МИНОБРНАУКИ РОССИИ

Санкт-Петербургский государственный электротехнический
университет «ЛЭТИ» им. В. И. Ульянова (Ленина)

БАЗОВЫЕ СВЕДЕНИЯ К ВЫПОЛНЕНИЮ КУРСОВОЙ РАБОТЫ ПО ДИСЦИПЛИНЕ «ПРОГРАММИРОВАНИЕ». ВТОРОЙ СЕМЕСТР

Учебно-методическое пособие

Санкт-Петербург
Издательство СПбГЭТУ «ЛЭТИ»
2024

УДК 004.42(07)
ББК З 973.2-018я7
Б17

Авторы: **М. М. Заславский, А. А. Лисс, А. В. Гаврилов, С. А. Глазунов, Я. С. Государкин, С. А. Тиняков, В. П. Голубева, К. В. Чайка, В. Е. Допира.**

Б17 Базовые сведения к выполнению курсовой работы по дисциплине «Программирование». Второй семестр: учеб.-метод. пособие. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2024. 36 с.

ISBN 978-5-7629-3270-7

Содержит необходимый теоретический материал и поясняющие примеры.

Представлены материалы для выполнения курсовой работы по дисциплине «Программирование». Предназначено для использования во втором семестре студентами направлений «Программная инженерия» и «Прикладная математика и информатика».

УДК 004.42(07)
ББК З 973.2-018я7

Рецензент канд. техн. наук Ю. Б. Остапченко (зам. ген. директора по специальной тематике ООО «Альвекс»).

Утверждено
редакционно-издательским советом университета
в качестве учебно-методического пособия

ISBN 978-5-7629-3270-7

ВВЕДЕНИЕ

В рамках курсовой работы предлагается выбрать одну из предложенных тем в соответствие с желаемым уровнем сложности: создание игры при помощи псевдографики и обработка изображений.

Процедура защиты курсовой работы состоит из трех основных этапов: реализация программы в соответствии с заданием курсовой работы, подготовка пояснительной записки и защита курсовой работы.

Реализация программы включает в себя написание файла с исходным кодом программы на языке Си, которая удовлетворяет целям и задачам курсовой работы. Исходный код программы должен быть компилируемым без ошибок и предупреждений компилятора GCC 7.5.0. По реализованной программе должна быть написана пояснительная записка, которая включает постановку задачи с указанием варианта задания, описание программы и алгоритма решения поставленной задачи, тестирование программы на произвольных входных данных и выводы.

Для подтверждения навыков, которые были приобретены в процессе выполнения работы, студент проходит процедуру защиты курсовой работы. Защита включает в себя выполнение задания, аналогичного пунктам из курсовой работы. Например, для курсовой с обработкой изображения необходимо будет расширить выбор действий в курсовой работе и добавить увеличение размера картинки в три раза.

1. СОЗДАНИЕ ИГРЫ ПРИ ПОМОЩИ ПСЕВДОГРАФИКИ

1.1. Цель и задачи

Целью данной курсовой работы является получение знаний о создании простейших игр в консоли (терминале), графический интерфейс которых реализован с использованием псевдографики.

Для достижения поставленной цели требуется решить следующие задачи:

1) получить базовые навыки работы в консоли;

2) изучить функцию разбора аргументов командной строки getopt;

3) изучить понятие псевдографики;

4) реализовать логику игры в соответствие с заданием;

5) реализовать консольный интерфейс с использованием функции getopt.

1.2. Основные теоретические сведения

1.2.1. Работа в терминале

Иногда при запуске программы бывает необходимо передать в нее аргументы, которые она будет использовать в процессе работы. Использование аргументов особенно полезно, если программа не имеет графического интерфейса. В таком случае для передачи необходимых параметров в программу используется командная строка.

Примером может служить утилита для проверки соединения ping. Для того чтобы проверить доступ к ip-адресу или домену, нужно передать их первым аргументом после названия команды:

```
ping 8.8.8.8
```

Для доступа к аргументам командной строки в программе на языке Си их нужно объявить как аргументы функции main:

```
int main(int argc, char* argv[])
```

После запуска программы в argc будет храниться количество переданных аргументов, а в argv – массив аргументов. При этом первым аргументом всегда является имя исполняемого файла.

Напишем небольшую программу, которая будет считывать значения переданных аргументов, а затем выводить эти значения в консоль:

```
#include <stdio.h>

int main(int argc, char* argv[]){
   for(int i = 0; i < argc; i++)
       printf(">>%s\n", argv[i]);
   return 0;
}
```

Если имя исполняемого файла a.out, чтобы запустить программу, нужно выполнить команду:

```
./a.out  1 2 3
```

Программа выведет название исполняемого файла и переданные ей аргументы:

```
>>./a.out
>>1
>>2
>>3
```

Такой способ взаимодействия с аргументами командной строки называется использованием неименованных позиционных аргументов. Данный способ является не очень удобным из-за того, что приходится каждый раз передавать все необходимые аргументы, соблюдая определенную последовательность. Чем больше аргументов будет принимать программа, тем сложнее пользователю будет их запомнить для использования.

Для удобства введем условные обозначения при передаче аргументов в программу. Пусть символ «-» указывает на то, что следующий за ним символ является ключевым, т. е. обозначает некоторый атрибут. Далее после пробела укажем значение этого атрибута. Если какой-либо атрибут не будет передан в программу, установим для него значение по умолчанию.

Модифицируем нашу программу в соответствии с введенными правилами. Пусть она будет выводить переданную строку заданное количество раз:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main(int argc, char* argv[]){
    int count = 0;
    char *print_str = "Hello World";
    // разбираем аргументы командной строки
    for (int i = 1; i < argc; i++){
        // сравниваем переданный аргумент с известными
        if (!strcmp("-s", argv[i]) && i + 1 < argc){
            // выполняем необходимые для данного аргумента действия
            print_str = argv[i + 1];
        } else if (!strcmp("-c", argv[i]) && i + 1 < argc){
            count = atoi(argv[i + 1]);
        }
    }
    for (int i = 0; i < count; i++){
        printf("%s\n", print_str);
    }
    return 0;
}
```

Запустим программу с необходимыми аргументами:

```
./a.out -c 3 -s hello
```

Вывод программы:

```
hello
hello
hello
```

Попробуем ввести более длинную строку:

```
./a.out -c 3 -s hello world
```

Вывод программы:

```
hello
hello
hello
```

Чтобы передать строку с пробелами в качестве аргумента, необходимо заключить ее в кавычки, либо экранировать пробельные символы в ней, используя «\». Запустим программу с измененными параметрами:

```
./a.out -c 3 -s hello\ world
```

Вывод программы:

```
hello world
hello world
hello world
```

Созданная программа простейшим образом обрабатывает переданные аргументы. Однако такой подход неудобен – приходится каждый раз заново «изобретать» систему разбора параметров при написании программы, самостоятельно контролировать наличие определенных из них. Неудобство особенно заметно, если придется добавить или изменить аргумент.

1.2.2. Функция getopt

Для разбора аргументов командной строки есть специальная функция getopt()[1]:

```
getopt(int argc, char *const argv[], const char *optstring);
```

Целое число argc содержит количество переданных в программу аргументов.

Массив аргументов argv содержит указатели на переданные в программу аргументы.

Параметр optstring является строкой, содержащей допустимые аргументы(опции), которые могут быть переданы в программу. Если программа разрешает только опции -h и -v, в качестве строки опций необходимо использовать «hv».

Функция getopt определена в заголовочном файле unistd.h Она возвращает различные значения:

– если опция успешно найдена, возвращается символ опции;

– -1, если все опции разобраны, других опций нет;

– ?, если getopt() встречает символ опции который отсутствует в optstring;

– если встречается опция с отсутствующим аргументом, то возвращаемое значение зависит от первого символа в строке optstring: если это ':', то возвращается ':'; в противном случае возвращается '?'.

Обычно чтение аргументов строки используется в цикле, а также с конструкцией switch-case. Добавим в программу помимо чтения аргументов строки с помощью функции getopt() оператор switch-case, который делает различные действия в зависимости от переданного аргумента. Функция printHelp() печатает доступные аргументы и их значения.

Для перемещения по массиву аргументов, программа использует глобальную переменную optind. По умолчанию она установлена в 1, тем самым указывая на индекс первого аргумента командной строки, идущего после аргумента с названием запущенной программы:

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

void printHelp(){
   printf("getopt example\n");
   printf("-v - verbose\n");
   printf("-h -? - help\n");
}

int main(int argc, char* argv[]){
   char *opts = "vh?";
   int opt;
   opt = getopt(argc, argv, opts);
   while(opt != -1){
       switch(opt){
           case 'v':
               // некоторый код
```

```
                break;
            case 'h':
            case '?':
                printHelp();
                return 0;
        }
        opt = getopt(argc, argv, opts);
    }
    argc -= optind;
    argv += optind;
    for(int i = 0; i < argc; i++){
        printf(">>%s\n", argv[i]);
    }
    return 0;
}
```

Если имя исполняемого файла a.out, чтобы запустить программу с опцией -h нужно выполнить команду:

```
./a.out  -h
```

Программа выведет название опций, заданных в функции printHelp():

```
getopt example
-v - verbose
-h -? - help
```

Аналогично можно передать один из заданных аргументов. Если переданный аргумент не задан, программа выведет в консоль сообщение:

```
 ./a.out: invalid option -- <название введенной опции>
getopt example
-v - verbose
-h -? - help
```

1.3. Создание консольной игры с использованием псевдографики

С помощью терминала и псевдографики можно разрабатывать простые игры. Реализуем игру «Крестики-нолики» для двух игроков. Игра является пошаговой, имеет конечное число состояний, которые можно визуализировать игровым полем.

Разделим задачу создания игры на подзадачи: создание логики игры и визуализация состояния поля при помощи псевдографики.

Псевдографикой является рисование растровых изображений при помощи различных символов. Пример изображения, созданного при помощи псевдографики, можно увидеть на рис. 1.1.

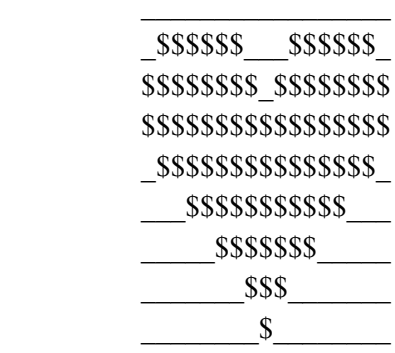

Рис. 1.1. Рисунок при помощи псевдографики

С помощью псевдографики будет происходить отображение внутреннего состояния игры и его изменения после хода одного из игроков или вследствие завершения игры.

1.3.1. Создание логики игры

Для данной игры необходимо создать игровое поле, обеспечить передачу хода между игроками, реализовать проверку выигрыша одного из игроков.

Состояние игры зависит от типа клеток на поле и их расположения относительно друг друга. Клетка поля может находиться в одном из трех состояний: empty (пустая клетка), cross (клетка с крестиком), zero (клетка с ноликом). Изначально все клетки находятся в состоянии empty. Так как количество состояний конечно, для реализации воспользуемся перечислением enum cell {empty, cross, zero}.

Будем использовать стандартное поле размера 3*3, определим константу N с помощью директивы define:

```
#define N 3
```

Таким образом, поле будет храниться следующим образом:

```
int field[N][N];
```

Перед началом игры необходимо проинициализировать поле. Для этого реализуем функцию set_default_field, которая присвоит каждой клетке поля значение empty:

```
void set_default_field(int field[N][N]){
   for (int i = 0; i < N; i++){
       for (int j = 0; j < N; j++){
```

```
            field[i][j] = empty;
        }
    }
}
```

Реализуем функцию change_cell, которая изменит состояние клетки, находящейся по заданным координатам в зависимости от того, кому принадлежит ход.

```
void change_cell(int x, int y, int symbol, int field[N][N]){
    if (symbol == zero || symbol == cross){
        field[x][y] = symbol;
    }
}
```

После хода каждого игрока, начиная с пятого, необходимо проверять, не выиграл ли один из игроков. Для проверки реализуем функцию check_winner, которая будет возвращать 0 или 1, указывая на то, выиграл игрок, сделавший ход, или нет. Функция реализует проверку строк. Проверка столбцов и диагоналей остается читателю на самостоятельное выполнение:

```
int check_winner(int field[N][N], int player){
    for (int i = 0; i < N; i++){
        int is_win = 1;
        for (int j = 1; j < N; j++){
            if (field[i][j] != field[i][j - 1] || field[i][j] != player){
                is_win = 0;
                break;
            }
        }
        if (is_win){
            return is_win;
        }
    }
    return 0;
}
```

1.3.2. Создание псевдографики

Для игры необходимо обеспечить отображение игрового поля на экране и дать пользователям интерфейс для взаимодействия с игрой.

Для того чтобы игрок смог сделать ход, он должен иметь возможность выбрать одну из клеток на игровом поле. Чтобы идентифицировать клетку, можно использовать ее координаты в поле (x и y). Договоримся, что началом координат будет верхний левый угол поля.

Считывание координат можно реализовать следующим образом:

```
int val = scanf("%d %d", &x, &y);
```

Значение val можно использовать для проверки корректности количества считанных значений. Таким образом, игроку во время своего хода будет необходимо ввести координаты клетки с клавиатуры. Для этого будем выводить подсказку о том, что нужно ввести и в каком формате, а затем считывать данные:

```
printf("Ход игрока %d. Введите координаты клетки для хода через пробел: ", player);
```

Так как игрок может ввести некорректные координаты в программу, необходимо реализовать их проверку. Для этого реализуем функцию check_coordinates:

```
int check_coordinates(int x, int y, int field[N][N]){
    return x >= 0 && y >= 0 && x < N && y < N && field[x][y] == empty;
}
```

После каждого хода изменение поля должно отображаться на экране. Функция print_field, будет рисовать поле на экране с помощью заданных символов. Выберем эти символы. Пусть « » (пробел) будут отображать пустую клетку, а заглавные латинские буквы «O» и «X» будут отображать соответственно нолики и крестики. Для того, чтобы поле всегда находилось в одном и том же месте и было видно его изменение, будем очищать консоль после хода каждого из игроков. Для этого используем управляющие последовательности для очищения терминала (\033[2J) и для возвращения курсора в левый верхний угол терминала(\033[H)[2]:

```
void print_field(int field[N][N]){
   printf("\033[2J\033[H");
   for (int i = 0; i < N; i++){
       for (int k = 0; k < N; k++){
```

```
            printf("--");
        }
        printf("\n");
        for (int j = 0; j < N; j++){
            printf("|");
            switch (field[i][j]){
                case (empty):
                    printf(" ");
                    break;
                case (cross):
                    printf("X");
                    break;
                case (zero):
                    printf("0");
                    break;
            }
        }
        printf("|\n");
    }
    for (int k = 0; k < N; k++){
        printf("--");
    }
    printf("\n");
}
```

Для вывода информации о результатах игры реализуем функцию print_result, которая будет выводить информацию о выигрыше одного из игроков, либо о ничьей в зависимости от переданного значения переменной результата:

```
void print_result(int result, int field[N][N]){
    print_field(field);
    if (result == empty){
        printf("Игра окончена. Ничья.\n");
    } else if (result == cross){
        printf("Игра окончена. Победил первый игрок.\n");
    } else {
```

```
        printf("Игра окончена. Победил второй игрок.\n");
    }
}
```

1.4. Пример задания к курсовой работе

Необходимо добавить на поле дополнительную клетку – «джокер», которая будет засчитываться как клетка обоих игроков.

1.5. Вопросы для контроля

1. Что такое getopt и зачем его использовать?
2. Что такое псевдографика?
3. Что такое управляющая последовательность?

2. ОБРАБОТКА ИЗОБРАЖЕНИЙ

2.1. Цель и задачи

Целью курсовой работы является изучение форматов файлов BMP и PNG, а также реализация функций для работы с этими форматами файлов.

Для достижения поставленной цели требуется решить следующие задачи:

1) изучить BMP и PNG форматы изображений;
2) получить информацию об изображении: размеры, содержимое и др.;
3) обработать массив пикселей в соответствии с заданием;
4) обработать исключительные случаи;
5) сохранить итоговое изображение в новый файл.

2.2. Основные теоретические сведения

2.2.1. BMP

Файл **BMP**[4] – это изображение, сохраненное в формате **Bitmap** (BMP), который разработан компанией Microsoft. Файл содержит несжатые данные изображения. В нем можно хранить двухмерные изображения в цветном или черно-белом виде. Формат BMP также известен как формат Device Independent Bitmap (DIB). Он используется для сохранения изображений, которые выглядят одинаково на различных устройствах.

Формат BMP начинается с заголовка файла, включающего идентификатор изображения, размер файла, ширину, высоту, метод сжатия данных (необязательно), параметры цвета и начальную точку отсчета данных. После заголовка формат хранит необработанные данные пиксельного изображения и необязательные данные цветового профиля ICC.

Поскольку файлы BMP не имеют сжатия, они могут быть большими. Поэтому форматы изображений JPEG и PNG являются распространенной альтернативой формату BMP для сохранения и передачи цифровых изображений.

2.2.2. Структура файла BMP

Как уже было сказано ранее, изображение состоит из блоков, расположенных в заранее определенной последовательности. Подробная информация о расположении, названиях и содержимом блоков представлена в табл. 1.1. В файле может присутствовать множество различных версий некоторых из этих блоков, что обусловлено длительной эволюцией формата. Подробнее о версиях BMP формата можно прочитать в [3].

Таблица 1.1

Структура файла BMP

Блок	Размер	Обязательный параметр	Описание
Заголовок файла (Bitmap file header)	14 байт	Да	Используется для хранения общей информации об изображении
Заголовок DIB (Bitmap info header)	Фиксированный размер (7 версий)	Да	Используется для хранения подробной информации об изображении и определения формата пикселей
Дополнительные битовые маски (Extra bit masks)	12 или 16 байт	Нет	Используется для определения формата пикселей
Цветовая палитра (Color table)	Переменный размер	Полу-опциональный*	Используется для определения цветов, используемых изображением (массивом пикселей)
Пробел1 (Gap1)	Переменный размер	Нет	Используется для выравнивания структуры
Массив пикселей (Pixel array)	Переменный размер	Да	Используется для определения фактических значений пикселей. Формат пикселей определяется заголовком DIB или дополнительными битовыми масками. Каждая строка в массиве заполняется до размера, кратного 4 байтам
Пробел2 (Gap2)	Переменный размер	Нет	Используется для выравнивания структуры
Цветовой профиль ICC (ICC color profile)	Переменный размер	Нет	Используется для определения цветового профиля для управления цветом

* Обязательно используется при битности цвета 8 и ниже, может использоваться при битности 8 и выше для оптимизации цветов.

Для начала рассмотрим заголовок файла (Bitmap file header). Этот блок байтов находится в начале файла и используется для идентификации файла, описание структуры представлено в табл. 1.2. Приложения сначала считывают этот блок, чтобы убедиться, что файл действительно является файлом BMP и что он не поврежден. Первые 2 байта файла формата BMP – это символ «B», затем символ «M» в кодировке ASCII. Все целочисленные значения хранятся в формате little-endian (т. е. наименее значимый байт первым).

Таблица 1.2

Структура заголовка Bitmap file header

Поле	Размер	Описание
Signature	2 байта	Поле заголовка, используемое для идентификации файла BMP и DIB, имеет шестнадцатеричное значение, равное BM в ASCII
FileSize	4 байта	Размер файла BMP в байтах
Reserved1	2 байта	Зарезервировано; фактическое значение зависит от приложения, создающего изображение
Reserved2	2 байта	Зарезервировано; фактическое значение зависит от приложения, создающего изображение
FileOffsetToPixelArray	4 байта	Смещение, т. е. начальный адрес байта, в котором находятся данные изображения (массив пикселей)

Причина существования различных заголовков заключается в том, что компания Microsoft несколько раз расширяла формат DIB.

Заголовок изображения Bitmap info header. В табл. 1.3 описаны поля Bitmap info header, которые пригодятся вам при выполнении курсовой работы и их описание.

Таблица 1.3

Структура заголовка Bitmap info header

Поле	Размер	Описание
HeaderSize	4 байта	Размер этого заголовка в байтах
ImageWidth	4 байта	Ширина изображения в пикселях (целое знаковое число – signed int)
ImageHeight	4 байта	Длина изображения в пикселях (целое знаковое число – signed int)
Planes	2 байта	Количество цветовых плоскостей (должно быть 1)
BitsPerPixel	2 байта	Глубина цвета изображения. Типичными значениями являются 1, 4, 8, 16, 24 и 32
Compression	4 байта	Используемый метод сжатия
ImageSize	4 байта	Размер изображения

Окончание табл. 1.3

Поле	Размер	Описание
XpixelsPerMeter	4 байта	Горизонтальное разрешение изображения (количество пикселей на метр, целое знаковое число – signed int)
YpixelsPerMeter	4 байта	Вертикальное разрешение изображения (количество пикселей на метр, целое знаковое число – signed int)
TotalColors	4 байта	Количество цветов в цветовой палитре, или 0 – по умолчанию 2^n
ImportantColors	4 байта	Количество используемых важных цветов, или 0, если каждый цвет важен; обычно игнорируется

Цветовые модели RGB и RGBA. Для представления цветов пикселя в компьютере используются цветовые модели, которые позволяют закодировать цвета набором байт с определенными значениями.

RGB представляет собой *цветовое пространство*, состоящее из трех каналов: красного, зеленого и синего. Вместе эти каналы представляют информацию о цвете пикселя. Значением каждого канала может быть число от 0 до 255, т. е. интенсивность каждого цвета равна значению каждого канала. Например, если для каждого канала задано значение 255, визуализируемый цвет будет белым. А если значения каждого канала 0, визуализируемый цвет будет черным. При этом комбинация разных значений каждого канала позволяет получить разные цвета.

RGBA – это цветовое пространство, которое включает дополнительный канал (альфа-канал) для представления информации о прозрачности изображения. В RGBA информация о прозрачности хранится в альфа-канале, а информация о цвете – в трех каналах RGB. Альфа-канал тоже принимает значения от 0 до 255: 0 представляет максимальный уровень прозрачности, а 255 – минимальный уровень прозрачности.

***Цветовая палитра* (*Color table*).** Цветовая палитра BMP находится в файле изображения BMP непосредственно после заголовка файла BMP, заголовка DIB. Таблица цветов представляет собой блок байтов, в котором перечислены цвета, используемые изображением. Каждый пиксель в индексированном цветном изображении описывается рядом битов (1, 4 или 8), которые являются индексом одного цвета, описанного в этой таблице. Цветовая палитра используется, чтобы информировать приложение о фактическом цвете, которому соответствует каждое из этих индексных значений. Целью таблицы

цветов в неиндексированных растровых изображениях (в которых отсутствует палитра) является перечисление цветов, используемых растровым изображением в целях повышения производительности на устройствах с ограниченными возможностями цветного отображения и облегчения будущего преобразования в другие форматы пикселей и палитры. Схематичное представление цветовой палитры находится в табл. 1.4.

Таблица 1.4

Содержимое Color table

Цветовая палитра
Определение цвета (индекс 0)
Определение цвета (индекс 1)
…
Определение цвета (индекс n – 1)

***Массив пикселей* (*Pixel array*).** Массив пикселей представляет собой блок, состоящий из 32-битных значений, описывающих изображение попиксельно. Обычно пиксели *хранятся «снизу вверх»*, *начиная с левого нижнего угла*, двигаясь слева направо, а затем построчно снизу вверх изображения. Схематичное представление массива пикселей находится в табл. 1.5.

Таблица 1.5

Содержимое Pixel array

Данные изображения Массив пикселей[x, y]					
Пиксель[0,h-1]	Пиксель[1,h-1]	…	Пиксель[w-2,h-1]	Пиксель[w-1,h-1]	Выравнивание
Пиксель[0,h-2]	Пиксель[1,h-2]	…	Пиксель[w-2,h-2]	Пиксель[w-1,h-2]	Выравнивание
…					
Пиксель[0,2]	Пиксель[1,2]	…	Пиксель[w-2,2]	Пиксель[w-1,2]	Выравнивание
Пиксель[0,1]	Пиксель[1,1]	…	Пиксель[w-2,1]	Пиксель[w-1,1]	Выравнивание
Пиксель[0,0]	Пиксель[1,0]	…	Пиксель[w-2,0]	Пиксель[w-1,0]	Выравнивание

2.2.3. Чтение и запись bmp-файлов, вывод информации из заголовков

Файлы в формате BMP состоят из заголовка файла (file header), информационного заголовка (info header), непосредственно массива пикселей и прочих опциональных частей. Для хранения заголовков были реализованы структуры BitmapFileHeader и BitmapInfoHeader:

```
typedef struct {
   unsigned short signature;
```

```
    unsigned int filesize;
    unsigned short reserved1;
    unsigned short reserved2;
    unsigned int pixelArrOffset;
} BitmapFileHeader;

typedef struct {
    unsigned int headerSize;
    unsigned int width;
    unsigned int height;
    unsigned short planes;
    unsigned short bitsPerPixel;
    unsigned int compression;
    unsigned int imageSize;
    unsigned int xPixelsPerMeter;
    unsigned int yPixelsPerMeter;
    unsigned int colorsInColorTable;
    unsigned int importantColorCount;
} BitmapInfoHeader;
```

Для реализации структур использовалось ключевое слово typedef. С помощью него можно задать альтернативное имя для структуры, которое будет использоваться в программе. Массив пикселей состоит из структур RGB, в которых хранятся значения красной, зеленой и синей компоненты цвета пикселя:

```
typedef struct {
    unsigned char b;
    unsigned char g;
    unsigned char r;
} Rgb;
```

Для корректного считывания данных из файла в реализованные структуры необходимо выполнить выравнивание с помощью команд:

```
#pragma pack (push, 1)
// структуры для bmp файла
#pragma pack(pop)
```

Это необходимо, так как по умолчанию выравнивание структуры кратно размеру машинного слова и равно размеру регистра операционной системы (4 или 8 байт) [5], из-за этого во время считывания данных в структуру может произойти запись некорректного значения в одно или несколько полей.

Для хранения массива пикселей используется двумерный массив Rgb **arr. Считывание файла происходит при помощи функции read_bmp, в этой функции последовательно считываются заголовки, а затем и сам массив пикселей:

```
Rgb **read_bmp(char file_name[], BitmapFileHeader*
bmfh, BitmapInfoHeader* bmif){
   FILE *f = fopen(file_name, "rb");
   fread(bmfh, 1, sizeof(BitmapFileHeader), f);
   fread(bmif, 1, sizeof(BitmapInfoHeader), f);
   unsigned int H = bmif->height;
   unsigned int W = bmif->width;
   Rgb **arr = malloc(H * sizeof(Rgb*));
   for(int i = 0; i < H; i++){
       arr[i] = malloc(W * sizeof(Rgb) + (W * 3) % 4);
       fread(arr[i], 1, W * sizeof(Rgb) + (W * 3) % 4,
f);
   }
   fclose(f);
   return arr;
}
```

Для вывода информации, хранящейся в заголовках, реализованы функции print_file_header и print_info_header:

```
void print_file_header(BitmapFileHeader header){
   printf("signature:\t%x (%hu)\n", header.signature,
header.signature);
   printf("filesize:\t%x (%u)\n", header.filesize,
header.filesize);
   printf("reserved1:\t%x (%hu)\n", header.reserved1,
header.reserved1);
   printf("reserved2:\t%x (%hu)\n", header.reserved2,
header.reserved2);
   printf("pixelArrOffset:\t%x (%u)\n", head-
er.pixelArrOffset, header.pixelArrOffset);
}
void print_info_header(BitmapInfoHeader header){
   printf("headerSize:\t%x (%u)\n", header.headerSize,
header.headerSize);
   printf("width:     \t%x (%u)\n", header.width, head-
er.width);
```

```
   printf("height:    \t%x (%u)\n", header.height,
header.height);
   printf("planes:    \t%x (%hu)\n", header.planes,
header.planes);
   printf("bitsPerPixel:\t%x (%hu)\n", head-
er.bitsPerPixel, header.bitsPerPixel);
   printf("compression:\t%x (%u)\n", head-
er.compression, header.compression);
   printf("imageSize:\t%x (%u)\n", header.imageSize,
header.imageSize);
   printf("xPixelsPerMeter:\t%x (%u)\n", head-
er.xPixelsPerMeter, header.xPixelsPerMeter);
   printf("yPixelsPerMeter:\t%x (%u)\n", head-
er.yPixelsPerMeter, header.yPixelsPerMeter);
   printf("colorsInColorTable:\t%x (%u)\n", head-
er.colorsInColorTable, header.colorsInColorTable);
   printf("importantColorCount:\t%x (%u)\n", head-
er.importantColorCount, header.importantColorCount);
}
```

Сохранение файла реализовано в функции write_bmp, где сначала в выходной файл записываются заголовки, а затем массив пикселей:

```
void write_bmp(char file_name[], Rgb **arr, int H, int
W, BitmapFileHeader bmfh, BitmapInfoHeader bmif){
   FILE *ff = fopen(file_name, "wb");
   fwrite(&bmfh, 1, sizeof(BitmapFileHeader), ff);
   fwrite(&bmif, 1, sizeof(BitmapInfoHeader), ff);
   for(int i = 0; i < H; i++){
       fwrite(arr[i], 1, W * sizeof(Rgb) + (W * 3) % 4,
ff);
   }
   fclose(ff);
}
```

2.2.4. Примеры

После считывания bmp-файла появляется возможность его преобразования: изменение цветов на всем изображении, рисование объектов, изменение размера изображения и другое. Для того чтобы понять, как изменять bmp-файл, реализуем несколько функций.

Работа с цветом. С цветами на картинке можно работать, меняя значения компонент каждого пикселя. Присваивая компонентам различные значения, можно совершить обмен цветовых каналов на изображении, усилить цветность изображения по одному из каналов, сделать инверсию цветов и многое другое.

Произведем обмен цветов изображения: поменяем местами красную и зеленую компоненту у каждого пикселя. Для этого напишем функцию swap, которая будет производить обмен значений компонент цветов пикселей, а также функцию swap_channels, которая будет проходить по всем пикселям изображения и применять к ним функцию swap:

```
void swap(char *a, char *b){
    char t = *a;
    *a = *b;
    *b = t;
}

Rgb **swap_channels(Rgb **arr, int H, int W){
    for(int i=0; i<H; i++){
        for(int j=0; j<W; j++){
            swap(&arr[i][j].r, &arr[i][j].g);
        }
    }
    return arr;
}
```

Вызовем функцию swap_channels в программе:

```
arr = swap_channels(arr, H, W);
```

В результате вызова функции изображение приобрело зеленый оттенок. Произведем усиление одного из цветов на изображении. Для этого напишем функцию add_red, которая будет устанавливать максимальное значение для красной компоненты пикселя:

```
Rgb **add_red(Rgb **arr, int H, int W){
    for(int i = 0; i < H; i++){
        for(int j = 0; j < W; j++){
            arr[i][j].r = 255;
        }
    }
    return arr;
}
```

Вызовем функцию add_red в программе:

```
arr = add_red(arr, H, W);
```

В результате вызова функции изображение приобрело ярко выраженный красный оттенок.

Рисование линий. При помощи изменения пикселей на картинке можно рисовать различные объекты. Рисование вертикальной и горизонтальной линии реализовано в функции draw_line. Функция принимает на вход массив пикселей arr, высоту изображения в пикселях H, ширину изображения в пикселях W, координаты начала линии x0 и y0, координаты конца линии x1 и y1, толщину линии в пикселях thickness, указатель на массив, содержащий rgb-компоненты цвета линии color. В функции сначала происходит определение вида линии (вертикальная, горизонтальная, наклонная*), а затем изменение цвета необходимых пикселей. Также в начале функции реализована проверка на корректность входных аргументов:

```
void draw_line(Rgb **arr, int H, int W, int x0, int y0,
int x1, int y1, int thickness, int* color){
   if (x0 < 0 || y0 < 0 || x1 < 0 || y1 < 0 || thick-
ness <= 0){
        return ;
   }
   // вертикальная линия
   if (x0 == x1){
        if (y0  > y1){
            swap_int(&y0, &y1);
        }
        for (int y = y0; y < y1; y++){
            for (int j = 0; j < thickness; j++){
                if (H - y >= 0 && x0 - j >= 0 && x0 - j
< W && H - y < H){
                    arr[H - y][x0 - j].r = *(color);
                    arr[H - y][x0 - j].g = *(color + 1);
                    arr[H - y][x0 - j].b = *(color + 2);
                }
            }
        }
        // горизонтальная линия
   } else if (y0 == y1){
```

```
        if (x0 > x1){
            swap_int(&x0, &x1);
        }
        for (int x = x0; x < x1; x++){
            for (int j = 0; j < thickness; j++){
                if (H - y0 + j >= 0 && x >= 0 && x < W
&& H - y0 + j < H){
                    arr[H - y0 + j][x].r = *(color);
                    arr[H - y0 + j][x].g = *(color + 1);
                    arr[H - y0 + j][x].b = *(color + 2);
                }
            }
        }
    }
}
```

Для корректной работы цикла в функции draw_line, где происходит проход по массиву пикселей и их изменение, была реализована функция swap_int:

```
void swap_int(int *a, int *b){
    int t = *a;
    *a = *b;
    *b = t;
}
```

В отличие от вертикальных и горизонтальных линий, наклонную линию не удастся нарисовать при помощи простого последовательного прохода в цикле по массиву пикселей. Это происходит из-за того, что наклонная линия в растровом изображении выглядит как «лестница» с разными длинами «ступенек», длина которых зависит от величины наклона линии. При рисовании такой линии задача заключается в том, чтобы получить приближение к прямой линии между двумя заданными точками. Одним из алгоритмов, позволяющих выполнить такое приближение, является алгоритм Брезенхема [6].

Вырезание области на изображении. При работе с изображением бывает полезно уметь вырезать интересующую область на изображении. В качестве примера реализуем функцию cut_image для обрезки изображения до квадратной области заданного размера. Функция принимает на вход массив пикселей arr, указатель на структуру информационного заголовка bmif, координаты левого верхнего угла области для обрезки x и y, желаемый размер квадрата будущей вырезанной области size. Функция будет возвращать указатель на новый массив измененный массив new_ arr:

```
Rgb ** cut_image(Rgb **arr, BitmapInfoHeader * bmif,
int x, int y, int size){
    // выделяем место под обрезанную картинку
    Rgb ** new_arr = malloc(size * sizeof(Rgb *));
    for (int i = 0; i < size; i++){
        new_arr[i] = malloc(size * sizeof(Rgb));
    }
    // выбираем определенную часть из исходного изобра-
жения
    for (int i = 0; i < size; i++){
        for (int j = 0; j < size; j++){
            if (i >= 0 && j >= 0 && j < bmif->width &&
                bmif->height - y - i >= 0 && bmif-
>height - y - i < bmif->height && x + j >= 0 && x + j <
bmif->width){
                new_arr[size - i - 1][j] = arr[bmif-
>height - y - i][x + j];
            }
        }
    }
    // удаляем старую картинку
    for(int i = 0; i < bmif->height; i++){
        free(arr[i]);
    }
    free(arr);
    // меняем размер картинки в информационном заголовке
    bmif->height = size;
    bmif->width = size;
    return new_arr;
}
```

2.3. PNG

Формат файлов PNG широко используется на веб-сайтах для отображения высококачественных цифровых изображений. Созданный для того, чтобы превзойти по производительности файлы GIF, PNG обеспечивает не только сжатие без потерь, но и более широкую и яркую цветовую палитру.

PNG – это сокращение от **Portable Network Graphic**, тип файла растрового изображения. Этот тип файлов особенно популярен среди вебдизайнеров, поскольку он позволяет работать с графикой с прозрачным или полупрозрачным фоном.

PNG – это следующая эволюция формата GIF, который к моменту появления PNG существовал уже восемь лет. У GIF было несколько недостатков, таких как необходимость получения патентной лицензии и ограниченный диапазон – всего 256 цветов, что не соответствовало постоянно растущему разрешению экрана компьютера. Чтобы избежать этих проблем, файлы PNG были сделаны свободными от патентов и включали значительно большую палитру цветов.

2.3.1. Чтение и запись PNG-файлов

В данном разделе будет рассмотрен пример работы с png изображением при помощи библиотеки libpng [7]. Изображение, над которым будут производиться все действия, приведено на рис. 2.1.

Для установки libpng можно использовать команду

sudo apt install libpng-dev

Работа с изображением делится на три части:

– чтение изображения, т. е. его заголовка и массива пикселей;

– обработка изображения;

– запись в файл.

Программа находится в файле demo_png.c. Для его компиляции и запуска будем использовать команду

gcc demo_png.c -lpng && ./a.out img.png res.png

Рис. 2.1. Исходное изображение res.png

Для линковки библиотеки libpng необходимо добавить флаг **-lpng**.

В данной реализации в аргументах указывается:

– путь до изначального изображения;

– путь, куда будет сохранено конечное изображение.

Пусть main выглядит следующим образом:

```
int main(int argc, char **argv) {
   if (argc != 3){
       fprintf(stderr,"Usage: program_name <file_in>
<file_out>\n");
       return 0;
   }
   // Структура в которой хранится информация об изоб-
ражении
   struct Png image;
   read_png_file(argv[1], &image);
   process_file(&image);
   write_png_file(argv[2], &image);
```

```
    return 0;
}
```

Для корректной работы программы необходимо подключить заголовочные файла, включая png.h, в котором и объявлены функции для работы с png изображением:

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <png.h>
```

Рассмотрим структуру Png. Структура выглядит следующим образом:

```
struct Png{
    int width, height;
    png_byte color_type;
    png_byte bit_depth;

    png_structp png_ptr;
    png_infop info_ptr;
    int number_of_passes;
    png_bytep *row_pointers;
};
```

В данном примере используется сигнатура Png, которая содержит в себе информацию об изображении, опишем основные поля:

– ширина и высота изображения;

– тип цвета, который используется в изображении (например, используется ли палитра, монохромное изображение, присутствует ли альфа канал);

– глубина цвета, т. е. сколько компонент (в данном случае байтов) описывают цвет (для RGBA – это 4, а для RGB – 3);

– количество проходов, которое необходимо, чтобы полностью обработать изображение. Данный параметр необходим для быстроты визуализации изображения. Можно заметить, что некоторые изображения в интернете сначала загружаются как будто в малом разрешении, но это не так. На самом идет последовательно подгрузка остальной части изображения. Более подробно это описано в документации libpng [7].

Рассмотрим 1-ю функцию программы, которая отвечает за чтение png изображения:

```
void read_png_file(char *file_name, struct Png *image)
{
int x,y;
```

```
char header[8];  //длина заголовка 8
/* Открыть и проверить, что файл png */
FILE *fp = fopen(file_name, "rb");
if (!fp){
   printf("Cannot read file: %s\n", file_name);
   return;
}
…
}
fread(header, 1, 8, fp);
if (png_sig_cmp(header, 0, 8)){
   printf("probably, %s is not a png\n", file_name);
   return;
}
```

Данный участок кода сначала читает файл стандартной функцией fopen и проверяет по возможной сигнатуре, является ли это изображение PNG. Важно, что если при сохранении файла не будет указана верная сигнатура, то в следующий раз картинка не откроется, даже если не был изменен ни один пиксель. Код для проверки сигнатуры PNG:

```
/* Инициализация структуры PNG */
image->png_ptr = png_create_read_struct(PNG_LIBPNG_VER_
STRING, NULL, NULL, NULL);
if (!image->png_ptr){
// Блок исполнится при ошибке в структуре PNG
   printf("error in png structure\n");
   return;
}
image->info_ptr = png_create_info_struct(image->png_ptr);
if (!image->info_ptr){
   printf("error in png info-structure\n");
   return;
}
==
if (setjmp(png_jmpbuf(image->png_ptr))){
}
```

Данный участок кода создает базовую структуру info_ptr, которая содержит информацию об изображении. Строчка setjmp(png_jmpbuf(image->png_ptr))

необходима, чтобы в случае ошибки у разработчика была возможность обработать этот случай. Если ошибка есть, то setjmp вернет истинное значение.

Рассмотри участок кода, который заполняет структуру PNG:

```
png_init_io(image->png_ptr, fp);
png_set_sig_bytes(image->png_ptr, 8);
png_read_info(image->png_ptr, image->info_ptr);

image->width = png_get_image_width(image->png_ptr, image-
>info_ptr);
image->height = png_get_image_height(image->png_ptr, image-
>info_ptr);
image->color_type = png_get_color_type(image->png_ptr,image-
>info_ptr);
image->bit_depth = png_get_bit_depth(image->png_ptr, image-
>info_ptr);
image->number_of_passes = png_set_interlace_handling(image-
>png_ptr);
png_read_update_info(image->png_ptr, image->info_ptr);

/* чтение файла */

image->row_pointers = (png_bytep *) malloc(sizeof(png_bytep)
* image->height);\
for (y = 0; y < image->height; y++){
   image->row_pointers[y] =
(png_byte*)malloc(png_get_rowbytes(
   image->png_ptr,
   image->info_ptr)
  );
}
png_read_image(image->png_ptr, image->row_pointers);
   fclose(fp);
}
```

Рассмотрим функцию write_png_file, которая отвечает за запись изображения в файл:

```
void write_png_file(char *file_name, struct Png *image) {
     int x,y;
       /* создание файла */
```

```
    FILE *fp = fopen(file_name, "wb");
    if (!fp){
          // Если не открылся файл
    }
      /* Инициализация структуры */
    image->png_ptr =
png_create_write_struct(PNG_LIBPNG_VER_STRING, NULL, NULL,
NULL);
    if (!image->png_ptr){
     // Если не удалось создать структуру
    }
    image->info_ptr = png_create_info_struct(image-
>png_ptr);
    if (!image->info_ptr){
    }
    if (setjmp(png_jmpbuf(image->png_ptr))){
    }
    png_init_io(image->png_ptr, fp);
      /* запись заголовка */
    if (setjmp(png_jmpbuf(image->png_ptr))){
          // Ошибка в записи заголовка
    }
    png_set_IHDR(image->png_ptr, image->info_ptr, image-
>width,
                 image->height, image->bit_depth, image-
>color_type,
                 PNG_INTERLACE_NONE,
PNG_COMPRESSION_TYPE_BASE,
                 PNG_FILTER_TYPE_BASE);
    png_write_info(image->png_ptr, image->info_ptr);
    if (setjmp(png_jmpbuf(image->png_ptr))){
    }
    png_write_image(image->png_ptr, image->row_pointers);
      /* Конец записи */
    if (setjmp(png_jmpbuf(image->png_ptr))){
    }
```

```
    png_write_end(image->png_ptr, NULL);
      /* Очистка памяти */
    for (y = 0; y < image->height; y++)
        free(image->row_pointers[y]);
     free(image->row_pointers);
     fclose(fp);
}
```

Не забываем очищать память, которая была выделена для изображения.

2.3.2. Обработка PNG-изображения

Рассмотрим функцию process_file, которая отвечает за обработку изображения:

```
void process_file(struct Png *image) {
   int x,y;

   if     (png_get_color_type(image->png_ptr,     image-
>info_ptr) !=
       PNG_COLOR_TYPE_RGBA){
       return;
   }

   for (y = 0; y < image->height; y++) {
       png_byte *row = image->row_pointers[y];
       for (x = 0; x < image->width; x++) {
            png_byte *ptr = &(row[x * 4]);
           printf("Pixel at position [ %d - %d ] has
RGBA values: %d - %d - %d - %d\n",
                  x,   y,   ptr[0],   ptr[1],   ptr[2],
ptr[3]);
           /* Изменяем полностью прозрачные пиксели на
полностью непрозрачные (значение альфа-канала) */
           if (ptr[3] == 0) {
             // Изменение альфа канала
                ptr[3] = 255;
           }
// Пример, в котором обнуляется красный и зеленый канал
           /* Красный - 0, Зеленый 0, остается только
синий канал */
```

```
            //      ptr[0] = 0;
            //      ptr[1] = 0;
        }
    }
}
```

Данная функция меняет у каждого пикселя альфа канал. Важная часть данной функции – это получение пикселя:

```
png_byte *ptr = &(row[x * 4]);
```

Так как каждый пиксель состоит из 4 значений, то для обращения к пикселю по индексу необходимо умножить значения индекса на 4. Важно, что не каждое png-изображение имеет альфа канал. Проверка на наличие альфа канала, показана в функции `process_file`.

2.4. Пример задания к курсовой работе

Необходимо считать произвольную картинку с расширением bmp, реализовать возможность выбора следующего действия из консоли: рисование на картинке квадратной рамки красного цвета заданной толщины вокруг всех пикселей синего цвета, при этом не изменяя синие пиксели.

2.5. Описание последовательности выполнения работы

Для выполнения курсовой работы необходимо реализовать программу, которая считывает картинку и позволяет выбрать опцию для вывода результата ее преобразования в соответствии с заданием.

Вам необходимо выполнить следующую последовательность задач:

1. Реализовать в программе опцию по считыванию действия из консоли.
2. Считать картинку из файла.
3. Придумать и реализовать алгоритм для рисования рамки вокруг одного пикселя.
4. Применить реализованный алгоритм ко всем пикселям картинки.
5. Сохранить преобразованную картинку в файл.
6. Очистить выделенную в программе память.

Для начала реализуем функцию, которая при запуске программы будет считывать аргументы из консоли и определять, какое действие необходимо выполнить:

```
int main(int argc, char* argv[]){
    // задаем доступные в программе опции
    char *opts = "bw?";
    int opt;
```

```
    int width = 0; // ширина границы для ф-ии drawBorder
    int makeDraw = 0; // флаг для использования ф-ии
drawBorder
    BitmapFileHeader* bmfh = malloc(sizeof(BitmapFile
Header));
    BitmapInfoHeader* bmif = malloc(sizeof(BitmapInfo
Header));
    Rgb **arr = read_bmp("blue.bmp", bmfh, bmif);
    unsigned int H = bmif->height;
    unsigned int W = bmif->width;
    // разбираем аргументы командной строки
    while(opt != -1){
        switch(opt){
            case 'b':
                makeDraw = 1;
                break;
            case 'w':
                if (optind < argc){
                    // считываем ширину границы из стро-
ки опций
                    sscanf(argv[optind], "%d", &width);
                }
                break;
            case '?':
                freeMemory(bmfh, bmif, arr);
                return 0;
        }
        opt = getopt(argc, argv, opts);
    }
    if (makeDraw && width > 0){
        drawBorder(arr, bmif->height, bmif->width, width);
    }
    write_bmp("blue_out.bmp", arr, H, W, *bmfh, *bmif);
    freeMemory(bmfh, bmif, arr);
    return 0;
}
```

Считывание картинки происходит при помощи структур BitmapFile Header, BitmapInfoHeader, Rgb, а также функции read_bmp, реализованных в разделе 2.3.2.

Определим алгоритм рисования рамки вокруг одного пикселя. Если выбранный пиксель на картинке нужно обвести, то для рисования вокруг него рамки представим, что он является центром квадрата со стороной 2* width (толщина рамки). Тогда задача сведется к тому, что будет необходимо определить верхний левый угол квадрата и пройтись по нему с помощью двойного цикла, меняя цвет каждого пикселя на красный, кроме пикселей, имеющих синий цвет. Реализуем данный алгоритм в функции drawPixelBorder:

```
void drawPixelBorder(Rgb **arr, int H, int W, int x, int y, int border_width){
    // поиск координат левого верхнего угла
    int left = y - border_width;
    int high = x - border_width;
    // рисование рамки вокруг пикселя
    for (int i = high; i < x + border_width; i++){
        for (int j = left; j < y + border_width; j++){
            if (i >= 0 && j >= 0 && i < H && j < W){
                if (!(arr[i][j].r == 0 && arr[i][j].g == 0 && arr[i][j].b == 255)){
                    arr[i][j].r = 255;
                    arr[i][j].g = 0;
                    arr[i][j].b = 0;
                }
            }
        }
    }
}
```

Для того чтобы нарисовать рамку вокруг каждого синего пикселя изображения, реализуем функцию drawBorder, которая пройдется по всем пикселям изображения и применит функцию drawPixelBorder к каждому синему пикселю:

```
void drawBorder(Rgb **arr, int H, int W, int border_width){
    for (int i = 0; i < H; i++){
        for (int j = 0; j < W; j++){
            // проверяем, что пиксель синий
            if (arr[i][j].r == 0 && arr[i][j].g == 0 && arr[i][j].b == 255){
                drawPixelBorder(arr, H, W, i, j, border_width);
```

```
            }
        }
    }
}
```

Сохранение преобразованной картинки происходит при помощи функции write_bmp, реализованной в разделе 2.3.2.

Очистим динамическую память, выделенную в программе, при помощи функции freeMemory:

```
void freeMemory( BitmapFileHeader* bmfh, BitmapInfoHeader* bmif, Rgb **arr){
    for(int i = 0; i < bmif->height; i++){
        free(arr[i]);
    }
    free(arr);
    free(bmfh);
    free(bmif);
}
```

2.6. Вопросы для контроля

1. Что такое Bitmap info header и Bitmap file header в BMP-файле?
2. Что такое альфа-канал в PNG-изображении?
3. Как поменять цвет пикселя в BMP-изображении?
4. Как поменять цвет пикселя в PNG-изображении?

Список рекомендуемой литературы

getopt(3) – Linux manual page. URL: https://man7.org/linux/man-pages/man3/getopt.3.html

ANSI escape code. URL: https://en.wikipedia.org/wiki/ANSI_escape_code

Image File Formats – BMP. URL: https://docs.fileformat.com/image/bmp/

Microsoft Windows Bitmap File Format. URL: https://www.fileformat.info/format/bmp/egff.htm

Data structure alignment – From Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Data_structure_alignment

Bresenham J. E. Algorithm for computer control of a digital plotter // IBM Systems journal. 1965. Т. 4, №. 1. P. 25–30

Документация libpng. URL: http://www.libpng.org/pub/png/pngdocs.html

Оглавление

Заславский Марк Маркович
Лисс Анна Александровна
Гаврилов Андрей Владимирович
Глазунов Сергей Алексеевич
Государкин Ярослав Сергеевич
Тиняков Сергей Алексеевич
Голубева Валентина Петровна
Чайка Константин Владимирович
Допира Валерия Евгеньевна

Базовые сведения к выполнению курсового проекта по дисциплине «Программирование». Второй семестр

Учебно-методическое пособие

Редактор М. Б. Шишкова

Компьютерная верстка И. С. Беляевой

Подписано в печать 11.01.24. Формат 60×84 1/16.
Бумага офсетная. Печать цифровая. Печ. л. 2,25.
Гарнитура «Times New Roman». Тираж 91 экз. Заказ 5.

Издательство СПбГЭТУ «ЛЭТИ»
197022, С.-Петербург, ул. Проф. Попова, 5Ф