
                  flag = '+';
                  strcat(path, path_include);
                  strcat(path, "/");
                  strcat(path, dop1);
                  if ((lib = open(path, O_RDONLY)) < 0) {
                     printf("Cannot open file. line: %d\n", _LINE_);
                     exit(1);
                  }
                  l.fl = lib;
                  l.fsize = getFileSize(path);
                  l.dataPtr = (char*)mmap(NULL, l.fsize, PROT_READ, MAP_SHARED, l.fl, 0); //Создаем
отображение файла в память
```

```
                        // PROT_READ страницы могут быть прочитаны
                        // PROT_WRITE стр могут быть описаны
                        //  MAP_SHARED стр могут сипользоваться совместно с др процессами, которые
также проектируют этот объект в память
                        if (l.dataPtr == MAP_FAILED)
                           // при ошибке возвращается значение MAP_FAILED
                        {
                           perror("Map");
                           printf("FileMappingCreate - open failed, fname = %s \n", argv[1]);
                           close(l.fl);
                           d.~Define();
                           delete [] _FILE_;
                           close(fp);
                           close(fl);
                           munmap(g.dataPtr, g.fsize);
                           exit(-1);
                        }
                        pos1 = 0;
                        delete[] path;
                        while (pos1 < l.fsize) {
                           c = l.dataPtr[pos1];
                           pos1++;
                           if (write(fl, &c, n) != n)
                              printf("Error in writing in. line: %d\n", _LINE_);
                        }
                        munmap(l.dataPtr, l.fsize);
                        l.fl = -1;
                        close(lib);
                        break;
                     }
                     entry = readdir(dir);
                  }
                  closedir(dir);
                  if (flag != '+') {
                     std::cout << "Error: file " << dop1 << " not found. line:" <<  _LINE_<<"\n";
                  }
                  for (int p = 0; p <= strlen(dop1); p++) {
                     dop1[p] = '\0';
                  }
                  for (int p = 0; p <= strlen(com); p++) {
                     com[p] = '\0';
                  }
                  flag = ' ';
                  state = -1;
               }
               else {
                  flag = dop[0];
                  state = 5;
                  for (int p = 0; p < i; p++) {
                     dop1[p] = dop[p + 1];
                  }
                  i1 = i;
               }
            }
```

```
else if (dop[0] == "") {
  if (dop[i - 1] == "") {
    //поиск ""
    dir = opendir(".");
    if( dir == NULL ) {
      printf( "Error opening dir. line: %d \n", _LINE_);
    }
    dop[i - 1] = '\0';
    for (i = 0; i < strlen(dop); i++) {
      dop1[i] = dop[i+1];
    }
    entry = readdir(dir);
    while (entry != NULL) {
      if (!strcmp(entry->d_name, dop1)) {
        flag = '+';
        if ((lib = open(dop1, O_RDONLY)) < 0) {
          printf("Cannot open file. line: %d\n", _LINE_);
          exit(1);
        }
        l.fl = lib;
        l.fsize = getFileSize(dop1);
        l.dataPtr = (char*)mmap(NULL, l.fsize, PROT_READ, MAP_SHARED, l.fl, 0); //Создаем
отображение файла в память
        // PROT_READ страницы могут быть прочитаны
        // PROT_WRITE стр могут быть описаны
        //  MAP_SHARED стр могут сипользоваться совместно с др процессами, которые
также проектируют этот объект в память
        if (l.dataPtr == MAP_FAILED)
          // при ошибке возвращается значение MAP_FAILED
        {
          perror("Map");
          printf("FileMappingCreate - open failed, fname = %s \n", argv[1]);
          close(l.fl);
          delete [] _FILE_;
          close(fp);
          d.~Define();
          close(fl);
          munmap(g.dataPtr, g.fsize);
          exit(-1);
        }
        pos1 = 0;
        while (pos1 < l.fsize) {
          c = l.dataPtr[pos1];
          pos1++;
          if (write(fl, &c, n) != n)
            printf("Error in writing in. line: %d\n", _LINE_);
        }
        munmap(l.dataPtr, l.fsize);
        l.fl = -1;
        close(lib);
        break;
      }
      entry = readdir(dir);
    }
```

```
                    closedir(dir);
                    if (flag != '+') { //не нашли в текущем каталоге
                        dir = opendir(path_include);
                        if( dir == NULL ) {
                            printf( "Error opening dir. line: %d \n", _LINE_);
                        }
                        entry = readdir(dir);
                        while (entry != NULL) {
                            if (!strcmp(entry->d_name, dop1)) {
                                flag = '+';
                                char* path = new char[strlen(path_include) + 30 + 1];
                                strcat(path, path_include);
                                strcat(path, "/");
                                strcat(path, dop1);
                                if ((lib = open(path, O_RDONLY)) < 0) {
                                    printf("Cannot open file. line: %d\n", _LINE_);
                                    exit(1);
                                }
                                l.fl = lib;
                                l.fsize = getFileSize(path);
                                l.dataPtr = (char*)mmap(NULL, l.fsize, PROT_READ, MAP_SHARED, l.fl, 0);
//Создаем отображение файла в память
                                // PROT_READ страницы могут быть прочитаны
                                // PROT_WRITE стр могут быть описаны
                                //  MAP_SHARED стр могут сипользоваться совместно с др процессами, которые
также проектируют этот объект в память
                                if (l.dataPtr == MAP_FAILED)
                                    // при ошибке возвращается значение MAP_FAILED
                                {
                                    perror("Map");
                                    printf("FileMappingCreate - open failed, fname = %s \n", argv[1]);
                                    close(l.fl);
                                    delete [] _FILE_;
                                    close(fp);
                                    d.~Define();
                                    close(fl);
                                    munmap(g.dataPtr, g.fsize);
                                    exit(-1);
                                }
                                pos1 = 0;
                                while (pos1 < l.fsize) {
                                    c = l.dataPtr[pos1];
                                    pos1++;
                                    if (write(fl, &c, n) != n)
                                        printf("Error in writing in. line: %d\n", _LINE_);
                                }
                                munmap(l.dataPtr, l.fsize);
                                l.fl = -1;
                                delete [] path;
                                close(lib);
                                break;
                            }
                            entry = readdir(dir);
                        }
```

```
            closedir(dir);
            if (flag != '+') {
               std::cout << "Error: file " << dop1 << " not found. line:"<< _LINE_<< "\n";
            }
         }
         for (int p = 0; p <= strlen(dop1); p++) {
            dop1[p] = '\0';
         }
         for (int p = 0; p <= strlen(com); p++) {
            com[p] = '\0';
         }
         flag = ' ';
         state = -1;
      }
      else {
         flag = dop[0];
         state = 5;
         for (int p = 0; p < i; p++) {
            dop1[p] = dop[p + 1];
         }
         i1 = i;
      }
   }
   else {
      for (int j = 0; j <= 8;j++) {
         com[j] = '\0';
      }
      if (c == '\n') {
         state = -1;
      }
      else {
         state = 1;
      }
   }
}
else {
   if (c == '\n' && dop[0] == '\0') {
      for (int j = 0; j <= 8;j++) {
         com[j] = '\0';
      }
      for (int p = 0; p <= strlen(dop1); p++) {
         dop1[p] = '\0';
      }
      i1 = 0;
      flag = ' ';
      state = -1;
   }
   else {
      if (dop[i-1] == flag || dop[0] == flag || (flag == '<' && dop[0] == '>')) {
         if (flag == '"') {
            //поиск "
            dir = opendir(".");
            if( dir == NULL ) {
               printf( "Error opening dir. line: %d \n", _LINE_);
```

```
                }
                entry = readdir(dir);
                while (entry != NULL) {
                    if (!strcmp(entry->d_name, dop1)) {
                        flag = '+';
                        if ((lib = open(dop1, O_RDONLY)) < 0) {
                            printf("Cannot open file. line: %d\n", _LINE_);
                            exit(1);
                        }
                        l.fl = lib;
                        l.fsize = getFileSize(dop1);
                        l.dataPtr = (char*)mmap(NULL, l.fsize, PROT_READ, MAP_SHARED, l.fl, 0);
//Создаем отображение файла в память
                        // PROT_READ страницы могут быть прочитаны
                        // PROT_WRITE стр могут быть описаны
                        //  MAP_SHARED стр могут сипользоваться совместно с др процессами, которые
также проектируют этот объект в память
                        if (l.dataPtr == MAP_FAILED)
                            // при ошибке возвращается значение MAP_FAILED
                        {
                            perror("Map");
                            printf("FileMappingCreate - open failed, fname = %s \n", argv[1]);
                            close(l.fl);
                            delete [] _FILE_;
                            close(fp);
                            d.~Define();
                            close(fl);
                            munmap(g.dataPtr, g.fsize);
                            exit(-1);
                        }
                        pos1 = 0;
                        while (pos1 < l.fsize) {
                            c = l.dataPtr[pos1];
                            pos1++;
                            if (write(fl, &c, n) != n)
                                printf("Error in writing in. line: %d\n", _LINE_);
                        }
                        munmap(l.dataPtr, l.fsize);
                        l.fl = -1;
                        close(lib);
                        break;
                    }
                    entry = readdir(dir);
                }
                closedir(dir);
                if (flag != '+') { //не нашли в текущем каталоге
                    dir = opendir(path_include);
                    if( dir == NULL ) {
                        printf( "Error opening dir. line: %d \n", _LINE_);
                    }
                    entry = readdir(dir);
                    while (entry != NULL) {
                        if (!strcmp(entry->d_name, dop1)) {
                            flag = '+';
```

```
                    char* path = new char[strlen(path_include) + 30 + 1];
                    strcat(path, path_include);
                    strcat(path, "/");
                    strcat(path, dop1);
                    if ((lib = open(path, O_RDONLY)) < 0) {
                        printf("Cannot open file. line: %d\n", _LINE_);
                        exit(1);
                    }
                    l.fl = lib;
                    l.fsize = getFileSize(path);
                    l.dataPtr = (char*)mmap(NULL, l.fsize, PROT_READ, MAP_SHARED, l.fl, 0);
//Создаем отображение файла в память
                    // PROT_READ страницы могут быть прочитаны
                    // PROT_WRITE стр могут быть описаны
                    //  MAP_SHARED стр могут сипользоваться совместно с др процессами, которые
также проектируют этот объект в память
                    if (l.dataPtr == MAP_FAILED)
                        // при ошибке возвращается значение MAP_FAILED
                    {
                        perror("Map");
                        printf("FileMappingCreate - open failed, fname = %s \n", argv[1]);
                        close(l.fl);
                        delete [] _FILE_;
                        close(fp);
                        close(fl);
                        d.~Define();
                        munmap(g.dataPtr, g.fsize);
                        exit(-1);
                    }
                    pos1 = 0;
                    while (pos1 < l.fsize) {
                        c = l.dataPtr[pos1];
                        pos1++;
                        if (write(fl, &c, n) != n)
                            printf("Error in writing in. line: %d\n", _LINE_);
                    }
                    munmap(l.dataPtr, l.fsize);
                    l.fl = -1;
                    delete[] path;
                    close(lib);
                    break;
                }
                entry = readdir(dir);
            }
            closedir(dir);
            if (flag != '+') {
                std::cout << "Error: file " << dop1 << " not found. line: "<< _LINE_<< "\n";
            }
        }
        for (int p = 0; p <= strlen(dop1); p++) {
            dop1[p] = '\0';
        }
        for (int p = 0; p <= strlen(com); p++) {
            com[p] = '\0';
```

```
                }
                flag = ' ';
                state = -1;
            }
            else {
                dir = opendir(path_include);
                if( dir == NULL ) {
                    printf( "Error opening dir. line: %d \n", _LINE_);
                }
                entry = readdir( dir );
                while (entry != NULL) {
                    if (!strcmp(entry->d_name, dop1)) {
                        flag = '+';
                        char* path = new char[strlen(path_include) + 30 + 1];
                        strcat(path, path_include);
                        strcat(path, "/");
                        strcat(path, dop1);
                        if ((lib = open(path, O_RDONLY)) < 0) {
                            printf("Cannot open file. line: %d\n", _LINE_);
                            exit(1);
                        }
                        l.fl = lib;
                        l.fsize = getFileSize(path);
                        l.dataPtr = (char*)mmap(NULL, l.fsize, PROT_READ, MAP_SHARED, l.fl, 0);
//Создаем отображение файла в память
                        // PROT_READ страницы могут быть прочитаны
                        // PROT_WRITE стр могут быть описаны
                        //  MAP_SHARED стр могут сипользоваться совместно с др процессами, которые
также проектируют этот объект в память
                        if (l.dataPtr == MAP_FAILED)
                            // при ошибке возвращается значение MAP_FAILED
                        {
                            perror("Map");
                            printf("FileMappingCreate - open failed, fname = %s \n", argv[1]);
                            close(l.fl);
                            delete [] _FILE_;
                            close(fp);
                            close(fl);
                            d.~Define();
                            munmap(g.dataPtr, g.fsize);
                            exit(-1);
                        }
                        pos1 = 0;
                        while (pos1 < l.fsize) {
                            c = l.dataPtr[pos1];
                            pos1++;
                            if (write(fl, &c, n) != n)
                                printf("Error in writing in. line: %d\n", _LINE_);
                        }
                        munmap(l.dataPtr, l.fsize);
                        l.fl = -1;
                        delete[] path;
                        close(lib);
                        break;
```

```
                }
                entry = readdir( dir );
            }
            closedir( dir );
            if (flag != '+') {
                std::cout << "Error: file " << dop1 << " not found. line: "<< _LINE_<< "\n";
            }
            for (int p = 0; p <= strlen(dop1); p++) {
                dop1[p] = '\0';
            }
            for (int p = 0; p <= strlen(com); p++) {
                com[p] = '\0';
            }
            flag = ' ';
            state = -1;
        }
    }
    else if (dop1[0] == '\0' && c != '\n') {
        for (int p = 0; p <= i; p++) {
            dop1[p] = dop[p];
        }
        i1 = i;
        state = 5;
    }
    else {
        for (int j = 0; j <= 8;j++) {
            com[j] = '\0';
        }
        state = -1;
        for (int p = 0; p <= strlen(dop1); p++) {
            dop1[p] = '\0';
        }
        i1 = 0;
        flag = ' ';
    }
  }
 }
 for (int j = 0; j <= strlen(dop);j++) {
   dop[j] = '\0';
 }
 i = 0;
}
else if (!strcmp(com,"if") && flag1 == '+') { //считаем до конца строки и проанализируем
 bu = 0;
 b = ' ';
 if (c != '\n') {
   while (b != '\n' && pos < g.fsize) {
     b = g.dataPtr[pos];
     pos++;
     buf[bu] = b;
     bu++;
   }
   if (pos == g.fsize) {
     munmap(g.dataPtr, g.fsize);
```

```
            delete [] _FILE_;
            close(fp);
            d.~Define();
            close(fl);
            break;
        }
        buf[bu - 1] = '\0';
    }
    dop[i] = '\0';
    _LINE_++;
    //далее анализируем строку
    uk = strstr(dop, "defined");
    if (uk != NULL) {
        bu = (int)(uk - dop + 7);
        if (dop[uk - dop - 1] == '!') {
            flag = '!';
        }
        if (dop[bu] == '(' && !strcmp(buf,"\0")) { //без пробелов defined(NAME) или !defined(NAME)
            i = 0;
            bu++;
            while (bu < strlen(dop)) {
                if (dop[bu] == ')') {
                    dop1[i] = '\0';
                    break;
                }
                else {
                    dop1[i] = dop[bu];
                    i++;
                    bu++;
                }
            }
            if (dop[bu] == ')') {
                if (flag == '!') {
                    if (!d.define_check(dop1)) {
                        flag1 = '+';
                    }
                    else {
                        flag1 = '-';
                    }
                }
                else {
                    if (!d.define_check(dop1)) {
                        flag1 = '-';
                    }
                    else {
                        flag1 = '+';
                    }
                }
            }
            else {
                flag1 = '-';
            }
        }
        else { // defined NAME или !defined NAME
```

```
                    i = 0;
                    while (i < strlen(buf) && buf[i] != ' ') {
                        dop1[i] = buf[i];
                        i++;
                    }
                    dop1[i] = '\0';
                    for (; i < strlen(buf); i++) {
                        if (buf[i] != ' ') {
                            std::cout << "Error: IF; line:" << _LINE_ <<"\n";
                            munmap(g.dataPtr, g.fsize);
                            delete [] _FILE_;
                            close(fp);
                            d.~Define();
                            close(fl);
                            exit(0);
                        }
                    }
                    if (flag == '!') {
                        if (!d.define_check(dop1)) {
                            flag1 = '+';
                        }
                        else {
                            flag1 = '-';
                        }
                    }
                    else {
                        if (!d.define_check(dop1)) {
                            flag1 = '-';
                        }
                        else {
                            flag1 = '+';
                        }
                    }
                }
            }
            else {
                if (d.define_check(dop)) {
                    while (d.define_check(dop)) {
                        strcpy(dop, d.define_second(dop)); //нашли окончательную замену макроса на число в
строковом виде
                    }
                    i = 0;
                    while (buf[i] == ' ') {
                        i++;
                    }
                    if (buf[i] == '>') { // только if NAME > INT
                        // if NAME < INT
                        // if NAME == INT
                        flag = '>';
                    }
                    else if (buf[i] == '<') {
                        flag = '<';
                    }
                    else if (buf[i] == '=' && buf[i + 1] == '=') {
```

```
    flag = '=';
    i++;
}
else {
    std::cout << "Error: IF; line:" << _LINE_ <<"\n";
    munmap(g.dataPtr, g.fsize);
    delete [] _FILE_;
    close(fp);
    d.~Define();
    close(fl);
    exit(0);
}
i++;
bu = 0;
while (buf[i] == ' ') {
    i++;
}
while (buf[i] != ' ' && i < strlen(buf)) {
    if (buf[i] < '0' || buf[i] > '9') {
        std::cout << "Error: IS'T INT:" << buf[i] << ". line: " <<_LINE_<< "\n";
        munmap(g.dataPtr, g.fsize);
        delete [] _FILE_;
        close(fp);
        d.~Define();
        close(fl);
        exit(0);
    }
    bu = bu * 10 + buf[i] - '0';
    i++;
}
i1 = 0;
for (i = 0; i < strlen(dop); i++) {
    if (dop[i] < '0' || dop[i] > '9') {
        std::cout << "Error: MACROS IS'T INT:" << dop[i] << ". line: " <<_LINE_<<"\n";
        delete [] _FILE_;
        munmap(g.dataPtr, g.fsize);
        close(fp);
        d.~Define();
        close(fl);
        exit(0);
    }
    i1 = i1 * 10 + dop[i] - '0';
}
if (flag == '>') {
    if (i1 > bu) {
        flag1 = '+';
    }
    else {
        flag1 = '-';
    }
}
else if (flag == '<') {
    if (i1 < bu) {
        flag1 = '+';
```

```
            }
            else {
                flag1 = '-';
            }
        }
        else if (flag == '=') {
            if (i1 == bu) {
                flag1 = '+';
            }
            else {
                flag1 = '-';
            }
        }
        else {
            std::cout << "Error: FLAG IS NONE; line:" << _LINE_ <<"\n";
            delete [] _FILE_;
            munmap(g.dataPtr, g.fsize);
            close(fp);
            d.~Define();
            close(fl);
            exit(0);
        }
    }
    else {
        std::cout << "Error: MACROS not defined. line: "<< _LINE_<< "\n";
        delete [] _FILE_;
        munmap(g.dataPtr, g.fsize);
        close(fp);
        d.~Define();
        close(fl);
        exit(0);
    }
}
flag = ' ';
for (int j = 0; j <= 8;j++) {
    com[j] = '\0';
}
for (int p = 0; p <= strlen(dop); p++) {
    dop[p] = '\0';
}
for (int p = 0; p <= strlen(buf); p++) {
    buf[p] = '\0';
}
for (int p = 0; p <= strlen(dop1); p++) {
    dop1[p] = '\0';
}
i = 0;
i1 = 0;
bu = 0;
state = -1;
}
else if (!strcmp(com,"ifdef") && flag1 == '+') {
    if (d.define_check(dop)) {
        flag1 = '+';
```

```
    }
    else {
        flag1 = '-';
    }
    state = -1;
    for (int j = 0; j <= 6;j++) {
        com[j] = '\0';
    }
    for (int p = 0; p <= i; p++) {
        dop[p] = '\0';
    }
    i = 0;
}
else if (!strcmp(com,"ifndef") && flag1 == '+') {
    if (!d.define_check(dop)) {
        flag1 = '+';
    }
    else {
        flag1 = '-';
    }
    state = -1;
    for (int j = 0; j <= 6;j++) {
        com[j] = '\0';
    }
    for (int p = 0; p <= i; p++) {
        dop[p] = '\0';
    }
    i = 0;
}
else if (!strcmp(com,"else") && flag1 == '-') {
    flag1 = '+';
    for (int j = 0; j <= 6;j++) {
        com[j] = '\0';
    }
    for (int p = 0; p <= i; p++) {
        dop[p] = '\0';
    }
    i = 0;
    state = -1;
}
else if (!strcmp(com,"elif") && flag1 == '-') {
    bu = 0;
    b = ' ';
    if (c != '\n') {
        while (b != '\n' && pos < g.fsize) {
            b = g.dataPtr[pos];
            pos++;
            buf[bu] = b;
            bu++;
        }
        if (pos == g.fsize) {
            munmap(g.dataPtr, g.fsize);
            delete [] _FILE_;
            close(fp);
```

```
            d.~Define();
            close(fl);
            break;
        }
        buf[bu - 1] = '\0';
    }
    dop[i] = '\0';
    _LINE_++;
    //далее анализируем строку
    uk = strstr(dop, "defined");
    if (uk != NULL) {
        bu = (int)(uk - dop + 7);
        if (dop[uk - dop - 1] == '!') {
            flag = '!';
        }
        if (dop[bu] == '(' && !strcmp(buf,"\0")) { //без пробелов defined(NAME) или !defined(NAME)
            i = 0;
            bu++;
            while (bu < strlen(dop)) {
                if (dop[bu] == ')') {
                    dop1[i] = '\0';
                    break;
                }
                else {
                    dop1[i] = dop[bu];
                    i++;
                    bu++;
                }
            }
            if (dop[bu] == ')') {
                if (flag == '!') {
                    if (!d.define_check(dop1)) {
                        flag1 = '+';
                    }
                    else {
                        flag1 = '-';
                    }
                }
                else {
                    if (!d.define_check(dop1)) {
                        flag1 = '-';
                    }
                    else {
                        flag1 = '+';
                    }
                }
            }
            else {
                flag1 = '-';
            }
        }
        else { // defined NAME или !defined NAME
            i = 0;
            while (i < strlen(buf) && buf[i] != ' ') {
```

```
                    dop1[i] = buf[i];
                    i++;
                }
                dop1[i] = '\0';
                for (; i < strlen(buf); i++) {
                    if (buf[i] != ' ') {
                        std::cout << "Error: ELIF; line:" << _LINE_ <<"\n";
                        munmap(g.dataPtr, g.fsize);
                        delete [] _FILE_;
                        close(fp);
                        d.~Define();
                        close(fl);
                        exit(0);
                    }
                }
                if (flag == '!') {
                    if (!d.define_check(dop1)) {
                        flag1 = '+';
                    }
                    else {
                        flag1 = '-';
                    }
                }
                else {
                    if (!d.define_check(dop1)) {
                        flag1 = '-';
                    }
                    else {
                        flag1 = '+';
                    }
                }
            }
        }
        else {
            if (d.define_check(dop)) {
                while (d.define_check(dop)) {
                    strcpy(dop, d.define_second(dop)); //нашли окончательную замену макроса на число в
строковом виде
                }
                i = 0;
                while (buf[i] == ' ') {
                    i++;
                }
                if (buf[i] == '>') { // только if NAME > INT
                    // if NAME < INT
                    // if NAME == INT
                    flag = '>';
                }
                else if (buf[i] == '<') {
                    flag = '<';
                }
                else if (buf[i] == '=' && buf[i + 1] == '=') {
                    flag = '=';
                    i++;
```

```
}
else {
    std::cout << "Error: ElIF; line:" << _LINE_ <<"\n";
    munmap(g.dataPtr, g.fsize);
    delete [] _FILE_;
    close(fp);
    d.~Define();
    close(fl);
    exit(0);
}
i++;
bu = 0;
while (buf[i] == ' ') {
    i++;
}
while (buf[i] != ' ' && i < strlen(buf)) {
    if (buf[i] < '0' || buf[i] > '9') {
        std::cout << "Error: IS'T INT:" << buf[i] << ". line: " <<_LINE_<< "\n";
        munmap(g.dataPtr, g.fsize);
        delete [] _FILE_;
        close(fp);
        d.~Define();
        close(fl);
        exit(0);
    }
    bu = bu * 10 + buf[i] - '0';
    i++;
}
i1 = 0;
for (i = 0; i < strlen(dop); i++) {
    if (dop[i] < '0' || dop[i] > '9') {
        std::cout << "Error: MACROS IS'T INT:" << dop[i] << ". line: " <<_LINE_<<"\n";
        delete [] _FILE_;
        munmap(g.dataPtr, g.fsize);
        close(fp);
        d.~Define();
        close(fl);
        exit(0);
    }
    i1 = i1 * 10 + dop[i] - '0';
}
if (flag == '>') {
    if (i1 > bu) {
        flag1 = '+';
    }
    else {
        flag1 = '-';
    }
}
else if (flag == '<') {
    if (i1 < bu) {
        flag1 = '+';
    }
    else {
```

```
                flag1 = '-';
            }
        }
        else if (flag == '=') {
            if (i1 == bu) {
                flag1 = '+';
            }
            else {
                flag1 = '-';
            }
        }
        else {
            std::cout << "Error: FLAG IS NONE; line:" << _LINE_ <<"\n";
            delete [] _FILE_;
            munmap(g.dataPtr, g.fsize);
            close(fp);
            d.~Define();
            close(fl);
            exit(0);
        }
    }
    else {
        std::cout << "Error: MACROS not defined. line: "<< _LINE_<< "\n";
        delete [] _FILE_;
        munmap(g.dataPtr, g.fsize);
        close(fp);
        d.~Define();
        close(fl);
        exit(0);
    }
}
flag = ' ';
for (int j = 0; j <= 8;j++) {
    com[j] = '\0';
}
for (int p = 0; p <= strlen(dop); p++) {
    dop[p] = '\0';
}
for (int p = 0; p <= strlen(buf); p++) {
    buf[p] = '\0';
}
for (int p = 0; p <= strlen(dop1); p++) {
    dop1[p] = '\0';
}
i = 0;
i1 = 0;
bu = 0;
state = -1;
}
else if (!strcmp(com,"endif")) {
flag1 = '+';
for (int j = 0; j <= 6;j++) {
    com[j] = '\0';
}
```

```
    for (int p = 0; p <= i; p++) {
        dop[p] = '\0';
    }
    i = 0;
    state = -1;
}
else if (!strcmp(com,"line") && flag1 == '+') {
    if (dop1[0] == '\0') {
        if (c == '\n') {
            for (int y = 0; y < i; y++) {
                if (dop[y] < '0' || dop[y] > '9') {
                    flag = 'f';
                    break;
                }
            }
            if (flag == 'f') {
                std::cout << "Error in #line. line: " << _LINE_ << "\n";
                flag = ' ';
                for (int y = 0; y <= 5; y++) {
                    com[i] = '\0';
                }
                state = 1;
            }
            else {
                _LINE_ = 0;
                for (int y = 0; y < i; y++) {
                    _LINE_ = _LINE_ * 10 + dop[y] - '0';
                }
                std::cout << "CHANGED LINE: " << _LINE_ << "\n";
                state = -1;
            }
            for (int y = 0; y <= strlen(dop); y++) {
                dop[y] = '\0';
            }
            i = 0;
        }
        else {
            state = 5;
            for (int p = 0; p <= i; p++) {
                dop1[p] = dop[p];
            }
            i1 = i;
            for (int p = 0; p <= i; p++) {
                dop[p] = '\0';
            }
            i = 0;
        }
    }
    else {
        for (int y = 0; y < i1; y++) {
            if (dop1[y] < '0' || dop1[y] > '9') {
                flag = 'f';
                break;
            }
```

```
}
if (flag == 'f') {
   std::cout << "Error in #line. line: " << _LINE_<< "\n";
   flag = ' ';
   for (int y = 0; y <= 5; y++) {
      com[i] = '\0';
   }
   for (int p = 0; p <= i1; p++) {
      dop1[p] = '\0';
   }
   i1 = 0;
   state = 1;
}
else {
   _LINE_ = 0;
   for (int y = 0; y < i1; y++) {
      _LINE_ = _LINE_ * 10 + dop1[y] - '0';
   }
   std::cout << "CHANGED LINE: " << _LINE_ << "\n";
   for (int p = 0; p <= i1; p++) {
      dop1[p] = '\0';
   }
   i1 = 0;
   dir = opendir(".");
   if( dir == NULL ) {
      printf( "Error opening dir. line: %d \n", _LINE_);
   }
   entry = readdir( dir );
   while (entry != NULL) {
      if (!strcmp(entry->d_name, dop)) {
         flag = 'f';
         std::cout << "Error in #line: filename. line: " << _LINE_ <<  "\n";
         break;
      }
      entry = readdir( dir );
   }
   closedir( dir );
   if (flag == ' ') {
      result= rename(_FILE_, dop);
      if ( result == 0 )
         puts ( "File successfully renamed" );
      else
         perror( "Error renaming file" );
      for (int y = 0; y <= i; y++) {
         _FILE_[y] = dop[y];
      }
   }
   flag = ' ';
   for (int j = 0; j <= 5;j++) {
      com[j] = '\0';
   }
   for (int y = 0; y <= strlen(dop); y++) {
      dop[y] = '\0';
   }
```

```
                for (int y = 0; y <= strlen(dop1); y++) {
                    dop1[y] = '\0';
                }
                state = -1;
            }
        }
    }
    else if (!strcmp(com,"error") && flag1 == '+') {
        if (flag == ' ') {
            std::cout << "Fatal error: ";
            flag = 'f';
        }
        std::cout << dop << ". line: " << _LINE_;
        if (c == '\n') {
            std::cout << "\n";
            delete [] _FILE_;
            munmap(g.dataPtr, g.fsize);
            close(fp);
            d.~Define();
            close(fl);
            break;
        }
        else {
            state = 5;
            for (int p = 0; p <= i; p++) {
                dop[p] = '\0';
            }
            i = 0;
        }
    }
    else if (c == ' ') {
        for (int j = 0; j <= 8;j++) {
            com[j] = '\0';
        }
        for (int p = 0; p <= i; p++) {
            dop[p] = '\0';
        }
        i = 0;
        state = 1;
    }
    else if (c == '\n') {
        for (int j = 0; j <= 8;j++) {
            com[j] = '\0';
        }
        for (int p = 0; p <= i; p++) {
            dop[p] = '\0';
        }
        i = 0;
        state = -1;
    }
}
else {
    if (i == 30) {
        printf("Lexem is bigger than 30. line: %d\n", _LINE_);
```

```
                    for (int j = 0; j < i;j++) {
                        dop[j] = '\0';
                    }
                    state = 1;
                }
                else {
                    dop[i] = c;
                    i++;
                }
            }
            break;
        };
    }
    delete [] _FILE_;
    munmap(g.dataPtr, g.fsize);
    close(fp);
    d.~Define();
    close(fl);
    //return 0;
    exit(0);
}
```

Демонстрация работы программы

```
MacBook-Pro-Amelia:lexical_analyzator amelia$ g++ prep.cpp -o kp
MacBook-Pro-Amelia:lexical_analyzator amelia$ ./kp cool1.cpp
Error: file loh1 not found. line:34
Error: file  not found. line: 35
Error: file loh3 not found. line: 36
Error: file  not found. line: 40
CHANGED LINE: 100
CHANGED LINE: 50
File successfully renamed
CHANGED LINE: 40
File successfully renamed
CHANGED LINE: 60
File successfully renamed
Error: file loh14 not found. line: 69
Error: file ELIFisworking not found. line: 77
Error: file loh15 not found. line: 87
```

cool1.cpp

```
if ((fl = creat("lprework.txt", 0666)) == -1) { //буфер для редактирования
      printf("Cannot create file.\n");
      exit(1);
    }
    /* список состояний:
     -1 обработка не символов для препроцессора
```

```
    0 обработка директивы
    1 удаляем комментарии пока встречаемый символ не \0
    2 удаляем комментарии пока не встретится * /
    3 подозрение на коментарий
    4 подозрение на закрытие комментария

    */
   while (1) {
      if((n = read(fp,&c, 1)) > 0) {
         switch (state)
         {
   #   define
#   define ERO 1
#   define ERO101 11
#   define SQRT(X,Y)   X,Y
#   define ERO24 12_O4   11
#   define ERO2  1202
#   define ERO3 13_O2
#   undef
# undef ERO24567
# undef ERO2   45
//# undef ERO2

PSSSSS ERO    ERO ERO3

/ hljb /
#undef
#   include "loh1"
#   include " loh2"
#   include "loh3 "
#   include "knot.h"
#   include " loh8
#   include " loh4"
#   include " loh5>
#   include < string.h >

HEEEEEEEEEEEEEEYYYYY

#  line 100
#  line 50  main.cpp
#  line 40  cool.cpp
#  line 60  cool1.cpp
#   include
#   include <loh9
#   include < loh10 "
#if defined(ERO2
#   include " loh11 "
privet
# else
#   include " loh14 "
andrey
#endif

#if ERO2 < 1000
```

```
#   include " loh200000 "
                privet
#elif ERO2 > 1000
#   include " ELIFisworking "
# else
#   include " loh30000 "
                andrey
#endif

#ifdef ER134
#   include " loh12 "
pr1
# else
#   include " loh15 "
andr1
#endif
// fkfgsthkpwotrkhpokrt /


ko ko

/  a
```

lprework.cpp

```
if ((fl = creat("lprework.txt", 0666)) == -1) {
      printf("Cannot create file.\n");
      exit(1);
   }

   while (1) {
      if((n = read(fp,&c, 1)) > 0) {
         switch (state)
         {


PSSSSS 1    1 13_O2

/ hljb /



#ifndef KNOT_H
#define KNOT_H

#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <errno.h>
```

```
#include <fcntl.h>
#include <unistd.h>

#include "vocabulary.h"
#include "message.h"

int BEGIN_PORT = 8000;
//char* BIND_URL = "tcp://*:";
//char* CON_URL = "tcp://localhost:";

char* BindURLPort(int port);

char* ConURLPort(int port);

int TakePort(void* socket);

typedef struct _n {
        int id;
   void* context_me;
   void* r_me;
   pid_t pid;
   voc* v;
   int port_fl;
   void* context_fl;
   void* r_fl;
} knot;

knot* knot_create(char* id, char* pport);
void knot_add(knot* k, int d);
void knot_destroy(knot* k);
void knot_destroy_brahch(knot* k);

#endif// -*- C++ -*-
//===-------------------------- string.h --------------------------------===//
//
//                     The LLVM Compiler Infrastructure
//
// This file is distributed under the University of Illinois Open Source
// License. See LICENSE.TXT for details.
//
//===--------------------------------------------------------------------===//

#ifndef _LIBCPP_STRING_H
#define _LIBCPP_STRING_H

/*
   string.h synopsis

Macros:

   NULL

Types:
```

```
   size_t

void* memcpy(void* restrict s1, const void* restrict s2, size_t n);
void* memmove(void* s1, const void* s2, size_t n);
char* strcpy (char* restrict s1, const char* restrict s2);
char* strncpy(char* restrict s1, const char* restrict s2, size_t n);
char* strcat (char* restrict s1, const char* restrict s2);
char* strncat(char* restrict s1, const char* restrict s2, size_t n);
int memcmp(const void* s1, const void* s2, size_t n);
int strcmp (const char* s1, const char* s2);
int strncmp(const char* s1, const char* s2, size_t n);
int strcoll(const char* s1, const char* s2);
size_t strxfrm(char* restrict s1, const char* restrict s2, size_t n);
const void* memchr(const void* s, int c, size_t n);
      void* memchr(      void* s, int c, size_t n);
const char* strchr(const char* s, int c);
      char* strchr(      char* s, int c);
size_t strcspn(const char* s1, const char* s2);
const char* strpbrk(const char* s1, const char* s2);
      char* strpbrk(      char* s1, const char* s2);
const char* strrchr(const char* s, int c);
      char* strrchr(      char* s, int c);
size_t strspn(const char* s1, const char* s2);
const char* strstr(const char* s1, const char* s2);
      char* strstr(      char* s1, const char* s2);
char* strtok(char* restrict s1, const char* restrict s2);
void* memset(void* s, int c, size_t n);
char* strerror(int errnum);
size_t strlen(const char* s);

*/

#include <__config>

#if !defined(_LIBCPP_HAS_NO_PRAGMA_SYSTEM_HEADER)
#pragma GCC system_header
#endif

#include_next <string.h>

// MSVCRT, GNU libc and its derivates may already have the correct prototype in
// <string.h>. This macro can be defined by users if their C library provides
// the right signature.
#if defined(__CORRECT_ISO_CPP_STRING_H_PROTO) || defined(_LIBCPP_MSVCRT) || \
    defined(__sun__) || defined(_STRING_H_CPLUSPLUS_98_CONFORMANCE_)
#define _LIBCPP_STRING_H_HAS_CONST_OVERLOADS
#endif

#if defined(__cplusplus) && !defined(_LIBCPP_STRING_H_HAS_CONST_OVERLOADS) &&
defined(_LIBCPP_PREFERRED_OVERLOAD)
extern "C++" {
inline _LIBCPP_INLINE_VISIBILITY
char* __libcpp_strchr(const char* __s, int __c) {return (char*)strchr(__s, __c);}
inline _LIBCPP_INLINE_VISIBILITY _LIBCPP_PREFERRED_OVERLOAD
```

```
const char* strchr(const char* __s, int __c) {return __libcpp_strchr(__s, __c);}
inline _LIBCPP_INLINE_VISIBILITY _LIBCPP_PREFERRED_OVERLOAD
    char* strchr(      char* __s, int __c) {return __libcpp_strchr(__s, __c);}

inline _LIBCPP_INLINE_VISIBILITY
char* __libcpp_strpbrk(const char* __s1, const char* __s2) {return (char*)strpbrk(__s1, __s2);}
inline _LIBCPP_INLINE_VISIBILITY _LIBCPP_PREFERRED_OVERLOAD
const char* strpbrk(const char* __s1, const char* __s2) {return __libcpp_strpbrk(__s1, __s2);}
inline _LIBCPP_INLINE_VISIBILITY _LIBCPP_PREFERRED_OVERLOAD
    char* strpbrk(      char* __s1, const char* __s2) {return __libcpp_strpbrk(__s1, __s2);}

inline _LIBCPP_INLINE_VISIBILITY
char* __libcpp_strrchr(const char* __s, int __c) {return (char*)strrchr(__s, __c);}
inline _LIBCPP_INLINE_VISIBILITY _LIBCPP_PREFERRED_OVERLOAD
const char* strrchr(const char* __s, int __c) {return __libcpp_strrchr(__s, __c);}
inline _LIBCPP_INLINE_VISIBILITY _LIBCPP_PREFERRED_OVERLOAD
    char* strrchr(      char* __s, int __c) {return __libcpp_strrchr(__s, __c);}

inline _LIBCPP_INLINE_VISIBILITY
void* __libcpp_memchr(const void* __s, int __c, size_t __n) {return (void*)memchr(__s, __c, __n);}
inline _LIBCPP_INLINE_VISIBILITY _LIBCPP_PREFERRED_OVERLOAD
const void* memchr(const void* __s, int __c, size_t __n) {return __libcpp_memchr(__s, __c, __n);}
inline _LIBCPP_INLINE_VISIBILITY _LIBCPP_PREFERRED_OVERLOAD
    void* memchr(      void* __s, int __c, size_t __n) {return __libcpp_memchr(__s, __c, __n);}

inline _LIBCPP_INLINE_VISIBILITY
char* __libcpp_strstr(const char* __s1, const char* __s2) {return (char*)strstr(__s1, __s2);}
inline _LIBCPP_INLINE_VISIBILITY _LIBCPP_PREFERRED_OVERLOAD
const char* strstr(const char* __s1, const char* __s2) {return __libcpp_strstr(__s1, __s2);}
inline _LIBCPP_INLINE_VISIBILITY _LIBCPP_PREFERRED_OVERLOAD
    char* strstr(      char* __s1, const char* __s2) {return __libcpp_strstr(__s1, __s2);}
}
#endif

#endif // _LIBCPP_STRING_H
```

HEEEEEEEEEEEEEYYYYY

andrey

andr1

ko ko

/ a

Вывод

Курсовая работа, как и весь курс, показывают, как можно использовать многие полезные утилиты и системные вызовы ОС, как устроены операционные системы в целом, а также где это может применяться.

В данном курсовом проекте я использовала некоторые из уже ранее пройденных в курсе тем: работа с файлами, файловыми дескрипторами и виртуальной памятью. Дополнительно я научилась работать с директориями, открывая их и считывая их поэлементно изнутри программы; переименовывать файлы из программы.

Говоря о доработке программы, можно добавить много нереализованных функций классического препроцессора Си/C++, проверять, был ли включен файл с помощью #**include** ранее (хранить список включенных файлов). Можно проще организовать конечный автомат, считывая у всех директив в буфер до конца строки, как это было сделано с директивой **if.** Также возможно упростить требования к вызову директив, касаемо отступов.
Кроме того, можно переписать препроцессор таким образом, чтобы он в зависимости от текущей операционной системы менял свою обработку, т.к. мой препроцессор годится только для **macOS**, например, потому что поиск библиотек, включаемых с помощью **#include**, осуществляется в системном каталоге мака, пути до системных каталогов других Unix систем мой препроцессор не знает.