

ICO2OBJ

Преобразователь картинок в формате ICO в объектные файлы
(Версия 0.1)

Yellow Rabbit

Позволяет конвертировать графические файлы формата ICO в объектные файлы для последующей линковки в исполняемые образы для БК11М¹.

Входными параметрами являются: перечень имен графических файлов, имя выходного файла и несколько управляющих ключей. На выходе создается объектный файл для линковки.

Конвертор накладывает следующие ограничения на формат входных файлов изображений:

- одна битовая плоскость;
- графические данные без сжатия;
- 4 бита на точку.

Преобразование цветов происходит по следующему принципу:

1. если номер цвета в палитре > 3 , то номер цвета принимается 0;
2. номер цвета является индексом в 4-х элементной таблице, которая ставит комбинацию из двух битов в соответствие номеру цвета.

Встроенная таблица перекодировки цветов может быть изменена опциями командной строки.

Преобразование геометрии происходит всегда, так как в файлах ICO данные хранятся снизу-вверх, справа-налево, но кроме этого предусмотрено преобразование типа “строка в столбец”. Это не транспонирование в буквальном смысле, так как по-другому располагаются не точки изображения, а байты. Еще один вариант: по-другому располагаются слова графических данных.

¹ GIT-репозиторий ассемблера, линковщика и утилит <https://github.com/yrabbit>

29 Декабря 2013 года в 16:24

1. Общая схема программы.

```

⟨ Включение заголовочных файлов 37 ⟩
⟨ Директивы препроцессора ⟩
⟨ Константы 30 ⟩
⟨ Собственные типы данных 5 ⟩
⟨ Прототипы 14 ⟩
⟨ Глобальные переменные 2 ⟩
int main(int argc, char *argv[])
{
    ⟨ Данные программы 3 ⟩
    const char *picname;
    ⟨ Разобрать командную строку 36 ⟩
    /* Поочередно обрабатываем все заданные файлы картинок */
    cur_input = 0;
    ⟨ Записать начало объектного файла 24 ⟩
    while ((picname = config.picnames[cur_input]) ≠ ~) {
        ⟨ Открыть файл картинки 4 ⟩
        handleOneFile(fpic, &hdr);
        fclose(fpic);
        ++ cur_input;
    }
    ⟨ Закрыть объектный файл 25 ⟩
    return (0);
}

```

2. Номер текущего обрабатываемого файла картинки.

⟨ Глобальные переменные 2 ⟩ ≡
static int *cur_input*;

Смотри также секции 15, 21, 31, 32, 34, и 38.

Этот код используется в секции 1.

3. ⟨ Данные программы 3 ⟩ ≡ **FILE** **fpic*;

Смотри также секцию 6.

Этот код используется в секции 1.

4. ⟨ Открыть файл картинки 4 ⟩ ≡ *fpic* = *fopen*(*picname*, "r"); **if** (*fpic* ≡ ~) { **PRINTERR**("Can't open %s\n", *picname*); **return** (ERR_CANTOPEN); } ⟨ Проверить заголовок картинки 7 ⟩

Этот код используется в секции 1.

5. Проверяем соответствие формату ICO.

⟨ Собственные типы данных 5 ⟩ ≡

```
typedef struct _ICO_Header {
    wint16_t zero0;
    wint16_t type;    /* должен быть 1 */
    wint16_t imagesCount;
} ICO_Header;
```

Смотри также секции 8 и 33.

Этот код используется в секции 1.

6. ⟨ Данные программы 3 ⟩ +≡

```
ICO_Header hdr;
```

7. ⟨ Проверить заголовок картинки 7 ⟩ ≡

```
if (fread(&hdr, sizeof (hdr), 1, fpic) ≠ 1) {
    PRINTERR("Can't read header of %s\n", picname);
    return (ERR_CANTOPEN);
}
if (hdr.zero0 ≠ 0 ∨ hdr.type ≠ 1 ∨ hdr.imagesCount ≡ 0) {
    PRINTERR("Bad file header of %s\n", picname);
    return (ERR_BADFILEHEADER);
}
PRINTVERB(1, "Handle file: %s.\n", picname);
PRINTVERB(2, "Images count: %d.\n", hdr.imagesCount);
```

Этот код используется в секции 4.

8. Обработать один файл картинки.

Каждый файл может содержать несколько изображений, которые описываются записями следующего вида.

⟨ Собственные типы данных 5 ⟩ +≡

```
typedef struct _IMG_Header {  
    uint8_t width;      /* если 0, то 256 */  
    uint8_t height;    /* если 0, то 256 */  
    uint8_t colors;  
    uint8_t reserved;  
    uint16_t planes;  
    uint16_t bpp;  
    uint32_t size;     /* размер в байтах */  
    uint32_t offset;   /* смещение до данных изображения от начала файла */  
} IMG_Header;
```

```

9. static void handleOneFile(FILE *fpic, ICO_Header *hdr)
{
    int cur_image;
    IMG_Header *imgs;
    /* размеры картинок не получится хранить в байте, так что храним * отдельно */
    int img_width, img_height;
    < Переменные для картинки 10 >
    imgs = (IMG_Header *) malloc(sizeof(IMG_Header) * hdr->imagesCount);
    if (imgs == ~) {
        PRINTERR("No memory for image directory of %s.\n", config.picnames[cur_input]);
        return;
    } /* читаем каталог изображений */
    if (fread(imgs, sizeof(IMG_Header), hdr->imagesCount, fpic) != hdr->imagesCount) {
        PRINTERR("Can't read image directory of %s.\n", config.picnames[cur_input]);
        free(imgs);
        return;
    }
    for (cur_image = 0; cur_image < hdr->imagesCount; ++cur_image) {
        img_width = imgs[cur_image].width;
        if (img_width == 0) {
            img_width = 256;
        }
        img_height = imgs[cur_image].height;
        if (img_height == 0) {
            img_height = 256;
        }
        if (imgs[cur_image].bpp != 4) {
            PRINTERR("Bad bits per pixel (%d) for image %d of %s.\n", imgs[cur_image].bpp,
                    cur_image, config.picnames[cur_input]);
            continue;
        }
        if (img_width % 4 != 0) {
            PRINTERR("Bad width (%d) for image %d of %s.\n", img_width, cur_image,
                    config.picnames[cur_input]);
            continue;
        }
        if (imgs[cur_image].size + location > #ffff) {
            PRINTERR("Section size (%d) too big for image %d of %s.\n",
                    imgs[cur_image].size + location, cur_image, config.picnames[cur_input]);
            continue;
        }
        < Обработать одно изображение 11 >
    }
    free(imgs);
}

```

10. Обработать одно изображение.

⟨ Переменные для картинки 10 ⟩ ≡

```
static uint8_t picInData[256 * 256/2];    /* максимальный объем памяти под одно изображение 256
        пикселей в ширину, 256 пикселей в высоту, 2 пиксела в байте */
static uint8_t picOutData[256 * 256/4];
int i, j, k;
uint8_t acc;
```

Этот код используется в секции 9.

11. ⟨ Обработать одно изображение 11 ⟩ ≡

```
PRINTVERB(2, "Image:%d, w:%d, h:%d, colors:%d, planes:%d, bpp:%d, " "size:%d, offset:%x\n",
        cur_image, img_width, img_height, imgs[cur_image].colors, imgs[cur_image].planes,
        imgs[cur_image].bpp, imgs[cur_image].size, imgs[cur_image].offset);
write_label();
fseek(fpic, imgs[cur_image].offset + 40 + 16 * 4, SEEK_SET);
fread(picInData, imgs[cur_image].size, 1, fpic);
```

Смотри также секцию 12.

Этот код используется в секции 9.

12. Переписываем данные из 16-ти цветного формата в 4-х цветный.

⟨ Обработать одно изображение 11 ⟩ +≡

```

k = 0;
if (config.transpose ≡ 0) {
  for (i = img_height - 1; i ≥ 0; --i) {
    for (j = 0; j < img_width/2; ++j) {
      acc = 0;
      acc += recodeColor(picInData[i * img_width/2 + j] & #f) << 2;
      acc += recodeColor((picInData[i * img_width/2 + j] & #f0) >> 4);
      ++j;
      acc += recodeColor(picInData[i * img_width/2 + j] & #f) << 6;
      acc += recodeColor((picInData[i * img_width/2 + j] & #f0) >> 4) << 4;
      picOutData[k++] = acc;
    }
  }
}
else if (config.transpose ≡ 1) {
  for (j = 0; j < img_width/2; j += 2) {
    for (i = img_height - 1; i ≥ 0; --i) {
      acc = 0;
      acc += recodeColor(picInData[i * img_width/2 + j] & #f) << 2;
      acc += recodeColor((picInData[i * img_width/2 + j] & #f0) >> 4);
      acc += recodeColor(picInData[i * img_width/2 + j + 1] & #f) << 6;
      acc += recodeColor((picInData[i * img_width/2 + j + 1] & #f0) >> 4) << 4;
      picOutData[k++] = acc;
    }
  }
}
else {
  for (j = 0; j < img_width/2; j += 4) {
    for (i = img_height - 1; i ≥ 0; --i) {
      acc = 0;
      acc += recodeColor(picInData[i * img_width/2 + j] & #f) << 2;
      acc += recodeColor((picInData[i * img_width/2 + j] & #f0) >> 4);
      acc += recodeColor(picInData[i * img_width/2 + j + 1] & #f) << 6;
      acc += recodeColor((picInData[i * img_width/2 + j + 1] & #f0) >> 4) << 4;
      picOutData[k++] = acc;
      acc = 0;
      acc += recodeColor(picInData[i * img_width/2 + j + 2] & #f) << 2;
      acc += recodeColor((picInData[i * img_width/2 + j + 2] & #f0) >> 4);
      acc += recodeColor(picInData[i * img_width/2 + j + 3] & #f) << 6;
      acc += recodeColor((picInData[i * img_width/2 + j + 3] & #f0) >> 4) << 4;
      picOutData[k++] = acc;
    }
  }
}
write_text(picOutData, k);

```

13.

```
static wint8_t recodeColor(wint8_t col)  
{  
    int i;  
    for (i = 0; i < 4; ++i) {  
        if (col  $\equiv$  config.colors[i]) {  
            return (i);  
        }  
    }  
    return (0);  
}
```

14. \langle Прототипы 14 $\rangle \equiv$

```
static void handleOneFile(FILE *, ICO_Header *);  
static wint8_t recodeColor(wint8_t);
```

Смотри также секции 26 и 28.

Этот код используется в секции 1.

15. Работа с объектным файлом.

Объектный файл состоит из нескольких блоков, для представления картинки понадобятся блоки²

- GSD — для меток картинок и т.д.;
- ENDGSD — конец меток и прочего;
- RLD — хотя картинки и располагаются друг за другом в памяти, иногда придется указывать смещение;
- TXT — собственно данные картинок;
- ENDMOD — конец модуля.

⟨ Глобальные переменные 2 ⟩ +≡

FILE *fobj;

16. Каждый блок начинается байтами 0 и 1, двумя байтами длины блока, а заканчивается байтом контрольной суммы.

```
static void write_block_with_header(uint8_t * data, uint16_t data_len, uint8_t * hdr, uint8_t hdr_len)
{
    uint8_t chksum;
    uint16_t len;
    len = data_len + hdr_len + 4;
    chksum = 0;
    fputc(1, fobj);
    fputc(0, fobj);
    chksum -= 1;
    fwrite(&len, sizeof (len), 1, fobj);
    chksum -= len & #ff;
    chksum -= (len & #ff00) >> 8;
    if (hdr_len ≠ 0) {
        fwrite(hdr, hdr_len, 1, fobj);
        for ( ; hdr_len > 0; --hdr_len) {
            chksum -= *hdr++;
        }
    }
    fwrite(data, data_len, 1, fobj);
    for ( ; data_len > 0; --data_len) {
        chksum -= *data++;
    }
    fputc(chksum, fobj);
}

static void write_block(uint8_t * data, uint16_t data_len)
{
    write_block_with_header(data, data_len, ~, 0);
}
```

² AA-KX10A-TC-PDP-11-MACRO-11-Reference-Manual-May88

17. Записать блок ENDMOD.

```
static void write_endmod(void)
{
    uint8_t buf[2];
    buf[0] = 6; /* ENDMOD */
    buf[1] = 0;
    write_block(buf, sizeof (buf));
}
```

18. Записать блок ENDGSD.

```
static void write_endgsd(void)
{
    uint8_t buf[2];
    buf[0] = 2; /* ENDGSD */
    buf[1] = 0;
    write_block(buf, sizeof (buf));
}
```

19. Записать начальные блоки GSD, которые содержат описания программных секций, имени модуля и пр. Для программной секции оставляем место под неизвестную на этом этапе длину.

```
static void write_initial_gsd(void){ uint16_t buf[9];
    buf[0] = 1; /* GSD */ /* Имя модуля */
    buf[1] = toRadix50("PI");
    buf[2] = toRadix50("C$$");
    buf[3] = buf[4] = 0; /* Программная секция */
    buf[5] = toRadix50(config.section_name);
    buf[6] = toRadix50(config.section_name + 3); /* Тип и флаги секции */
    buf[7] = #500 + °40 + config.save; /* Длина секции */
    buf[8] = location; write_block ( ( uint8_t * ) buf, 9 * 2 ); }
```

20. Записать начальный блок перемещения (RLD).

```
static void write_rld(void){ uint8_t buf[2];
    buf[0] = 4; /* RLD */
    buf[1] = 0;
    buf[2] = 7; /* Location counter definition */
    buf[3] = 0; ( ( uint16_t * ) (buf + 4) ) [0] = toRadix50(config.section_name); ( ( uint16_t * )
    (buf + 4) ) [1] = toRadix50(config.section_name + 3);
    buf[8] = buf[9] = 0; write_block ( ( uint8_t * ) buf, 10 ); }
```

21. Записать метку картинки. Метка получается из шаблона, заданного в командной строке, к которому добавляется номер картинки в десятичной системе счисления. Шаблон усекается так, чтобы имя метки не превысило 6-ти символов.

⟨ Глобальные переменные 2 ⟩ +=

```
static int location = 0;
static int label_count = 0;
```

22.

```
static void write_label(void){ uint16_t buf[5];
    char name[7], label[7];
    int len;
    buf[0] = 1; /* GSD */ /* Имя метки */
    snprintf(label, 6, "%d", label_count++);
    len = strlen(label);
    strcpy(name, config.label);
    name[6 - len] = '\0';
    strcat(name, label);
    buf[1] = toRadix50(name);
    buf[2] = toRadix50(name + 3);
    buf[3] = °150 + 4 * 256;
    buf[4] = location; write_block ( ( uint8_t * ) buf, 5 * 2 ); }
```

23. Записать графические данные картинки.

```
static void write_text(uint8_t * data, int len){ uint16_t hdr[2];
    hdr[0] = 3;
    hdr[1] = location; write_block_with_header (data, len, ( uint8_t * ) hdr, 4 );
    location += len; }
```

24. ⟨Записать начало объектного файла 24⟩ ≡

```
fobj = fopen(config.output_filename, "w");
if (fobj == ~) {
    PRINTERR("Can't open %s.\n", config.output_filename);
    return (ERR_CANTOPENOBJ);
} /* Пропускаем место для начальной GSD */
fseek(fobj, 9 * 2 + 5, SEEK_SET);
write_rld();
```

Этот код используется в секции 1.

25. ⟨Закрывать объектный файл 25⟩ ≡

```
write_endgsd();
write_endmod();
/* Возвращаемся назад и пишем длину секции */
fseek(fobj, 0, SEEK_SET);
write_initial_gsd();
fclose(fobj);
```

Этот код используется в секции 1.

26. ⟨Прототипы 14⟩ +≡

```
static void write_block ( uint8_t * , uint16_t ); static void write_block_with_header ( uint8_t * ,
    uint16_t, uint8_t * , uint8_t );
static void write_endmod(void);
static void write_endgsd(void);
static void write_initial_gsd(void);
static void write_rld(void);
static void write_label(void); static void write_text ( uint8_t * , int );
```

27. Вспомогательные функции.

Упаковка строки в RADIX50.

```

uint16_t toRadix50(char *str)
{
    static char radtbl[] = "_ABCDEFGHIJKLMNOPQRSTUVWXYZ$.0123456789";
    uint32_t acc;
    char *rp;
    acc = 0;
    if (*str == 0) {
        return (acc);
    }
    rp = strchr(radtbl, toupper(*str));
    if (rp == ~) {
        return (acc);
    }
    acc += ((uint32_t)(rp - radtbl)) * °3100;
    ++str;
    if (*str == 0) {
        return (acc);
    }
    rp = strchr(radtbl, toupper(*str));
    if (rp == ~) {
        return (acc);
    }
    acc += ((uint32_t)(rp - radtbl)) * °50;
    ++str;
    if (*str == 0) {
        return (acc);
    }
    rp = strchr(radtbl, toupper(*str));
    if (rp == ~) {
        return (acc);
    }
    acc += ((uint32_t)(rp - radtbl));
    return (acc);
}

```

28. \langle Прототипы 14 $\rangle + \equiv$

```
static uint16_t toRadix50(char *);
```

29. Разбор параметров командной строки.

Для этой цели используется достаточно удобная свободная библиотека *argp*.

```
#define VERSION "0.1"
```

30. <Константы 30> ≡

```
const char *argp_program_version = "ico2obj, "VERSION;
const char *argp_program_bug_address = "<yellowrabbit@bk.ru>";
```

Этот код используется в секции 1.

31. <Глобальные переменные 2> +≡

```
static char argp_program_doc[] = "Convert ICO images to object file";
static char args_doc[] = "file [...]";
```

32. Распознаются следующие опции:

- o — имя выходного файла.
- v — вывод дополнительной информации (возможно указание дважды);
- l LABEL — шаблон метки для изображения (6 символов RADIX50);
- s SECTION_NAME — имя программной секции (6 символов RADIX50);
- a — создавать секции с атрибутом SAV;
- t — транспонировать картинку;
- [0123] — номера цветов для битов.

<Глобальные переменные 2> +≡

```
static struct argp_option options[] = {
    {"output", 'o', "FILENAME", 0, "Output filename"},
    {"verbose", 'v', ~, 0, "Verbose output (-vv --- more debug info)"},
    {"section", 's', "SECTION_NAME", 0, "Program section name"},
    {"attr", 'a', ~, 0, "Set program section SAV attribute"},
    {"label", 'l', "LABEL", 0, "Label for images"},
    {"trans", 't', ~, 0, "Transpose image (-tt --- transpose by word)"},
    {"color0", '0', "COLOR", 0, "Color number for bits 00"},
    {"color1", '1', "COLOR", 0, "Color number for bits 01"},
    {"color2", '2', "COLOR", 0, "Color number for bits 10"},
    {"color3", '3', "COLOR", 0, "Color number for bits 11"},
    {0}
};
static error_t parse_opt(int, char *, struct argp_state *);
static struct argp argp = {options, parse_opt, args_doc, argp_program_doc};
```

33. Эта структура используется для получения результатов разбора параметров командной строки.**<Собственные типы данных 5> +≡**

```
typedef struct _Arguments {
    int verbosity;
    char output_filename[FILENAME_MAX]; /* Имя файла с текстом */
    char label[7]; /* Метка для картинок в объектном файле */
    char section_name[7]; /* Имя программной секции */
    int save; /* установлен атрибут SAV для секции */
    char **picnames; /* Имена файлов картинок picnames[?] == NULL -> конец имен */
    int colors[4]; /* Номера цветов для битов */
    int transpose;
} Arguments;
```

34. $\langle \text{Глобальные переменные 2} \rangle + \equiv$

```
static Arguments config = {0, {0}, {'P', 'I', 'C', 0, 0, 0, 0},
    {'S', 'P', 'I', 'C', 'T', 'L', 0}, 0, ~,
    /* Начальные номера цветов */
    {0, 1, 2, 3}, 0, };
```

35. Задачей данного простого парсера является заполнение структуры **Arguments** из указанных параметров командной строки.

```
static error_t parse_opt(int key, char *arg, struct argp_state *state)
{
    Arguments *arguments;
    arguments = (Arguments *) state->input;
    switch (key) {
        case 't': ++arguments->transpose;
            break;
        case 'a': arguments->save = 1;
            break;
        case 'l':
            if (strlen(arg) == 0 || strlen(arg) > 6) return (ARGP_ERR_UNKNOWN);
            strcpy(arguments->label, arg);
            break;
        case 's':
            if (strlen(arg) == 0 || strlen(arg) > 6) return (ARGP_ERR_UNKNOWN);
            strcpy(arguments->section_name, arg);
            break;
        case 'v': ++arguments->verbosity;
            break;
        case 'o':
            if (strlen(arg) == 0) return (ARGP_ERR_UNKNOWN);
            strncpy(arguments->output_filename, arg, FILENAME_MAX - 1);
            break;
        case '0': arguments->colors[0] = atoi(arg);
            break;
        case '1': arguments->colors[1] = atoi(arg);
            break;
        case '2': arguments->colors[2] = atoi(arg);
            break;
        case '3': arguments->colors[3] = atoi(arg);
            break;
        case ARG_KEY_ARG: /* Имена файлов картинок */
            arguments->picnames = &state->argv[state->next - 1]; /* Останавливаем разбор параметров */
            state->next = state->argc;
            break;
        default: break;
    }
    return (0);
}
```

36.

```

#define ERR_SYNTAX 1
#define ERR_CANTOPEN 2
#define ERR_CANTCREATE 3
#define ERR_BADFILEHEADER 4
#define ERR_CANTOPENOBJ 5
⟨Разобрать командную строку 36⟩ ≡
    argp_parse(&argp, argc, argv, 0, 0, &config);    /* Проверка параметров */
    if (strlen(config.output_filename) == 0) {
        PRINTERR("No output filename specified\n");
        return (ERR_SYNTAX);
    }
    if (config.picnames == ~) {
        PRINTERR("No input filenames specified\n");
        return (ERR_SYNTAX);
    }

```

Этот код используется в секции 1.

37. ⟨Включение заголовочных файлов 37⟩ ≡

```

#include <string.h>
#include <stdlib.h>
#ifdef _linux_
#include <stdint.h>
#endif
#include <argp.h>

```

Этот код используется в секции 1.

38.

```

⟨Глобальные переменные 2⟩ +≡
#define PRINTVERB (level, fmt, a... ) (((config.verbosity) ≥ level) ? printf((fmt), ##a) : 0)
#define PRINTERR (fmt, a... ) fprintf(stderr, (fmt), ##a)

```

39. Индекс.

linux: 37.
_Arguments: 33.
_ICO_Header: 5.
_IMG_Header: 8.
a: 38.
acc: 10, 12, 27.
arg: 35.
argc: 1, 35, 36.
argp: 32, 36.
ARGP_ERR_UNKNOWN: 35.
ARGP_KEY_ARG: 35.
argp-option: 32.
argp-parse: 36.
argp-program_bug_address: 30.
argp-program_doc: 31, 32.
argp-program_version: 30.
argp_state: 32, 35.
args_doc: 31, 32.
Arguments: 33, 34, 35.
arguments: 35.
argv: 1, 35, 36.
atoi: 35.
bpp: 8, 9, 11.
buf: 17, 18, 19, 20, 22.
chksum: 16.
col: 13.
colors: 8, 11, 13, 33, 35.
config: 1, 9, 12, 13, 19, 20, 22, 24, 34, 36, 38.
cur_image: 9, 11.
cur_input: 1, 2, 9.
data: 16, 23.
data_len: 16.
ERR_BADFILEHEADER: 7, 36.
ERR_CANTCREATE: 36.
ERR_CANTOPEN: 4, 7, 36.
ERR_CANTOPENOBJ: 24, 36.
ERR_SYNTAX: 36.
error_t: 32, 35.
fclose: 1, 25.
FILENAME_MAX: 33, 35.
fnt: 38.
fobj: 15, 16, 24, 25.
fopen: 4, 24.
fpic: 1, 3, 4, 7, 9, 11.
fprintf: 38.
fputc: 16.
fread: 7, 9, 11.
free: 9.
fseek: 11, 24, 25.
fwrite: 16.
handleOneFile: 1, 9, 14.
hdr: 1, 6, 7, 9, 16, 23.
hdr_len: 16.
height: 8, 9.
i: 10, 13.
ICO_Header: 5, 6, 9, 14.
imagesCount: 5, 7, 9.
IMG_Header: 8, 9.
img_height: 9, 11, 12.
img_width: 9, 11, 12.
imgs: 9, 11.
input: 35.
j: 10.
k: 10.
key: 35.
label: 22, 33, 35.
label_count: 21, 22.
len: 16, 22, 23.
level: 38.
location: 9, 19, 21, 22, 23.
main: 1.
malloc: 9.
name: 22.
next: 35.
offset: 8, 11.
options: 32.
output_filename: 24, 33, 35, 36.
parse_opt: 32, 35.
picInData: 10, 11, 12.
picname: 1, 4, 7.
picnames: 1, 9, 33, 35, 36.
picOutData: 10, 12.
planes: 8, 11.
PRINTERR: 4, 7, 9, 24, 36, 38.
printf: 38.
PRINTVERB: 7, 11, 38.
radtbl: 27.
recodeColor: 12, 13, 14.
reserved: 8.
rp: 27.
save: 19, 33, 35.
section_name: 19, 20, 33, 35.
SEEK_SET: 11, 24, 25.
size: 8, 9, 11.
snprintf: 22.
state: 35.
static: 32.
stderr: 38.
str: 27.
strcat: 22.
strchr: 27.
strcpy: 22, 35.

strlen: 22, 35, 36.
strncpy: 35.
toRadix50: 19, 20, 22, 27, 28.
toupper: 27.
transpose: 12, 33, 35.
type: 5, 7.
uint16_t: 5, 8, 16, 19, 20, 22, 23, 26, 27, 28.
uint32_t: 8, 27.
uint8_t: 8, 10, 13, 14, 16, 17, 18, 19, 20, 22, 23, 26.
verbosity: 33, 35, 38.
VERSION: 29, 30.
width: 8, 9.
write_block: 16, 17, 18, 19, 20, 22, 26.
write_block_with_header: 16, 23, 26.
write_endgsd: 18, 25, 26.
write_endmod: 17, 25, 26.
write_initial_gsd: 19, 25, 26.
write_label: 11, 22, 26.
write_rld: 20, 24, 26.
write_text: 12, 23, 26.
zero0: 5, 7.

- 〈Включение заголовочных файлов 37〉 Используется в секции 1.
- 〈Глобальные переменные 2, 15, 21, 31, 32, 34, 38〉 Используется в секции 1.
- 〈Данные программы 3, 6〉 Используется в секции 1.
- 〈Закрыть объектный файл 25〉 Используется в секции 1.
- 〈Записать начало объектного файла 24〉 Используется в секции 1.
- 〈Константы 30〉 Используется в секции 1.
- 〈Обработать одно изображение 11, 12〉 Используется в секции 9.
- 〈Открыть файл картинки 4〉 Используется в секции 1.
- 〈Переменные для картинки 10〉 Используется в секции 9.
- 〈Проверить заголовок картинки 7〉 Используется в секции 4.
- 〈Прототипы 14, 26, 28〉 Используется в секции 1.
- 〈Разобрать командную строку 36〉 Используется в секции 1.
- 〈Собственные типы данных 5, 8, 33〉 Используется в секции 1.

ICO2OBJ

	Секция	Страница
Общая схема программы	1	2
Обработать один файл картинки	8	4
Обработать одно изображение	10	6
Работа с объектным файлом	15	9
Вспомогательные функции	27	12
Разбор параметров командной строки	29	13
Индекс	39	16