Федеральное государственное бюджетное образовательное учреждение высшего образования "Национальный исследовательский университет "МЭИ"

ОТЧЕТ

По лабораторной работе № 4

# «Представление данных в памяти»

ФИО студента Локтюшов Владимир Алексеевич

Номер группы А-03-18

Имя преподавателя Мохов Андрей Сергеевич

Козлюк Дмитрий Александрович

Москва, «НИУ «МЭИ», 2019

Ход работы.

1. Чтобы написать функции для печати отдельных байт и блока данных в шестнадцатеричном и в двоичном представлении, необходимо написать функции печати байтов в двоичном и шестнадцатеричном виде:

char nibble\_to\_hex (uint8\_t i)

{

assert(0x0 <= i && i <= 0xf);

char digits[] = "0123456789abcdef";

return digits[i];

}

char bit\_digit(uint8\_t byte, uint8\_t bit)

{

if (byte & (0x1 << bit))

{

return '1';

}

return '0';

}

1. С помощью полученных функции напишем требуемые:

void print\_in\_binary(uint8\_t byte)

{

for (uint8\_t bit = 7; bit > 0; bit--)

{

cout << bit\_digit(byte, bit);

}

cout << bit\_digit(byte, 0);

}

void print\_in\_binary(const void\* data, size\_t size)

{

const uint8\_t\* bytes = as\_bytes(data);

for (size\_t i = 0; i < size; i++)

{

print\_in\_binary(bytes[i]);

if ((i + 1) % 4 == 0)

{

cout << '\n';

}

else

{

cout << ' ';

}

}

}

void print\_in\_hex (uint8\_t byte)

{ //переводит одну половину потом другую

cout << nibble\_to\_hex(byte >> 4)

<< nibble\_to\_hex(byte & 0xf);

}

const uint8\_t\* as\_bytes(const void\* data)

{

return reinterpret\_cast<const uint8\_t\*>(data);

}

void print\_in\_hex(const void\* data, size\_t size)

{

const uint8\_t\* bytes = as\_bytes(data);

for (size\_t i = 0; i < size; i++)

{

print\_in\_hex(bytes[i]);

if ((i + 1) % 16 == 0)

{

cout << '\n';

}

else

{

cout << ' ';

}

}

}

1. Теперь требуется написать калькулятор побитовых операций. На вход он будет получать численные значения 2 операндов в десятичном виде, и символьное значение оператора. Вывод включает в себя двоичные и шестнадцатеричные формы операндов и результатов операций:

void bitcalc()

{

uint16\_t operand1,operand2,result;

char op; // ii?aaoi?

cin>> operand1>>op>>operand2;

assert(op == '&' || op=='|' || op=='^');

switch (op)

{

case '&':

result = operand1 & operand2;

break;

case '|':

result = operand1 | operand2;

break;

case '^':

result = operand1 ^ operand2;

break;

// default: cout<<"incorrect operator";

}

print\_in\_hex(&operand1, sizeof(operand1));

cout<<op<<" ";

print\_in\_hex(&operand2, sizeof(operand2));

cout<<"= ";

print\_in\_hex(&result, sizeof(result));

cout<<'\n';

print\_in\_binary(&operand1, sizeof(operand1));

cout<<op<<" ";

print\_in\_binary(&operand2, sizeof(operand2));

cout<<"= ";

print\_in\_binary(&result, sizeof(result));

}

В главной функции (main) :

bitcalc();

1. Опишем структуру **Student**, объявим и заполним массив из 2 студентов и их старосты.

Видим, что адрес массива совпадает с адресом первого элемента, а его размер равен сумме размеров каждого элемента, который в свою очередь, предопределен структурой: 17 байт на имя (каждый символ по 2 байта), 2 байта на год (целое беззнаковое), 4 байта на среднюю оценку (с плавающей запятой), 4 байта на адрес переменной старосты.

1. Требуется записать некоторые константы: максимальную длину имени файла, запрещенные символы,разделители, ширину расширения:

const size\_t MAX\_SIZE = 256;

const char\* separators = " \r\n,.!?:;()-";

const char\* forbidden = "\*<>?|\"";

const size\_t MAX\_EXTENSION = 3;

Добавим ввод имени файла, .

char text[MAX\_SIZE];

cout << "Enter file name: ";

fgets(text, MAX\_SIZE, stdin);

char\* namefile = text;

1. Остальную часть, включающую в себя проверки на корректность имени файла, загрузку в память содержимого файла, ввод строки, поиск совпадений строки с содержимым файла, оформим в виде цикла **while**. Так будут выглядеть функции проверки имени файла на корректность:

while(true)

{

if( strcspn( namefile, forbidden ) == strlen( namefile ) ) ;

else

{

cout << "Wrong format\n";

break;

}

if( strchr(namefile, ':' ) == nullptr ) ;

else

{

char\* colon = strchr(namefile,':');

if( colon - namefile + 1 <= 1 )

{

cout << "Wrong format";

break;

}

while( colon != nullptr )

{

if( isalpha( namefile[colon - namefile - 1] ) > 0 &&

namefile[colon - namefile + 1] == '\\')

colon = strchr(colon + 1, ':');

else

{

cout << "Wrong format";

break;

}

}

}

if( strrchr( namefile, '.' ) == nullptr )

{

\*(namefile + strlen(namefile) - 1) = '.';

strcat( namefile, "txt" );

}

else

{

char\* ext = strrchr( namefile, '.' );

for( int i = 1; ext[i] != '\0'; i ++ )

\*(ext + i) = char(tolower(\*(ext + i)));

if( strncmp( ext, ".txt", MAX\_EXTENSION + 1 ) == 0 );

else

{

cout << "Wrong format";

break;

}

}

break;

}

1. Загрузим в память и сохраним в динамическом массиве-строке содержимое файла, определим размер содержимого в байтах. Добавим запрос строки у пользователя.

FILE\* file = fopen(namefile, "r" );

fseek( file, 0, SEEK\_END );

long filesize = ftell( file );

char\* filedata = new char[filesize];

rewind( file );

fread( filedata, 1, filesize, file );

char line[MAX\_SIZE];

cout << "Enter string: ";

fgets(line, MAX\_SIZE, stdin);

const char\* str = line;

1. Подсчитаем количество вхождений введенной строки в файл, выведем его. Также необходимо закрыть файл, откуда копировалось содержимое, и освободить динамически выделенную память. Допишем оператор **break,** совершающий выход из главного цикла, следовательно завершающий работу программы:

while(true)

{

if( strstr( filedata, str ) != nullptr )

cout << strstr( filedata, str );

else

{

cout << "NULL";

break;

}

}

fclose( file );

delete[] filedata;

return 0;

}