Библиотека для работы с файлами EPPL7. Руководство программиста

В.Б. Вагнер

29 сентября 1998 г.

Оглавление

Введение

Библиотека предназначена для работы с векторными и растровыми файлами EPPL7 из программ на языке C. Библиотека расчитана на работу в 32-разрядной среде. В принципе небольшие программы способны работать и в 16-разрядной среде, но поскольку специальных мер для этого не принималось, работа в моделях памяти Compact, Large и Huge не гарантируется.

В библиотеках предусмотрен код для работы на процессорах с прямым порядком байт (MSB First), хотя он пока не оттестирован.

Тем не менее не рекомендуется использовать читать файлы EPPL7 непосредственно, так как при этом придется обращать внимание на порядок байтов в слове.

Поскольку оригинальный пакет EPPL7 работал под DOS, все его файлы, хранящиеся в двоичном формате используют порядок байтов процессора Intel --- младший байт первым.

Библиотеки не используют системных вызовов и библиотечных функций, специфичных для UNIX[1].

При разработке библиотек предполагалось, что тип `long int` имеет размер 32 бита, `short int` --- 16 бит и `double` --- 64 бита, в формате, удовлетворяющем стандарту IEEE.

[1] За исключением утилит для работы с файловой системой и обрабоки прерывания программы. Эти функции локализованы в модуле `file_utils.c` и описаны в главе 4

Глава 1

Логическая модель файлов данных

1.1 Растр

Теоретически, растровая карта является функцией от координат, значения которой являются числами или элементами некоторого конечного множества, определенной внутри некоторой геометрической фигуры на декартовой координатной площади.

Практически как область определения, так и область значений квантуются, а целью библиотеки является возможность произвольным образом задавать значения растра в любой нужной точке.

Поэтому, растровые файлы EPPL7 (epp) рассматриваются как большая матрица коротких (16-бит) целых. Размеры матрицы могут достигать 30000×30000 а значения - принимать любое значение, допустимое для типа `unsigned short` т.е. 0--65535.

Соответствие между вещественными картографическими координатами и целочислеными индексами в матрице (рядами и колонками) хранится в заголовке epp-файла, и в библиотеке предусмотрены функции для пересчета одних в другие.

Номера рядов/колонок в файле не обязательно начинаются с единицы. Для них допустимы любые значения в диапазоне -32767--+32767.

то позволяет в файле, покрывающем небольшной участок территории иметь совместимость с большим файлом не только по альтернативным координатам, но и по номерам рядов/колонок.

Обычно ячейки в файле квадратные, хотя все процедуры библиотеки будут корректно работать с файлами, в которых размеры ячейки по вертикали и по горизонтали различны.

начение площади ячейки вместе с размерностью (квадратные метры, футы, гектары и др.) хранится в заголовке файла. Линейные единицы альтернтативных координат нигде не описываются, но для совместимости не рекомендуется использовать никаких единиц кроме метров или градусов (если файл хранится в географических координатах).

При этом площадь ячейки можно перевести в другие единицы, соответственно изменив код размерности, для того чтобы получить более удобочитаемые числа при рассчете площадей.

Преобразование множества значений карты в множество значений (классов) epp-файла традиционно считается функцией легенды, и находится за пределами формата epp, а, следовательно и данной библиотеки.

Поскольку матрица всегда прямоугольна, а область определения карты --- не всегда, предусмотрено специальное значение, называемое offsite.

При чтении epp-файла предполагается, что размеры его бесконечны, но попытка прочитать его элемент за прелами реального файла приводит к возврату значения offsite, которое означает, что в данной точке значение не определено. начение offsite свое для каждой карты. Оно хранится в заголовке файла и в соответствующем поле структуры данных, используемой для доступа к файлу.

Большая часть алгоритмов, используемых для генерации растровых карт или для их анализа, не зависит от порядка обхода ячеек карты, хотя и требует полного их перебора. В этих случаях рекомендуется использовать *последовательный* перебор по строкам, как самый экономичный по памяти.

Другой распространенный вариант доступа, это когда при обработки конкретной ячейки растра требуется информация о значениях, содержащихся в соседних ячейках.

В этом режиме доступ к каждой строке растра требуется несколько раз --- сначала при обработке одной или нескольких предыдущих строк, потом при обработке самой этой строки, и наконец, при обработке следующих строк. Для таких ситуаций реализован *кэшированный* доступ, при котором несколько строк в распакованном виде хранятся в памяти.

И, наконец, для таких приложений как интерактивные редакторы, реализован *прямой доступ на чтение/запись*. тот способ доступа существенно медленнее, чем кэшированный, поскольку строки epp-файла хранятся в памяти в упакованном виде.

Для упрощения последовательного перебора ячеек существуют функции-*итераторы*, которые получают в качестве параметра функцию и последовательно применяют ее ко всем ячейкам файла.

1.2 Вектор

Векторные файлы EPPL7 (dgt) представляют собой последовательнсть объектов --- линий и точек.

Установление соотвествия между объектом и координатами представляет для них существенно более сложную задачу, чем установление соответствия между координатами и классами растра.

Для большинства способов обрабоки векторных файлов тоже достаточно последовательного перебора объектов.

В тех случаях, когда требуется, например, перебор всех пар объектов, можно загрузить векторный файл в память.

Для векторных файлов тоже существуют итераторы, позволяющие применить переданную функцию ко всем линиям, ко всем точкам или вообще ко всем объектам в файле.

Логическая структура векторного файла EPPL7 такова:

В его заголовке определяется прямоугольная область в картографических (альтернативных) координатах, которую покрывает файл.

Внутри файла все координаты хранятся во внутренних единицах, для которых всегда нижняя и левая границы файла имеют значение -32767, а верхняя и правая --- $+32767$. то обеспечивает максимально возможную точность и компактность, но приводит к проблемам при расчете

расстояний и углов, и применении других функций аналитической геометрии, поскольку величина единичного отрезка по оси X и по оси Y не совпадают.

Все объекты в векторном файле имеют целочисленный (32-битный) идентификатор. Недопустимым значением для идентификатора объекта является 0, поскольку точка с нулевыми координатами и нулевым идентификатором используется как признак конца файла.

Векторный файл EPPL7 может хранить два типа объектов: линии и точки.

Точка имеет две координаты и идентификатор.

Для линии хранится идентификатор, координаты минимального прямоугольника, заключающего в себе линию, число точек и набор пар координат.

Количество точек в линии не может превышать 500.

Функции библиотеки заботятся о том, чтобы координаты объемлющего прямоугольника (маски линии) были корректными.

Глава 2

Функции доступа к epp файлам

Функции переменные, необходимые для доступа к epp-файлам описаны в файле `epp.h`. В нем описана структура данных `EPP`, хранящая информацию об открытом epp-файле.

Внутреннее устройство этой структуры описано в разделе 5.2.

Практически лезть внутрь этой структуры при программировании прикладных задач нет необходимости. Вся работа с ней производится с помощью функций библиотеки.

та структура данных хранит наиболее существенную информацию из заголовка EPP-файла. начения тех полей которые не хранятся в этой структуре, можно прочитать или изменить с помощью функций, описанных в разделе 2.6.

2.1 Поля структуры EPP

В этом разделе описаны только те поля, значения которых достаточно часто используются при реализации прикладных алгоритмов. Остальные поля и принципы работы с ними описаны в разделе 5.2. Все поля, про которые не сказано обратного, можно только читать. Для их изменения используйте соответствующие функции, поскольку изменение значений некоторых полей может потребовать согласованного изменения каких-то других.

`int fr,lr,fc,lc` — Номера первой строки и колонки, а также значения на единицу больше последней строки и колонки. На единицу больше они потому, что в этом случае конструкции `lc - fc` и `lr - fr` дают количество строк и колонок в файле. Для изменения используется функция `shift_epp`

`double Xleft,YBottom,XRight,YTop` — альтернативные (карографические) координаты границ карты.

`int offsite` — значение offsite. Если файл открыт на запись, это значение можно менять.

`double cell_area` — площадь ячейки в единицах, указанных в заголовке файла. См. раздел 2.6.

`int kind` — количество бит на ячейку. Может принимать значения 8 или 16.

Информация из заголовка файла epp, которая не попала в вышеперечисленный список полей, в структуре EPP не хранится. Для работы с ней нужно получить полную копию заголовка файла с помощью функции `get_epp_header`.

`i` — n

t mode режим, в котором открыт файл. Комбинация констант `MAP_INPUT` и `MAP_OUTPUT`.

2.2 Открытие файла и режимы доступа

```
EPP* open_epp( char *filename );
```

Открывает файл с именем `filename` и возвращает указатель на структуру EPP. Файл должен существовать. Режим открытия устанавливается в `MAP_INPUT`.

```
EPP* fopen_epp(FILE *f);
```

Открывает файл аналогично `open_epp`, но получает в качестве парамера не имя, а указатель на структуру FILE. то позволяет использовать для передачи растровой информации не только именованные файлы на диске. Дескриптор файла f должен допускать позиционирование.

```
EPP *creat_epp(char *pathname, int first_col, int first_row, int last_col,
          int last_row, double AXLeft,double AYTop, double AXRight, double
          AYBottom, int scale, int base, int offsite);
```

Создает новый epp-файл и открывает его в режиме `MAP_OUTPUT`.

При создании файла указываются его размеры, альтернативные координаты, значение offsite и значения полей заголовка `base` и `scale`.

Для создания epp файле в уже открытом файле UNIX существует функция `fcreat_epp`, которой вместо имени файла передается указатель на структуру `FILE`.

Во многих случаях нужно создать выходной файл того же размера и с той же координатной системой, что и входной. Для упрощения этого процесса существует функция

```
EPP *creat_epp_as(char *filename,EPP *pattern);
```

которая создает файл, копируя большую часть информации заголовка из уже существующего файла, указатель на который передается в параметре `pattern`. Для нее также существует аналог, получающий не имя, а открытый файл.

Все эти функции по умолчанию создают 8-битный epp-файл. Точнее, количество бит определяется значением глобальной переменной `Create16bit`, которой нужно присвоить ненулевое значение для того, чтобы создавались 16-битные файлы

2.3 Изменение режимов доступа

При использовании данной библиотеки существующий файл всегда открывается в режиме последовательного чтения, а новый --- в режиме последовательной записи. Для того, чтобы получить другой режим доступа нужно менять режим доступа явным образом.

```
void reset_epp(EPP *epp)
```

Переокрывает для чтения файл, открытый для последовательной записи. Все незаполненные строки заполняются offsite.

```
int set_epp_cache(EPP* epp, int lines);
```

Создает кэш на `lines` строк. Если кэш ранее существовал, изменяет его размер. `set_epp_cache(epp,0)` отменяет кэширование совсем.

```
int load_epp(EPP* epp);
```

агружает файл, открытый для чтения в память и меняет его режим на `MAP_INPUT|MAP_OUTPUT`,, т.е. прямое чтение-запись.

```
int save_epp(EPP *epp);
```

сохраняет изменения в загруженном файле на диск. Старое содержимое при этом теряется.

```
int save_epp_as(EPP *epp,char *newname);
```

Сохраняет файл под новым именем. Старый файл при этом закрывается.

```
int fsave_epp_as(EPP *epp,FILE *f);
```

Аналогична `save_epp_as`, но получает не имя файла, а уже открытый файл.

Функции сохранения не меняют режима файла. Просто больше некуда было их поместить.

Все функции сохранения и функция `load_epp` после обработки каждой строки вызывают функцию, на которую указывает глобальная переменная `EndLineProc` (см. раздел 2.5). Ее результат они игнорируют.

```
void close_epp(EPP *epp);
```

акрывает файл. Если файл был открыт на последовательную запись, все изменения сохраняются, а незаполненные строки заполняются offsite.

Если файл был загружен в память, автоматического сохранения **не** производится.

2.4 Доступ к ячейкам файла

Основным способом доступа к ячейкам файла являются функции

```
int epp_get(EPP* epp,int x,int y);
```

и

```
void epp_put(EPP*epp,int x,int y,int value);
```

Функция `epp_get` возвращает значение ячейки с указанными координатами (во внутренних единицах файла) или offsite, если данная точка находится за пределами карты.

Ее нельзя применить к файлу, открытому для последовательной записи, при этом возникает ошибка `ME_INVALID_MODE`.

При работе с файлом открытым для последовательного чтения, опрашивать ячейки можно в любом порядке, но обращение к другой строке требует ее считывания с диска и распаковки.

При работе с кэшированным или загруженным в память файлом, тем более можно обращаться к его ячейкам в произвольном порядке.

Функция `epp_put` при попытке изменить ячейку за пределами файла генерирует ошибку `ME_POINT_OUTSIDE`.

В режиме последовательной записи можно записывать точки внутри текущей строки в произвольном порядке, а при обращении к строке с номером, большим чем текущая, текущая строка сбрасывается на диск, и дальнейшая запись в нее невозможна. Если произошла запись в строку с номером n, в то время как предыдущая текущая строка имела номер m, где $m < n - 1$, все строки от $m + 1$ до $n - 1$ заполняются offsite.

Попытка записи в строку, уже записанную на диск, вызывает ошибку `ME_INVALID_PUT`.

В файлах, загруженных в память, то есть доступных и для чтения и для записи, этих ограничений нет.

Для специальных целей, преимущественно для целей визуализации, реализована функция

```
unsigned short *epp_get_line(int x,int y);
```

возвращающая текущую строку как массив `unsigned short int`. Она возвращает указатель на внутренний буфер структуры EPP, поэтому освобождать память не нужно.

Количество элементов массива, имеющих осмысленные значения, вызывающая программа должна определить сама.

Попытка вызвать эту функцию при `x` $<$ `epp->fc` приводит к ошибке `ME_POINT_OUTSIDE`, а функция возвращает `NULL`.

2.5 Итераторы

```
int for_each_cell(EPP *epp,EPP_ITER_PROC action);
```

Выполняет функцию `action` для всех не-offsite ячеек существующего файла или для всех ячеек файла, открытого для последовательной записи. Возвращает 0, если выполнение завершено успешно, -1 если выполнение прервано в результате ненулевого результата `EndLineProc` и -2, если выполнение прервано самой функцией `action`

Тип `EPP_ITER_PROC` определен как

```
typedef int (*EPP_ITER_PROC)(int col, int row, int value);
```

Параметры `col` и `row` передают координаты текущей точки, параметр `value` содержит класс точки для существующего файла и offsite для создаваемого.

Возвращаемое значение в случае создаваемого файла записывается в ячейку.

В случае существующего файла нулевое значение соответствует нормальному состоянию, а ненулевое приводит к прекращению работы.

```
long count_cells(EPP *epp, EPP_ITER_PROC condition);
```

Возвращает количество ячеек в существующем файле, удовлетворяющих заданному условию. Функция `condition` должна возвращать 0 для ячеек, которые условию не удовлетворяют и ненулевое значение для тех, которые удовлетворяют. В случае прерывания работы, возвращает те же коды, что и `for_each_cell`

Параметры, передаваемые функции `condition` такие же, как у `action` в случае существующего файла.

Итераторы выполняются достаточно долго, поэтому для них предусмотрен способ выдачи прогресс-индикатора. Если глобальной переменной `EndLineProc` присвоено значение, отличное от `NULL`, то функция, на которую она указывает, вызывается после окончания обработки каждой строки файла.

Функция получает три целых параметра:

`row` — номер строки epp-файла, с учетом значения поля `fr`

`seqno` — порядковый номер строки (от 1 до `lr-fr`)

`total` — общее число строк в файле.

Если эта функция возвращает ненулевое значение, обработка файла прерывается.

Таким образом простейшая функция индикации прогресса (дающая результат, очень похожий на поведение EPPL7 в аналогичной ситуации), выглядит так:

```
int show_progress(int row,int seqno,int total)
{
 fprintf(stderr,"\rProcessing row %d of %d",seqno,total);
 fflush(stderr);
 return 0;
}
```

2.6 Работа с заголовком epp-файла

аголовок epp-файла имеет следующую структуру:

```
typedef struct EPPHEADER
 { short int fr,lr,fc,lc;
   double fry,lry,fcx,lcx;
   short int kind;
   short int base,scale;
   unsigned short int offsite;
   double sfact;
   long access_ptr;
   unsigned short  minclass, maxclass;
   char area_unit;
   char coord_sys;
   char Reserved[6];
   char date[16], time[8];
   char comment[32];
 } EPPHEADER;
```

Семантика этих полей следующая:

`fr,lr,fc,lc` — совпадают с соответствующими полями структуры `EPP`

`fry,lry,fcx,lcx` — альтернативные координаты

`kind` — количество бит на ячейку

`base,scale` — тайна, покрытая мраком. Я так и не смог уяснить из документации к EPPL7, что эти поля делают, и не знаю, какие команды EPPL, не считая `HEADER` с ними работают.

`offsite` — хранит значение offsite.

`sfact` — площадь ячейки. См. также поле `area_unit`

`access_ptr` — смещение (в 128 байтовых блоках) таблицы длин строк в файле

`minclass,maxclass` — минимальный и максимальный класс в файле (не считая offsite

`area_unit` — Единица измерения площади.

`coord_sys` — тип проекции

`date,time` — дата и время создания файла в текстовом формате.

`comment` — краткий комментарий к файлу (заголовок карты). Значение этого поля EPPL7 запрашивает у пользователя при создании каждого файла как "Description"

Возможные значения единиц площади и типов проекции описаны как константы в файле `eppl.h`, который включается и файлом `epp.h` и файлом `dgt.h`

AREA_NO_UNIT Площадь ячейки не определена;

AREA_SQ_FOOT Квадратные футы;

AREA_SQ_METER Квадратные метры;

AREA_SQ_KM Квадратные километры;

AREA_SQ_MILE Квадратные мили (видимо, имеются в виду английские, а не морские);

AREA_HECTARE Гектары;

AREA_ACRE Акры.

Как видите, миллиметры не предусмотрены.

Количество типов проекций очень невелико и показывает ориентацию EPPL7 на крупномасштабное картографирование.

COORD_NONE Координатная система не определена (свежее сканерное изображение);

COORD_UTM Американская топографическая карта. (от Гаусса-Крюгера отличается на 0.2%, но отличается.

COORD_STPLANE Проекция State Plane. Используется, судя по названию, для карт штатов С А.

COORD_LL Географические координаты.

Для работы с этой информацией предусмотрены следующие функции:

```
void get_epp_header(EPP* epp, EPPHEADER *h);
```

Возвращает заголовок файла в виде вышеописанной структуры. Используется для чтения полей, к которым нет более удобного доступа.

```
char *getcomment(EPP *epp);
```

Возвращает комментарий, обрезая концевые пробелы. Возвращает указатель на статический буфер, который будет перезаписан при следующем обращении к этой функции.

```
void change_epp_header(EPP* epp,EPPHEADER h);
```

Копирует в заголовок файла поля `fcx, lcx, fry, lry, base, scale, offsite, sfact, coord_sys, area_unit, comment` из переданной структуры. Остальные поля оставляет как были.

```
void setcomment(EPP *epp,char *comment);
```

Изменяет только комментарий. В отличие от `change_epp_header` не требует, чтобы строка была дополнена пробелами до 32 символов. Ее аругумент --- обычная NULL-terminated строка.

```
int shift_epp(EPP* epp, int new_fr, int new_fc);
```

Изменяет значения полей `fr` и `fc`, и соответственно `lr` и `lc`, поскольку число строк и колонок в файле поменять нельзя.

Возвращает ненулевое значение в случае успеха или 0, если какие-то координаты при изменении вылезли за допустимый диапазон.

2.7 Пересчет координат

```
int epp_row(EPP *epp,double y);
```

Возвращает номер строки по альтернативному y.

```
int epp_col(EPP *epp,double x);
```

Соответственно, номер колонки по альтернативному x.

```
double alt_x(EPP *epp,int col);
```

Возвращает альтертантивный x указанной колонки.

```
double alt_y(EPP *epp,int row);
```

И альтернативный y указанной строки.

ти две функции возвращают альтернативные координаты правого верхнего угла ячейки. Для получения координат центра ячейки предназначены функции:

```
double alt_xc(EPP *epp,int col);
```

```
double alt_yc(EPP *epp,int row);
```

2.8 Наложение файлов

Растровые файлы легко накладывать друг на друга, если у них совпадает размер ячейки и координатная система.

Для проверки этих условий в библиотеку включена функция

```
int compare_cell_size(EPP *f1,EPP *f2);
```

она возвращает 0 если размеры ячеек несовместимы и ненулевой код, если файлы можно накладывать.

Второе условие, которое необходимо проверить --- совпадает ли правило пересчета альтернативных координат в ряды/колонки.

Для этого предназначена функция

```
int is_aligned(EPP *f1,EPP *f2);
```

Она возвращает ненулевой код, если обращение к `epp_get` с одинаковыми аргументами для обоих файлов возвращает информацию об одной и той же точке.

Несовпадение рамки файлов не играет никакой роли, поскольку `epp_get` корректно обрабатывает обращение к точке за пределами файла.

Если `compare_cell_size` возвращает TRUE, а `is_aligned` --- FALSE, то файлы могут быть выравнены путем изменения номера первой строки и/или колонки с помощью функции `shift_epp`.

Если же необходимо совместно использовать файлы, у которых не совпадает размер ячейки, то необходимо расчитать и использовать пересчетные коэффициенты.

Для этого используются функции:

```
EPP_LINK link_epp(EPP *base,EPP *overlay);
```

Устанавливает связь между файлами. Возвращает указатель на структуру `LINK_BUFFER`, содержащую набор пересчетных коэффициентов или `NULL`, если связь установить невозможно (координатные системы двух файлов имеют противоположно направленные оси).

Функции:

```
int linked_row(EPP_LINK link,int row);
```

```
int linked_col(EPP_LINK link,int col);
```

пересчитывают ряды/колонки файла `base` в ряды/колонки файла `overlay`

2.9 Прочее

В некоторых случаях до окончания создания файла нельзя определить, есть ли в нем классы свыше 255. В этих случаях приходится всегда создавать 16-битный файл, а по окончании работы, если max<=255 конвертировать его в восьмибитный. Для упрощения (и ускорения) этой операции, в библиотеку включена функция:

```
int fast_convert_to_8bit(EPP *source, char *filename);
```

та функция получает открытый на последовательную запись файл, переоткрывает его на чтение и копирует в новый файл, указанный как filename.

Функция работает так быстро, как только возможно, поскольку использует напрямую некоторые возможности библиотеки, не вынесенные в интерфейс.

Для сведения читателей, имеющих опыт работы с epp-файлом на низком уровне, укажем, что запаковки она не выполняет вообще, а распаковывает только младшие 8 бит 16-битного файла.

По окончании работы закрывает оба файла.

2.10 Обработка ошибок

Большая часть функций библиотеки корректно обрабатывает ошибочные ситуации. Достаточно часто индикатором ошибки служит результат, возвращаемый функцией.

Существует глобальная переменная `map_error` в которую помещается код ошибки. Возможные коды определены как константы в файле `epp_err.h`.

`ME_OK` — операция завершилась успешно.

`ME_POINT_OUTSIDE` — Попытка сделать `epp_put` за пределами файла.

`ME_INVALID_MODE` — Попытка читать карту, открытую в режиме последовательной записи, или модифицировать карту, открытую в режиме последовательного чтения

`ME_READ_ERROR` — Ошибка системного вызова или библиотечной функции при чтении с диска. Рекомендуется посмотреть значение глобальной перемнной `errno`.

`ME_WRITE_ERROR` — То же самое, но при записи.

`ME_INVALID_PUT` — Попытка изменить ячейку, уже записанную на диск.

`ME_OUT_OF_MEMORY` — Не хватило памяти для размещения необходимых структур данных

`ME_ACCESS_DENIED` — Не удалось открыть файл в требуемом режиме

`ME_NO_FILE` — Файл с указанным именем не существует

`ME_INVALID_FILE` — Файл не является правильным epp или dgt файлом

`ME_CREATE_ERROR` — Ошибка создания файла, не существует путь или недопустимое имя. Подробности — в системной переменной `errno`.

Если при какой-то операции `map_error` установлена в ненулевое значение, то имеет смысл перед продолжением работы ее обнулить. Некоторые функции библиотеки проверяют его после вызова других функций, а обнуление после успешной операции не гарантировано.

Глава 3

Функции доступа к векторным файлам

Функции доступа к векторным файлам описаны в заголовочном файле `dgt.h`. В нем определена структура `DGT`, аналогичная по смыслу структуре `EPP`.

Поскольку в заголовке векторных файлов никакой информации кроме альтернативных координат и типа проекции не хранится, способов прямой работы с заголовком не предусмотрено.

3.1 Типы данных

В отличие от растровых файлов, значением которых в точке является обычное целое число, векторные файлы хранят довольно сложные объекты.

Приведем их описания:

`POINT` — Структура для хранения координат точки. Имеет два поля типа `signed short int` — `x` и `y`. Несмотря на схожесть описания со структурой `XPoint`, попытка передать массив точек типа `POINT` в функцию tt XDrawLine почему-то к успеху не приводит

`DGT_ITEM` — Структура для хранения информации о векторном объекте — полилинии или точке (не путать точку-объект с точкой-узлом линии)). Ее поля:

`long int ID` — 32-битный идентификатор объекта.

`short int x1,y1` — левый нижний угол маски, если объект — полилиния или координаты точки, если объект — точка.

`short int x2,y2` — правый верхний угол маски. У точек — не используется.

`shott int npoints` — число узлов в линии. Если 0, то объект — точка, от 2 до 500 — полилиния, остальные значения недопустимы.

`POINT s[500]` — массив координат точек полилинии.

Структура `DGT` хранит всю информацию, необходимую при работе с векторным файлом. Программисту имеет смысл непосредственно читать следующие ее поля:

`double XLeft,YBottom,XRight,YTop` — альтернативные координаты границ файла.

`int mode` — Режим доступа. Комбинация тех же констант `MAP_INPUT` и `MAP_OUTPUT`, что и поле `mode` структуры `EPP`.

`unsigned char projection` — Тип координатной системы. Возможные значения описаны в разделе 2.6.

`DGT_ITEM *buffer` — Буфер для хранения текущего объекта. Обычно доступ к этой структуре выполняется с помощью макросов, описанных в разделе 3.6.

`int eof` — Флаг, выставляемый в ненулевое значение, если прочитан признак конца файла.

`int item_no` — Порядковый номер текущего объекта

3.2 Открытие файла

```
DGT *open_dgt(char *filename)
```

```
DGT *fopen_dgt(FILE *f)
```

Открывают существующий файл для последовательного чтения. Возвращают указадель на структуру `DGT` или `NULL`, если произошла ошибка.

```
DGT *creat_dgt(char *filename, double x_left, double y_bottom, double
        x_right, double y_top, unsigned char proj)
```

```
DGT *fcreat_dgt(FILE *f, double x_left, double y_bottom, double x_right,
        double y_top, unsigned char proj)
```

```
DGT *mcreat_dgt(double x_left, double y_bottom, double x_right, double
        y_top, unsigned char proj)
```

Создают файл для последовательной записи. Получают в качестве параметров пределы альтернативных координат и тип проекции.

Особое внимание стоит обратить на функцию `mcreat_dgt`, которая создает пустой файл в памяти, открытый на чтение-запись. Подробнее работа с файлами в памяти описана в разделе 3.7

Для открытия файлов с пределами и проекцией аналогичными уже существующему файлу, существуют функции:

```
DGT *creat_dgt_as(char *filename,DGT *pattern);
```

```
DGT *fcreat_dgt_as(FILE *f,DGT *pattern);
```

```
DGT *mcreat_dgt_as(DGT *pattern);
```

3.3 Смена режима и закрытие файла

Функция

```
void reset_dgt(DGT* dgt);
```

делает текущим объектом первый. Если файл до этого был открыт в режиме последовательной записи, он переоткрывается в режиме последовательного чтения.

```
int load_dgt(DGT *dgt);
```

агружает файл в память, делая возможным прямой доступ на чтение-запись и многие другие возможности, описанные в разделе 3.7

```
void close_dgt(DGT *dgt);
```

акрывает файл, освобождая всю выделенную под него память. Изменения в последнем объекте сохраняются, если файл был открыт для последовательной записи.

3.4 Чтение и запись объектов

После открытия существующего файла, первый объект автоматически считывается в буфер, и доступен с помощью макросов для работы с буфером (см. раздел 3.6).

Для перехода к следующему объекту вызывается макрос

```
void dgt_next(DGT *dgt);
```

В некоторых случаях удобно работать с копиями объектов, размещенных в динамической памяти. В этом случае можно использовать функцию

```
DGT_ITEM *dgt_get(DGT *dgt)
```

Которая размещает текущий объект в памяти, перемещается на следующий объект и возвращает указатель копию объекта, бывший текущим до ее вызова.

Для записи в файл можно использовать либо функцию

```
void dgt_put(DGT *dgt, DGT_ITEM *item);
```

либо модифицировать текущий объект с помощью макросов и функций для работы с текущей записью и затем вызвать `dgt_next` для перехода к новому объекту.

3.5 Итераторы

Поскольку векторные файлы хранят два типа объектов --- линии и точки, и только очень немногие алгоритмы (в основном трансформация координат) затрагивают и те и другие, определены итераторы:

```
for_each_line(DGT *dgt,void (*item_proc)(DGT *dgt), DGT *outstream);
```

```
for_each_point(DGT *dgt,void (*item_proc)(DGT *dgt), DGT *outstream);
```

Они вызывают указанную функцию только для линий или точек файла. Функции передается указатель на структуру `DGT` в буфере которой находится объект.

ти итераторы можно применять к файлам, открытым для последовательного чтения или к файлам загруженным в память.

Если параметр `outstream` не `NULL`, то в файл, на который он указывает, копируются все объекты, которые не обрабатывались и обработанные объекты.

То есть, например, в случае `for_each_line` точка будет скопирована без обработки, а линия записана после того, как ее изменит `item_proc`.

Кроме этого, существует итератор

```
void for_each_item(DGT *dgt,void (*item_proc)(DGT *dgt));
```

который выполняет переданную функцию и для линий и для точек. У него нет параметра `outstream`, так что если нужно записать измененные объекты, `item_proc` должна об этом позаботиться сама.

3.6 Работа с текущим объектом

Итак, нужная вам линия (точка) находится в буфере структуры `DGT`. то можно делать с ней дальше?

Во-первых, нужно уметь определять, линия это или точка. Для этого определены макросы:

```
int dgt_is_line(DGT *dgt)
```

```
int dgt_is_point(DGT *dgt)
```

Во-вторых, независимо от того, линия это или точка, нам может потребоваться идентификатор объекта.

```
long int dgt_id(DGT *dgt)
```

В-третьих, координаты точки:

```
short int dgt_pointx(DGT *dgt)
```

```
short int dgt_pointy(DGT *dgt)
```

или координаты прямоугольника-маски линии:

```
short int dgt_xl(DGT *dgt)
```

```
short int dgt_yb(DGT *dgt)
```

```
short int dgt_xr(DGT *dgt)
```

```
short int dgt_yt(DGT *dgt)
```

В-четвертых, количество точект в линии:

```
short int dgt_line_len(DGT *dgt)
```

И, наконец, координаты i-того узла линии:

```
short int dgt_nx(DGT *dgt,int i)
```

```
short int dgt_ny(DGT *dgt,int i)
```

Обратите внимание, что первая точка линии имеет номер 0, а последняя ---`dgt_line_len(dgt)-1`.

Кроме того, иногда удобнее получить ссылку на точку или узел линии, как на структуру `POINT`. Для этого предназначены макросы:

```
POINT dgt_node(DGT *dgt,int i)
```

```
POINT dgt_point(DGT *dgt)
```

Если вам по каким-то причинам нужно работать не со структурой DGT, а с отдельным объектом, размещенным в динамической памяти, то замените префикс dgt_ на префикс item_ и в качестве первого параметра указывайте указатель на структуру DGT_ITEM.

Теперь мы можем прочитать любой параметр объекта. Вопрос в том, как его изменить.

Ответ первый, парадоксальный: любой из вышеприведенных макросов (кроме dgt_is_line и dgt_is_point) можно использовать в **левой** части оператора присваивания.

Ведь это не функции, а макросы, дающие доступ к некоторым полям структуры DGT_ITEM.

Однако так поступать не рекомендуется. Дело в том, что в таком случае нигде не будет отмечено, что объект изменен, поэтому изменения могут быть не сохранены. Если уж очень хочется, то после прямого изменения полей объекта, используйте функцию

```
void dgt_touch(DGT *dgt)
```

чтобы пометить объект как измененный.

Но лучше пользоваться следующими функциями:

```
void dgt_set_id(DGT *dgt,long int newID);
```

Устанавливает идентификатор объекта.

```
void dgt_set_point(DGT *dgt, int x, int y);
```

адает координаты точки и делает объект точкой. Если вы хотите сохранить одну координату, вместо нее передайте константу KEEP_OLD_VALUE

```
int dgt_add_node(DGT *dgt,int x,int y);
```

Добавляет точку в конец линии (или создает первую точку и помечает объект как линию. Возвращает 0 в случае, если все в порядке или ненулевое значение, если точку добавить невозможно (уже есть 500 точек).

```
int dgt_ins_node(DGT *dgt,int index,int x,int y)
```

Вставляет точку в линию перед узлом номер index. Возвращает 0, если все в порядке, 1 если число точек превысит 500, и 2 если узла с таким номером нет.

```
int dgt_mv_node(DGT *dgt,int index,int x,int y)
```

Меняет координаты указаннокго узла на заданные. Возвращает 0, если все в порядке и ненулевое значение, если такого узла нет.

Если вам надо изменить только одну координату, передайте вместо второй константу KEEP_OLD_VALUE.

```
int dgt_rm_node(DGT *dgt,int index)
```

Удаляет указанный узел. Ошибка возникает, если это был предпоследний узел в линии. (Корректная линия должна содержать не менее двух узлов).

И, наконец, разрезание линий.

```
int dgt_split(DGT *dgt,int index)
```

Разрезает линию в указанном узле. При этом узел дублируется и попадает в обе половинки. Текущим становится хвост линии, а начало, если файл был открыт в режиме последовательной записи, сбрасывается на диск. той функцией нельзя пользоваться из итератора for_each_line, если он применяется к файлу, открытому в режиме последовательного чтения.

```
int dgt_cut_segment(DGT *dgt,int index)
```

Разрезает линию, удаляя отрезок от $n-1$ до n-ного узла. Все остальное в точности как в предыдущем случае.

Процедура `dgt_next` позаботится о некоторых из ошибок, которые можно допустить при редактировании объекта.

1. Маска линии будет скорректирована.
2. Из двух соседних узлов с совпадающими координатами будет оставлен один.
3. Линии из одной точки будут выброшены.
4. Точка с нулевым идентификатором будет выброшена.

3.7 DGT-файлы в памяти

Векторный файл, загруженный в память, можно рассматривать как массив объектов. Последовательный доступ продолжает работать, но существуют функции, позволяющие сделать текущим объектом любой, независимо от порядка и обращаться к объектам, не делая их текущими.

Последний способ имеет существенный недостаток --- если вы хотите изменить длину линии, то вам придется самому позаботиться о выделении памяти под новую линию.

Итак, процедуры, специфичные для файла в памяти:

```
int dgt_seek(DGT *dgt,int index);
```

Делает текущим объект с указанным номером. (после чего можно спокойно его менять с помощью любых функций и макросов предыдущего раздела.)

```
DGT_ITEM* dgt_item_ptr(DGT *dgt,int index);
```

Возвращает указатель на объект с указанным номером. Для доступа к этому объекту можно использовать макросы `item_....`

тот способ доступа рекомендуется использовать как доступ только для чтения. В крайнем случае, если необходимо изменить не текущую линию, воспользуйтесь функцией:

```
int dgt_replace(DGT *dgt,int index,DGT_ITEM* new_item);
```

та функция получает указатель на динамически размещенный объект и помещает его в файл. Для того, чтобы сэкономить память, размещайте объект с помощью функции

```
DGT_ITEM *alloc_item(DGT_ITEM *item);
```

та функция получает указатель на объект, выделяет памяти ровно столько, сколько нужно, чтобы разместить этот объект, и копирует его туда.

Функции вставки объекта в указанном месте в библиотеке нет. При добавлении объекта с помощью `dgt_put` его место определяется в соответствии с текущим порядком сортировки. Если порядок сортировки вообще не задавался, то это будет следующий объект после текущего, и он станет текущим.

Порядок сортировки dgt файла в памяти можно изменить при помощи функции

```
void dgt_sort(DGT *dgt,int (*compare_func)(DGT_ITEM* i1, DGT_ITEM * i2));
```

Функция `compare_func` должна возвращать 1, если i1 надо разместить позже, чем i2, -1 если наоборот, и 0, если по данному критерию сортировки эти объекты равны.

Изменение критерия сортировки при уже загруженном файле приводит к его пересортировке с помощью функции `qsort`.

Поэтому лучше вызывать `dgt_sort` **непосредственно перед** `load_dgt`.

Вызов `dgt_sort(dgt,NULL)` приводит к отмене сортировки. Порядок расположения существующих объектов не меняется, но новые добавляемые объекты будут располагаться без учета функции сортировки.

3.8 Работа с координатами

Для пересчета координат из внутреннего представления dgt-файла в альтернативные и обратно применяются функции:

```
int dgt_x(DGT* dgt,double x);

int dgt_y(DGT* dgt,double y);

double real_x(DGT* dgt,int x);

double real_y(DGT* dgt,int y);
```

Достаточно часто возникает задача пересчета интервалов, которая может быть выполнена с меньшим числом операций, чем вычисление собственно координат. Для этого определены функции

```
double real_dx(DGT *dgt,int dx);

double real_dy(DGT *dgt,int dy);
```

И, наконец, вычисление расстояний.

```
double dgt_dist(DGT *dgt, int x1, int y1, int x2, int y2);
```

Вычисляет расстояние между двумя точками заданными парами внутренних координат в единицах альтернативных координат.

Для вычисления расстояния между двумя узлами линии можно использовать функцию

```
double dgt_pdist(DGT *dgt, POINT p1, POINT p2);
```

Во многих алгоритмах расстояния используются только для сравнения между собой (и, возможно, с некоторыми константами). В этих случаях можно избежать не только перевода расстояния в осмысленные единицы, но и извлечения корня и свести все вычисления к операциям с целыми.

Для этого предназначены функции

```
long int sq_dist(DGT *dgt,int x1, int y1, int x2, int y2);

long int sq_pdist(DGT *dgt,POINT p1, POINT p2);
```

вычисляющие квадрат расстояния между точками в некоторых условных единицах. (известно, что квадрат при $x >= 0$ --- функция монотонная).

Для того, чтобы иметь возможность сравнивать результаты функции `sq_dist` с константами, заданными изначально как расстояния в альтернативных координатах, используется функция

```
long int sq_const(DGT *dgt,double dist);
```

Она переводит расстояние, заданное в единицах альтернативных координат, в условное значение, аналогичное возвращаемым `sq_dist`

Кроме расстояний, несоответствие единиц координат по осям влияет и на на вычисление углов. Поэтому в библиотеку включена функция

```
double dgt_atan2(DGT *dgt,int dy, int dx);
```

вычисляющая арктангенс отношения $\frac{\Delta y}{\Delta x}$ с поправкой на координатную систему указанного файла. В остальном ее действие полностью эквивалентно библиотечной функции `atan2`.

Глава 4

Функции-утилиты

В библиотеку включен ряд функций, которые не работают непосредственно с файлами EPPL7, но полезны при написании маленьких командно-строчных программ для работы с этими файлами.

В отличие от остальной библиотеки, эти функции системно-зависимы. При переносе библиотеки в другую операционную среду их требуется переписать полностью и может быть даже изменить их набор.

Например, стандартные библиотеки большинства компиляторов C под DOS и Windows не включают функции `getopt`, которая активно используется в Environmental Planning Utilities.

Функции этой группы описаны в заголовочном файле `<eppl_ut.h>`.

4.1 Обработка имен файлов

```
char *default_ext(const char *name, const char *ext);
```

проверяет, оканчивается ли имя файла на указанное расширение. Если нет, то добавляет его.

Очевидно, что в файловых системах FAT и HPFS, где несколько расширений недопустимы, эта функция должна проверять наличие не данного, а любого расширения.

```
char *force_ext(const char *name, const char *ext);
```

заменяет последнее расширение на указанное, или добавляет это расширение, если имя файла не содержало точки.

```
char *last_ext(const char *name);
```

Возвращает последнее расширение файла или `NULL`, если у файла нет расширения.

Все эти три функции возвращают указатель на статический буфер, который будет перезаписан при следующем обращении к ним.

```
FILE *lookup_file(const char *name, const char *sufux, const char *dir);
```

Ищет файл с указанными именем в текущей директории, в ее поддиректории `dir`, если таковая имеется и в поддиректории `dir` системной библиотечной директории `/usr/local/lib/fgis`.

Возвращает указатель на структуру `FILE`. Файл открывается только для чтения.

Функция предназначена для поиска вспомогательных файлов, таких как палитры и штриховки.

4.2 Прогресс-индикатор

В библиотеку также включена стандартная функция вывода сообщения `Processing row ... of ....`

```
int show_progress(int rowno,int seqno,int nrows);
```

Она выводит это сакраментальное сообщение на `stderr`. Кроме этого, данная функция обрабатывает `SIGINT`, поэтому прерывание программы по Ctrl-C приводит к корректному ее завершению.

Она имеет аналог

```
int show_percent(int rowno,int seqno,int nrows);
```

который выводит сообщение вида `99.9% completed`. В отличие от `show_progress` эта функция может использоваться и для dgt-файлов. Кроме того, на больших файлах она несколько экономит время, поскольку пытается вызывать медленные операции ввода-вывода, только если есть прогресс по крайней мере на 0.1%, что может составлять до 30 строк.

В то же время даже на файлах максимального размера большая часть операций будет выполняться с такой скоростью, что за 0.1% времени работы пользователь не успеет решить, что программа зависла.

Кроме того, существует функция

```
int check_int(int rowno,int seqno,int nrows);
```

которая ничего не выводит, а только проверяет сигнал. Рекомендуется всегда иметь возможность использовать ее в качестве альтернативы `show_percent`, например для фонового выполнения.

Устанавливается одна из этих функций с помощью функции

```
void install_progress_indicator(int (*func)(int,int,int));
```

Функция

```
int clear_progress(int result);
```

Стирает сообщение, сделанное функцией --- прогресс-индикатором и выводит сообщение `Done`, если `result = 0` или `Aborted`.

Возвращает `result`, поэтому логичный способ ее применения такой:

```
install_progress_indicator(show_percent);
if ((res=clear_progress(for_each_cell
                        (infile,my_iterator_func))))
  unlink(out_file_name);
close_epp(infile);
close_epp(outfile);
return res;
```

Глава 5

Внутреннее устройство библиотеки

5.1 Устройство растрового файла EPPL7

5.2 Работа объекта EPP

5.3 Векторные объекты

5.4 Векторные файлы в памяти

Предметный указатель