Приложение С. Описание стандартных модулей

В стандартной библиотеке SYSTEM.UPL содержатся следующие модули:

- UniCRT - модуль работы с экраном в текстовом режиме;
- UniDOS - модуль связи с операционной системой;
- UniGRAPH - модуль использования графического экрана;
- UniLEX - модуль лексической обработки текста.

Их описательные (interface) части находятся, соответственно, в файлах: **UNICRT.DOC**, **UNIDOS.DOC**, **UNIGRAPH.DOC** и **UNILEX.DOC**. Файлы упакованы и находятся в файле **EXAMPLES.ARC**. Чтобы распаковать их, введите в командную строку UniDOS-а следующее (лучше всего используйте два дискета):

```
B:\>unarc a:examples XXX.DOC
```

где **XXX** - имя модуля, описание которого надо распаковать.

Эти файлы предоставляются только для удобства. Они уже компилированы и записаны в библиотеке.

С.1. Описание модуля UniCRT

Модуль UniCRT предоставляет средства удобного использования возможностей BIOS-а для работы с экраном микрокомпьютеров фамилии Пылдин 601/601A/601M (601M эквивалентен 601A).

Экспортируются следующие константы, представляющие собою коды соответствующих цветов в текстовом режиме:

- Black = 0; Черный
- Blue = 1; Синий
- Green = 2; Зеленый
- Cyan = 3; Сине-зеленый
- Red = 4; Красный
- Magenta = 5; Фиолетовый
- Brown = 6; Коричневый
- LightGray = 7; Светло серый
- DarkGray = 8; Темно серый
- LightBlue = 9; Голубой
- LightGreen = 10; Ярко зеленый
- LightCyan = 11; Ярко сине-зеленый
- LightRed = 12; Ярко красный
- LightMagenta = 13; Ярко фиолетовый
- Yellow = 14; Ярко желтый
- White = 15; Белый
- Blink = 128; Значение, добавляемое к коду основного цвета для получения мерцающего изображения.

procedure ActvCrsr(size: ShortCard);

Действие: Переключает активный курсор (тот, который появляется во время ввода с клавиатуры) в один из возможных размеров: значение 0 - курсор не виден; значения 1, 2, 3, 4 обозначают, соответственно, что курсор имеет размер в $^{1}/_{4}$, $^{2}/_{4}$, $^{3}/_{4}$ и $^{4}/_{4}$ из своего максимального размера. По умолчанию активный курсор имеет размер в $^{1}/_{4}$ из своего максимального размера (1).

procedure ClrEol;

Действие: Стирает все символы (замещая их пробелами) с текущей позиции курсора до конца строки без перемещения курсора.

procedure ClrEos;

Действие: Стирает все символы с текущей позиции курсора до конца экрана без перемещения курсора.

procedure ClrScr;

Действие: Очищает экран и устанавливает курсор в его верхнем левом углу.

procedure Delay(ms: cardinal);

Действие: Выжидает прохождения указанного времени, определенного параметром в миллисекундах.

function GetColor: ShortCard;

Действие: Результатом этой функции является текущий атрибут цветности вывода символов на экран. Он имеет тот же самый смысл, как и для процедуры SetColor.

procedure GotoXY(X, Y: ShortCard);

Действие: Перемещение курсора экрана в позицию (X, Y). Левый верхний угол экрана имеет координаты (1, 1), при том координата X увеличивается слева направо, а координата Y сверху вниз.

function KeyPressed: boolean;

Действие: В качестве результата возвращает TRUE, если с момента последнего чтения с клавиатуры до момента ее активизации была нажата клавиша. Другими словами, проверяется есть ли что-то в буфере клавиатуры.

procedure PassCrsr(size: ShortCard);

Действие: Переключает пассивный курсор (тот, который появляется, когда не ожидается ввод с клавиатуры, т.е. во время работы программы) в одном из возможных размеров: значение 0 - курсор не виден; значения 1, 2, 3, 4 обозначают, соответственно, что курсор имеет размер в $^1/_4$, $^2/_4$, $^3/_4$ и $^4/_4$ из своего максимального размера. По умолчанию пассивный курсор не виден (0).

procedure PlayMelody(var Melody);

Действие: Воспроизведение мелодии. Мелодия составлена из последовательности нот (см. описание PlayNote). Каждым элементом последовательности является либо нота (занимающая 3 байта), либо команда для изменения тембра, либо конец (один байт). Если элементом является нота, то ее первый байт носит информацию об ее номере, второй - об ее длительности, а третий - о длительности паузы за нотой. Изменение тембра (от 1 до 7) задается содержимым (от $F8 до $FE) одного байта. Конец мелодии задается содержимым $FF одного байта.

procedure PlayNote(Note, Tembr: ShortCard; length: cardinal);

Действие: Воспроизводит звучание ноты. Note - номер ноты от 0 до 72. Номеру 0 соответствует пауза, номерам от 1 до 72 соответствуют номера нот, при том номеру 29 соответствует нота ЛА первой октавы. Tembr принимает значения от 1 до 7 и интерпретируется как тембр. Length - длительность звучания ноты в единицах по 1.536 миллисекундам.

function ReadKey: char;

Действие: Ожидает нажатия клавиши и возвращает ее код. Коды клавишей описаны в Приложении D.

function ScrXsize: byte;

Действие: В качестве результата возвращает горизонтальний размер экрана, т.е. допустимое число символов строк.

function ScrYsize: byte;

Действие: В качестве результата возвращает вертикальный размер экрана, т.е. допустимое число строк на экране.

procedure SetColor(Attribute: ShortCard);

Действие: Устанавливает новый цвет выводимых на экран символов. На Пылдине 601 выполнение процедуры не имеет последствий. Ее действие проявляется только на Пылдине 601A/601M и только, если установлено цветное изображение. Цвета, которые можно использовать, описываются константами. Основное правило, которое нужно соблюдать - это то, что цвет для символов может быть только из первых 8 цветов, а цвет для фона может быть из всех 15 цветов. Атрибут цвета задается формулой 16 * цвет букв + цвет фона. Если необходимо получить мерцающее изображение, добавьте к так полученному атрибуту константу Blink.

procedure TextMode(Mode: ShortCard);

Действие: Переключает экран в одном из следующих режимов: черно-белое изображение с 40-символьными строками; черно-белое изображение с 80-символьными строками; цветное изображение с 40-символьными строками. В качестве фактического параметра используйте одну из следующих экспортированных констант:

```
BW40 = 0;    { черно-белое 40x25 }
BW80 = 4;    { черно-белое 80x25 }
CO40 = 5;    { цветное 40x25 }
```

function WhereX: ShortCard;

Действие: В качестве результата возвращает текущее положение X курсора, т.е. номер столбца, в котором он находится.

function WhereY: ShortCard;

Действие: В качестве результата возвращает текущее положение Y курсора, т.е. номер строки, в которой он находится.

C.2. Описание модуля UniDOS

Модуль UniDOS предоставляет пользователю средства для удобного использования возможностей операционной системы для работы с файлами.

Модулем экспортируются следующие константы, представляющие собою маски для обработки файловых атрибутов:

- ReadOnly = $01; файл только для чтения;
- Hidden = $02; скрытый файл;
- SysFile = $04; системный файл;
- VolumeId = $08; идентификатор диска;

- Directory= $10; поддиректория;
- Archive = $20; архивирован ли файл;
- AnyFile = $3f; маска для обнаружения какого либо файла.

Экспортируются и следующие типы, необходимые некоторым процедурам и функциям:

```
type PathStr = string[79];
     ComStr = string[127];
     DirStr = string[67];
     NameStr = string[12];
     ExtStr = string[4];
       SearchRec = packed record
       Name:    packed array [1..11] of char;
       Attr:    byte;
       tmp:        packed array [1..10] of byte;
       Time:    longint;
       StClust: word;
       FLen:    longint;
       Buff:  packed array [1..$19] of byte;
     end { SearchRec };
     DateTime = record
       Year, Month, Day,
       Hour, Min, Sec: word;
     end { DateTime };
```

После каждого выполнения процедуры или функции необходимо проверять при помощи функции DOSerr произошли ли ошибки ввода/вывода (коды ошибок даны в Приложении Е.3).

procedure ChDir(const s: string);

Действие: Устанавливает новую текущую поддиректорию.

function DiskFree(drv: byte): longint;

Действие: В качестве результата возвращает число свободных байтов носителя, находящегося в устройстве с номером, определяемым параметром. Номера устройств следующие: 0 - текущее устройство; 1- устройство A:, 2 - B: и т.д.

function DosErr: word;

Действие: В качестве результата возвращает код ошибки последней использованной процедуры или функции из этого модуля.

function DosVers: word;

Действие: В качестве результата возвращает версию операционной системы. Старший байт содержит главная (major) версия, а младший - подверсию (minor).

procedure Erase(const s: string);

Действие: Уничтожает файл, чье имя задано параметром. Эта процедура возвращает код ошибки через IOresult, а не через DOSerr.

Ограничение: Файл должен быть закрытым в этот момент.

procedure ErrText(var s: ComStr);

Действие: Возвращает в **s** текст ошибки последней использованной процедуры или функции из этого модуля. Код ошибки должен быть меньше 128 (т.е. ошибка

операционной системы). Если код больше 128, это - код ошибки файловой системы UniPascal-я. Никакая из процедур (функций) этого модуля (UniDOS) не вызывает ошибку файловой системы UniPascal-я.

procedure FindFirst(const Path: PathStr; Attr:word; var F:SearchRec);

Действие: Ищет первого файла, чье имя соответствует имени, заданному параметром Path. Attr - маска, при помощи которой проводится поиск файлов. Например, пусть необходимо найти файл с именем MYFILE.*. Не смотря на то объявлен ли он как файл только для чтения (read only) или нет, параметры нужно установить как следует: Path = 'MYFILE.*' и ATTR = ReadOnly. Если хотим обнаружить все файлы, начинающиеся с A, нужно установить Path = 'A*.*' и ATTR = AnyFile. Переменная-параметр F получит результат: имя обнаруженного файла, его атрибуты и т.д. Этот результат необходимо сохранить, чтобы мог быть использован для следующего поиска процедурой FindNext. Если обнаружен требуемый файл, DOSerr возвращает 0, а если не обнаружен, она возвращает код ошибки, из-за которой он не обнаружен.

procedure FindNext(var F: SearchRec);

Действие: Эта процедура используется для обнаружения следующего появления файла после применения процедуры FindFirst. Ее действие аналогично тому процедуры FindFirst, но ей не надо задавать имя и атрибуты, так как они уже были заданы во время активизации FindFirst. Параметром F должна быть та же самая переменная (типа SearchRec), которая была задана процедуре FindFirst. Например, допустим, что необходимо отпечатать содержимое директории, т.е. хотим реализовать команду DIR операционной системы. Фрагмент программы вывода имен всех файлов имеет вид:

```
...
var F: searchrec;
begin
  ...
  FindFirst('*.*', AnyFile, F);
  while DosErr = 0 do begin          { обрабатываем, если обнаружен файл }
    WriteDirEntry(f);                  { обработка, т.е. печать имени файла }
  FindNext(F);                           { обнаружение следующего файла }
  end { while };
  ...
end.
```

Здесь не приведен текст процедуры WriteDirEntry, а многоточие является обозначением, что там находятся другие части программы.

procedure GetDir(drv: byte; var s: DirStr);

Действие: Переменной **s** присваивается путь (Path) текущей поддиректории для устройства с номером drv.

procedure GetFAttr(const s: string; var Attr: word);

Действие: Переменной **Attr** присваивается атрибут файла с именем, заданным параметром **s**. Для более удобной обработки атрибута можно использовать экспортированные константы.

Ограничение: Файл должен быть закрытым.

procedure GetFtime(const s: string; var Time: longint);

Действие: Переменной **Time** присваивается время (дата и час), когда сделана последняя модификация файла. Для распаковки можно использовать процедуру UnpackDT.

Ограничение: Файл должен быть закрытым.

procedure MkDir(const s: string);

Действие: Создает поддиректорию с именем и пути, определенными параметром **s**.

procedure PackDT(var T: DateTime; var p: longint);

Действие: Упаковка времени для SetFtime.

procedure Rename(const old, new: string);

Действие: Меняет старое (old) имя файла на новое (new). Эта процедура возвращает код ошибки через IOresult, а не через DOSerr.

Ограничение: Файл должен быть закрытым.

procedure RmDir(const s: string);

Действие: Уничтожает заданную поддиректорию.

procedure SetFattr(const s: string; var Attr: word);

Действие: Задает новый (меняет) атрибут для файла **s**.

Ограничение: Файл должен быть закрытым.

procedure SetFtime(const s: string; var Time: longint);

Действие: Задает новое (меняет) время для файла **s**.

Ограничение: Файл должен быть закрытым в этот момент.

procedure UnpackDT(var p: longint; var T: DateTime);

Действие: Распаковывает полученное выполнением процедуры GetFtime время.

С.3. Описание модуля UniGRAPH

Модуль UniGRAPH предоставляет пользователю средства для удобного использования графических возможностей микрокомпьютеров фамилии Пылдин 601/601A/601M (601M эквивалентен на 601A).

Графический экран микрокомпьютера Пылдин 601/601A/601M можно разделить в зависимости от их разрешающих способностей в следующие три вида (в скобках дана разрешающая способность Пылдина 601A/601M):

- высокая разрешающая способность:320x200(640x200), 2 цвета;
- средняя разрешающая способность:160x200(320x200), 4 цвета;
- низкая разрешающая способность: 80x200(160x200), 16 цветов.

Разрешающая способность характеризируется допустимым числом точек строки экрана, допустимым числом строк на экране и числом цветов, при помощи которых можно нарисовать (изобразить, оцветить) одну точку. Данные выше значения для различных видов графических изображений интерпретируются так: в режиме средней разрешающей способности имеются 200 строк, содержащие 160 (320) точек, и возможность использовать 4 цвета для изображения каждой точки.

Кодированная информация, определяющая показываемое изображение на графическом экране микрокомпьютера (независимо от разрешающей способности)

располагается в оперативной памяти и занимает 8 или 16 килобайтов, соответственно, для Пылдина 601 и Пылдина 601А/601М.

В режиме высокой разрешающей способности можно использовать только два цвета - черный и белый, в режиме низкой разрешающей способности - 16 цветов, а в режиме средней разрешающей способности - только 4 цвета, но выбираемые из одной палитры (выбор палитры и цвета делается отдельно). Номера цветов совпадают с теми номерами, заданными в модуле UniCRT. Палитры следующие:

1 палитра	черный(0),	зеленый(2),	красный(4),	желтый(6);
2 палитра	синий(1),	сине-зеленый(3),	фиолетовый(5)	белый(7);
3 палитра яркые:	серый(8),	зеленый(10),	красный(12) ,	желтый(14);
4 палитра яркые:	синий(9),	сине-зеленый(11),	фиолетовый(13),	белый(15).

Независимо от разрешающей способности, координатная система одна и та же (координаты являются логическими). Графическая система компьютера (находящаяся в XBIOS-е) делает преобразование логических координат в физические и выполняет черчение на экране. Верхний левый угол экрана имеет координаты (0, 0), а нижний правый угол - (639, 399), т.е. экран имеет размеры (в точках) 640x400 при использованием логических координат.

Выписание символов возможно и в графическом режиме с высокой или средней разрешающей способностью. Выписывание символов происходит автоматическим образом (при помощи WRITE) на графическом экране, если он является активным. Программист ничего не обязан делать, т.е. механизм прозрачен. Кроме READ/WRITE, в графическом режиме можно использовать все процедуры модуля UniCRT. Графический курсор (который невидим, т.е. он не индицируется как текстовой курсор) и текстовой курсор не являются одним и тем же курсором, т.е. текущая графическая позиция и текущая текстовая позиция независимы. Например: gotoxy(10, 10); и moveto(40, 40); вызывает следующее: текст будет выписыван с позиции (текстовой) (10, 10), а черчение, например, линии начинается с позиции (графической) (40, 40).

function ActualXsz: cardinal;

Действие: В качестве результата возвращает физический размер экрана по горизонтали (допустимое число точек строки) или 0, если не установлен графический режим.

function ActualYsz: cardinal;

Действие: В качестве результата возвращает физический размер экрана по вертикали (допустимое число строк) или 0, если не установлен графический режим.

function AllctCharSet(Npg: shortcard; free: boolean): pointer;

Действие: Резервирует память для дефиниции Npg * 32 символов на границе 256 байтов (как требует XBIOS), т.е. младший байт начального адреса будет содержать 0. Возможна дефиниция 32 последовательных символов, начинающихся с симола, код которого один из следующих: $00, $20, $40, $60, $80, $a0, $c0, $e0. Дефиниция каждого символа состоит из содержимого 8 байтов (по одному для каждой строки графического изображения символа). Следовательно, дефиниция 32 символов состоит из 256 последовательных байтов (по 8 байтам каждому символу). Эта функция только отводит область памяти для этих символов и возвращает ее начальный адрес. Запись в область дефиниции должно произойти позже (например, чтением с файла). Так как память для дефиниции резервируется на 256-байтовой границе, в большинстве случаев отведеная область содержит сверх необходимого числа байтов (но лишнее число байтов не превышает 255). Если параметр FREE = TRUE, неиспользованная память (зарезервированные лишние байты) освободится функцией AllctCharSet сразу и может быть использована (этим способом образуется "дыра" в

динамической памяти). Если параметр FREE = FALSE, неиспользованная памят не освободится.

function AllctScreen(free: boolean): pointer;

Действие: Резервирует необходимое количество памяти для графического экрана (8/16 килобайтов). Если уже существует графический экран, ничего не делает. При резервировании памяти для графического экрана отведенное число байтов обычно превосходит необходимое (с не более чем 15 байтов) из-за выравнивание на 16-байтовой границе (это необходимо из-за аппаратной особенности). Если параметр FREE = TRUE, неиспользуемая область освобождается сразу для использования. Если параметр FREE = FALSE, освобождение просходит в момент освобождения графического экрана.

function GraphSize: cardinal;

Действие: В качестве результата возвращает размер области, необходимой для размещения графического экрана. Результат зависит от компьютера: 8 килобайтов, при Пылдине 601 и 16 при Пылдине 601A/601M.

procedure DefaultCharSet;

Действие: Восстанавливает стандартные дефиниции символов (такие, какие они были до выполнения программы).

procedure DrawBar(X, Y: cardinal);

Действие: Рисует заполненный прямоугольник, один верх которого определяется текущим положением курсора, а другой - заданными координатами, которые могут быть абсолютными или относительными в зависимости от установленного процедурой SetCoordType вида.

procedure DrawCircle(rX, rY: cardinal);

Действие: Рисует эллипс, центром которого является текущее положение курсора, радиусом по горизонтали - rX и радиусом по вертикали - rY.

procedure DrawDisk(rX, rY: cardinal);

Действие: Рисует заполненный эллипс, центром которого является текущее положение курсора, радиусом по горизонтали - rX и радиусом по вертикали - rY.

procedure DrawShape(var Shape; Rot, Scale: shortcard);

Действие: Выполняет заданную последовательность действий: передвижение графического курсора без рисования и передвижение с рисованием. Изображение получается последовательным вычерчиванием отрезка линии. Для этой цели нужно дать указания о новом положении курсора в полярных координатах: направление и длина отсечки от курсора до новой точки. Длина задается числами от 1 до 16 (в расстояниях между двумя соседними логическими точками). Направление определено движением часовой стрелки, а размер угла задается числами от 0 до 7 (в единицах по 45°). Параметром Shape определяется последовательность из элементарных движений (передвижение или передвижение с рисованием), необходимых для получения желанного изображения. Эти данные расположены в **N** последовательных байтов. Им предшествует одно слово, содержащее это число **N**. Число предполагается записанным в порядке: старшая часть, младшая часть (в представлении чисел в UniPascal-е эти части расположены наоборот). Каждый из следующих **N** байтов задает новое положение графического курсора без или с вычерчиванием отрезка линии. Информация об этом действии записывается в байте, как следует. Содержимое 1 7-ого бита служит указанием

на то, что курсор должен рисовать, а 0 - передвигаться без рисования. Биты от 6 до 4 определяют угол вращения, а биты от 3 до 0 - размер отсечки (0 - 1 единица, 1 - 2 единицы, ..., 15 - 16 единиц).

Параметром ROT задается вращение определенного параметром SHAPE и еще ненарисованного изображения углом, определяемым числом от 0 до 7 (тем же способом как при элементарных движениях).

Параметром SCALE задается увеличение (масштабирование) определенного параметром SHAPE и еще ненарисованного изображения при помощи числа от 0 до 15 (0 означает, что изображение определенного параметром SHAPE, не будет масштабированным, 1 - удваивается длина каждого отрезка, ..., 15 - длина отрезка умножается 16).

procedure FreeScreen;

Действие: Освобождает занятую графическим экраном область памяти, если она была резервированной той же самой программой, иначе ничего не делает.

function GetVmode: word;

Действие: В качестве результата возвращает слово, содержимое старшего байта которого представляет собою видеорежим, а младшего - атрибут/палитру.

procedure LineTo(newX, newY: word);

Действие: Рисует отрезок от текущего положения курсора до точки с заданными координатами, которые могут быть абсолютными или относительными в зависимости от установленного процедурой SetCoordType вида.

procedure MoveTo(newX, newY: word);

Действие: Перемещает графический курсор в новое положение. Координаты могут быть абсолютными или относительными в зависимости от установленного процедурой SetCoordType вида.

function Ncolors: shortcard;

Действие: В качестве результата возвращает число допустимых цветов: 2 при высокой разделительной способности, 4 при средней, 16 при низкой, 16 при цветном текстовом экране, 2 при черно-белом текстовом экране.

procedure Point(X, Y: word);

Действие: Изображает точку, имеющую координаты X, Y. Координаты могут быть абсолютными или относительными в зависимости от установленного процедурой SetCoordType вида.

function RelCoord: boolean;

Действие: В качестве результата возвращает вид используемых координат: TRUE при относительных и FALSE при абсолютных.

procedure SetCoordType(relative: boolean);

Действие: Устанавливает вид координат, которые будут использоваться другими процедурами - абсолютные или относительные. Относителные координаты задаются так, как будто началом координатной системы является текущая позиция курсора. Значением TRUE параметра устанавливаются относительные координаты, а FALSE - абсолютные.

procedure SetCharSet(pages: CharPgSet; addr: pointer);

Действие: Процедура устанавливает для пользования другими процедурами дефинированные пользователем символы в графическом режиме, отменяя стандартные. Параметром PAGES задается множество, в котором должны быть определены номера групп по 32 символам, чья дефиниция будет использована. Например, следующий фрагмент программы позволяет нам дефинировать 3 группы по 32 символам (группа 0 - $00..$1F, группа 6 - $C0..$Df и группа 7 - $E0..$FF) и использовать их:

```
...
var ChSetBuff: pointer;
begin
...
ChSetBuff:= AllctCharSet(3, true {/false});
DefineCharSet;                    { процедура, созданная программистом для
                                  дефинирования символов, например, чтением с дискета }
SetCharSet([0, 6, 7], ChSetBuff);                    { с этого момента при
                  использовании WRITE для изображения символов в графическом
      режиме будут использованы символы, дефинированные пользователем }
...
```

procedure SetGRcolor(color: byte; mode: byte);

Действие: Устанавливает цвет при рисовании (COLOR) и способ рисования (MODE). Параметром COLOR устанавливается для пользования другими процедурами цвет (число, допустимое для использованного видеорежима). Параметром MODE устанавливается способ, по которому будет выполняться рисование одной точки в зависимости от существующего и заказываемого состояния этой точки на экране. Значения, которые MODE может принимать, числа от 0 до 15. Каждое число представляет собою код, которым определяется функция (N), которая надо применить над существующим состоянием (S) и заказываемым состоянием (P) при ее использовании другими процедурами. Значение 0 и 1 ее аргументов P и S обозначают, соответственно, свечение и несвечение точки.

- m_draw = 0; { N = P }
- m_xor = 1; { N = S xor P }
- m_set = 2; { N = 1 }
- m_notP = 3; { N = not P }
- m_clear = 4; { N = 0 }
- m_none = 5; { N = S }
- m_reverse = 7; { N = not S }
- m_or = 8; { N = S or P }
- m_notSandP = 9; { N = not S and P }
- m_notSorP = 10; { N = not S or P }
- m_not_SandP= 11; { N = not (S and P) }
- m_SandnotP = 12; { N = S and not P }
- m_and = 13; { N = S and P }

procedure SetVmode(ModeNo, Pallette: shortcard);

Действие: Переключает указанный видеорежим. Номер видеорежима (ModeNo) экспортируется как константа и может принимать значения, имеющие следующий смысл:

- text_bw40 = 0; { черно-белый/цветной текстовой экран 40x25 }

- graph_lo = 1; { графика 80(160)x200, 16 цветов }
- graph_mid = 2; { графика 160(320)x200, 4 цвета }
- graph_hi = 3; { графика 320(640)x200, 2 цвета }
- text_bw80 = 4; { черно-белый текстовой экран 80x25 (601A) }

Параметром Pallette задается номер палитры для графического экрана с средней разрешающей способностью, номер цвета для графического экрана с низкой разрешающей способностью или атрибут для текстового экрана размером 40x25. В остальных случаях этот параметр игнорируется. Если заказано включение графического режима, но для графического экрана не резервировано место, процедура ничего не делает.

function WhereGRx: cardinal;

Действие: В качестве результата возвращает текущее положение X (по горизонтали) графического курсора.

function WhereGRy: cardinal;

Действие: В качестве результата возвращает текущее положение Y (по вертикали) графического курсора.

С.4. Описание модуля UniLEX

Модуль UniLEX предоставляет пользователю удобные процедуры и функции для лексической обработки текста, которые являются частью интерпретатора Y кода. Лексическая обработка связана с выделением слов, распознаванием чисел и др. Этот модуль используется и компилятором UniPascal-я. Программист имеет возможность задавать свои символы, которые могут образовать слова, свои разделители и т.д.

Основная и самая важная задача лексического анализа - это разбивка входного потока символов в отдельные лексемы. Например, если в силе синтаксис языка Pascal, следующий входной поток

x:= 1 + 2 * y - z / 2

преобразуется в поток следующих лексем:

идентификатор(x), присваивание, число(1), плюс, число(2), звездочка, идентификатор(y), минус, идентификатор(z), косая_черточка, число(2)

Анализ лексем существенно проще выполняется в сравнении с анализом входного потока (текста). Задача о разбивки входного потока в лексемы обычно решается при помощи подпрограммы. Такая подпрограмма работает как конечный автомат, на входе которого подается входной поток, а на выходе получается очередная выделенная лексема. В интерпретаторе Y кода есть программная модель конечного автомата на языке ассемблера и поэтому она работает значительно быстрее в сравнении с какой либо программой, которая была бы на языке UniPascal. Эта программная модель конечного автомата организована как функция, называемая TokenSearch (поиск лексемы). Ее входные параметры определяют входной поток, который она должна обрабатывать, и правила, по которым распознаются лексемы.

Для TokenSearch входной поток задается частьями в буфере, который должен быть заполненным частью входного потока (например, символы одной строки текста находятся в буфере). Символы входного текста разделяются в 8 классов (A, B, C, A_1, B_1, C_1, D, E), а слова - в 3 класса (A, B, C). Понятие "слово" здесь обозначает последовательность символов.

- класс символов A, B, C - здесь входят символы, с которых могут начинаться слова классов слов A, B, C, соответственно;
- класс символов A_1, B_1, C_1 - здесь входят символы, которые могут содержаться в словах с второго до их последнего символа классов слов A, B, C, соответственно;
- класс символов D - здесь входят все разделители, которые пропускаются (обычно - это пробел и табуляция);
- класс символов E - здесь входят специальные символы - начало специального слова, обрабатываемого программистом отдельно (обычно - это числа: целые и вещественные).

При активизации TokenSearch сначала пропускаются все разделители (символы входного потока, которые принадлежат классу D). Потом анализируется (распознавается) лексема, следующая за разделителями. Результат функции описывается перечисляемым типом TOKEN (лексема) и его константы имеют следующий смысл:

- InvalidCh - первый символ заданного входного потока не приналдежит ни одному из возможных классов или никакое слово не было выделено (символы класса C и C_1, но нету зарезервированного слова в таблице класса C);
- UserCh - первый символ (за разделителями) заданного входного потока принадлежит классу E (отдельно обрабатываемые программистом лексемы);
- EndBuff - нет ни одного символа в входном буфере, т.е. входной буфер пустой или был дотигнутым его конец, потому что все символы оказались разделителями и были пропущенными;
- IdentA - получено слово класса A (невходящее в таблицу зарезервированных слов класса A);
- IdentB - получено слово класса B (невходящее в таблицу зарезервированных слов класса B);
- Akeyword - получено зарезервированное слово класса A (слово класса A, входящее в таблицу зарезервированных слов класса A);
- Bkeyword - получено зарезервированное слово класса B (слово класса B, входящее в таблицу зарезервированных слов класса B);
- Ckeyword - получено зарезервированное слово класса C (слова класса C могут быть только зарезервированными, в противном случае они вообще будут нераспознаваемыми и в качестве результата будет возвращена лексема InvalidCh).

Пример: если указано, что символы ':' и '=' принадлежат классам C и C_1, но в таблице записана только пара ':=', то эти два символа распознаваемы только как пара.

Если нет разделителя между несколькими последовательными символами символьного класса C_1, результатом будет максимально длинное слово. Например, если '+' и '++' находятся в таблице зарезервированных слов класса C и входной поток содержит +++, первое обращение к TokenSearch выдает результат ++ (а не +++), а второе выдает +;

Пример: Пусть символы принадлежать классам как следует: **A..Z** классам A и A_1; цифры 0..9 классам A1 и E; '*', '+' и '-' классам B и B_1; пробел и табуляция классу D; ',' классу C, ':' и '=' классам C и C_1. Тогда можно распознавать лексемы:

- слова класса A: последовательность символов **A..Z** и **0..9**, начинающаяся с одного из символов **A..Z** и оканчивающаяся символом, предшествующим символу, отличающемуся от **A..Z**, **0..9**. Эти слова нам знакомы как идентификаторы языка Pascal. Например, COUNTER, WORD, LIST1, A1, B52 и т.д. Если для класса слов A дана и таблица зарезервированных слов, то те слова, которые попадают в таблице, будут определены как зарезервированные слова класса A, а не как обычные слова;
- слова класса B: последовательность символов: * + -. Например, * + - ** ++ -- *+ +- и т.д. Аналогическим образом (как класс A) можно задать таблицу зарезервированных слов класса B;
- слова класса C. На основании определения классов символов нельзя определить какие слова принадлежат классу C. Необходима еще таблица возможных слов класса C. Если в эту таблицу дефинированы ',' ':=' '==' '=', то их распознавание становится возможным. Например, если входной поток следующий: ',,,:====', будут выделены следующие лексемы: три раза 'запятая', последованные словами ':=' '==' '=', независимо от того, что между ними нет пробелов;
- начало специальной последовательности класса E, если встретится символ от **0** до **9**, непринадлежащий слову класса A.
- все пробелы и табуляции будут пропущены, потому что принадлежат классу D.

Пусть входной поток следующий:

LIST1:==++A-B C D,,E1 123 C

Тогда каждым вызовом TokenSearch будут распозаты последовательно следующие лексемы (в скобках даются их имена):

A-слово(LIST1) C-слово(:=) C-слово(=) B-слово(++) A-слово(A) B-слово(-) A-слово(B) A-слово(C) A-слово(D) C-слово(,) C-слово(,) A-слово(E1) E-символ(если обработать полностью, можно распознать число 123) A-слово(C)

Правила распознавания лексем, входной поток (буфер) и таблица зарезервированных слов задаются параметром типа структуры. Поля этой структуры имеют следующий смысл:

- IdName: указатель к переменной типа STRING максимальной длиной не меньше значения, записанного в поле IdSize. Значение IdName^ при активировании TokenSearch не указывает влияния. В это поле она записывает имя выделенного слова класса A или B;
- IdSize: максимальное число значащих символов имени класса A или B. Если слово длиннее, слово анализируется полностью, но в IdName^ записываются только первые IdSize символов. Не меняется функцией TokenSearch;
- flags: байт, каждый бит которого имеет специфическое предназначение. Функция TokenSearch меняет содержимое только бита 0. (Все остальные биты остаются без изменения.):
  - бит 0 - при активизировании TokenSearch не имеет значения. Если TokenSearch достигнула конца буфера, она устанавливает этот бит в 1, иначе - 0. Например, пусть происходит анализ входного текста BEGIN. После распознавания лексемы бит 0 будет иметь содержимое 1, если BEGIN находился в конце буфера, и 0 - в противном случае;

- бит 1 - содержимое 1 этого бита указывает на то, что распознавание зарезервированных слов класса A или B должно происходить независимо от того строчными или прописными буквами латинского алфавита кодированы они. В таком случае (бит 1 = 1) таблица зарезервированных слов должна содержать только прописные буквы латинского алфавита;
- бит 2 - использование этого бита то же самое как бит 1, но его действие относится к кириллице;
- бит 3 - содержимое 0 указывает на то, что имя выделенной лексемы класса A или B, будет записано (в IdName^) без изменения. Содержимое 1 вызывает замену всех латинских букв имени до его записи строчными или прописными (битом 5 выбирается какими именно);
- бит 4 - использование этого бита то же самое как бит 3, но его действие относится к кириллице;
- бит 5 - устанавливает строчными (1) или прописными (0) буквами будут записываться имена слов класса A или B (см. бит 3 и 4);
- бит 6 - не используется, но его значение должно быть 0;
- бит 7 - указывает на то для всех ли символов дефинирована принадлежность к классам или это относится только к первым 128 символов (7-битовой ASCII код). При содержимом 1 бита 7 таблица принадлежности символов занимает 128 байтов и используется 7-битовой ASCII код, т.е. все символы с кодом больше 128 не принадлежат никакому классу. Если бит 7 = 0, используется 8-битовой ASCII код и таблица принадлежности символов занимает 256 байт;

- buffer: указатель буфера входного потока. Если буфер типа string, то указатель должен указывать на его первый элемент. Не изменяется функцией TokenSearch;
- buffsz: размер входного буфера в байтах. Не изменяется функцией Token-Search;
- index: индекс начального элемента в буфере, с которого надо начать сканирование. В результате выполнения функции параметр index получает значение индекса того элелемента, которого достигла функция при сканировании;
- start: при активизации TokenSearch не имеет значения. Функция записывает в start значение индекса того элемента в буфере, с которого начинается выделенная лексема;
- KeyWordNo: Если обнаружено, что лексема является зарезервированным словом класса A, B или C, KeyWordNo получает в качестве результата номер зарезервированного слова в таблице зарезервированных слов, соответствующего класса (A, B или C);
- Classes: указатель массива, содержащего 128 или 256 однобайтовых элементов (в зависимости от типа ASCII кода: 7- или 8-битовой - бит 7 поле flags). Элемент с индексом N этого массива сопоставлен ASCII коду N. Биты соответствующего байта указывают на принадлежность (1) или непринадлежность (0) соответствующему классу: бит 0 классу D, 1 - C, 2 - E, 3 - A, 4 - B, 5 - A_1, 6 - B_1, 7 - C_1. Например, значение $40_{10} = 28_{16} = 00101000_2$ семидесятого элемента массива означает, что буква F (с десятичным ASCII кодом 70) принадлежит классам A и A_1;
- ClassAkw: указатель таблицы зарезервированных слов класса A. Таблица содержит в произвольном порядке все зарезервированные слова класса A,

каждому из которых предшествует один байт, определяющий число символов этого слова. Конец таблицы определяется байтом с нулевым содержимым. Например, если таблица должна содержать зарезервированные слова BEGIN, END, IF, THEN и ELSE, и декларирована как упакованный массив символов, то ей можно присвоить значение #5'BEGIN' #3'END' #2'IF' #4'THEN' #5'ELSE' #0;

- ClassBkw: указатель таблицы зарезервированных слов класса B. Ее формат тот же самый как формат таблицы ClassAkwˆ;
- ClassCkw: указатель таблицы зарезервированных слов класса B. Ее формат тот же самый как формат таблицы ClassAkwˆ;

Для удобства экспортируются константы, которыми легче установит состояние флагов и принадлежность классам. Константы следующие:

- Class_D = $01; { класс D, разделители }
- Class_C = $02; {класс C (только из зарезервированных слов)}
- Class_E = $04; { класс E, обрабатываемый програмистом }
- Class_A = $08; { класс A }
- Class_B = $10; { класс B }
- Class_A1 = $20; { класс A_1 }
- Class_B1 = $40; { класс B_1 }
- Class_C1 = $80; { класс C_1 }

- flag_EOB = $01; { достигнут ли конец буфера }
- flag_CKC = $02; { Case insensitive for Keywords (Cyr) }
- flag_CKL = $04; { Case insensitive for Keywords (Lat) }
- flag_CCC = $08; { Correct Case for identifiers (Cyr) }
- flag_CCL = $10; { Correct Case for identifiers (Lat) }
- flag_LC = $20; { Do case correction by LowCase }
- flag_ASCII= $80; { use ASCII code only (#0..#127) }

Их использование проиллюстрируем следующим примером. Пусть хотим дефинировать все буквы латинского алфавита как принадлежащие классам A и A_1, буквы - кириллицы классам B и B_1 и цифры - классам E и A_1. Тогда можем записать во все байты-дефиниции принадлежности всех букв латинского алфавита выражение Class_A + Class_A1, всех букв кириллицы - Class_B + Class_B1 и всех цифр - Class_E + Class_A1.

Кроме основной функции лексического анализа модуль UniLEX экспортирует еще несколько процедур:

function GetInteger(const S; i:word; l:word; var E:word): LongInt;

Действие: Задачей GetInteger является преобразование символьного представления целого числа (подчиненного синтаксису языка UniPascal) в соответствующее ему внутреннее. Другими словами, считывает целое число, но не с текстового файла, а с заданного буфера. Ее параметры следующие: **S** - входной буфер, **i** - начальный индекс в нем, **l** - максимальное число байтов, которые подлежать анализе, с **i**-го элемента буфера. Параметром-переменной **E** возвращается число анализированных цифр. Функцией IOresult возвращается значение 0 при правильно записанном числе (нет ошибок) и значение 144 при обнаружении ошибки. Не считается ошибкой ситуация, при которой в буфере записано правильное представление числа, последованного каким либо другим

символом. Например, для входного потока 1234xyz в качестве результата возвращается 1234 и параметр **E** = 4. Ошибка индицируется только если входной поток не содержит в своем начале символьное предствление числа (например, ABC1234 или --1234). Начальные пробелы пропускаются. Если число - вещественное, хотя и процедура предназначена обрабатывать только целые числа, она может и не сигнализировать об ошибке в некоторых случаях. Например, 123.45 будет анализированным как 123 и параметр **E** = 3. Но .123, считается ошибкой.

function GetReal(const Src; i: word; l: word; var e: word): Real;

Действие: Действие этой функции аналогично действию функции GetInteger. Разница в том, что она делает преобразование символьного представления вещественных чисел (подчиненного синтаксису языка UniPascal) в соответствующее ему внутреннее представление.

procedure Real2Str(R: real; W: Natural; var S: string[79]);

Действие: Действие этой процедуры обратно действию функции GetReal. Она делает преобразование внутреннего представления вещественного числа в соответствеющее ему символьное представление. Символьное представление должно иметь заданное параметром **W** число цифр за десятичной точкой. Эта процедура работает как стандартная процедура WRITE(R:0:W), но ее результат не записывается в текстовой файл, а в строку, заданную параметром **S**.

procedure Exp2Str(R: real; W: Natural; var S: string[15]);

Действие: Действие этой процедуры аналогично действию процедуры Real2Str, только символьное представление должно быть в экспоненциальной форме. Параметром **W** определяется число символов (для цифр, точки, порядка и знака), которое должно содержать символьное представление. Минимальное значение **W** - 8, а максимальное - 15 (как при WRITE(R:W)).

procedure Int2Str(Value: LongInt; var s: string[11]);

Действие: Действие этой процедуры аналогично действию процедуры Real2Str, только она делает преобразование внутреннего представления целого числа в его соответствующее символьное. Символьное представление занимает столько позиций, сколько необходимо, но не больше 11 (10 цифр и знак).

procedure Long2Str(Value: LongWord; var s: string[8]);

Действие: Действие этой процедуры аналогично действию процедуры Int2Str. Но она делает преобразование внутреннего представления целого числа из двойного слова в соответствующее ему символьное представление в шестнадцатеричной системе счисления. Символьное представление занимает 8 символов, по одному для каждой половины байта двойного слова.

procedure Word2Str(Value: word; var s: string[4]);

Действие: Действие этой процедуры аналогично действию процедуры Long2Str. Но она делает преобразование целого числа из слова, а не из двойного слова. Символьное представление занимает 4 символа.

procedure Byte2Str(Value: byte; var s: string[2]);

Действие: Действие этой процедуры аналогично действию процедуры Long2Str. Но она делает преобразование целого числа из байта, а не из двойного слова. Символьное представление занимает 2 символа.

function UpCaseCyr(ch: char): char;

Действие: Эта функция работает как стандартная функция UpCase, но кроме букв латинского алфавита делает преобразование и строчных букв кириллицы в прописные.

function LoCase(ch: char): char;

Действие: Эта функция работает как стандартная функция UpCase, но делает обратное преобразование (прописных букв в строчные).

function LoCaseCyr(ch: char): char;

Действие: Эта функция работает как функция UpCaseCur, но делает обратное преобразование (прописных букв в строчные).

procedure UpCaseS(var s: string);

Действие: Эта процедура делает преобразование всех строчных латинских букв, находящихся в строке **S**, в прописные.

procedure UpCaseSCyr(var s: string);

Действие: Эта процедура работает как процедура UpCaseS, но кроме букв латинского алфавита делает преобразование строчных букв кириллицы в прописные.

procedure LoCaseS(var s: string);

Действие: Эта процедура работает как процедура UpCaseS, но делает обратное преобразование (прописных букв в строчные).

procedure LoCaseSCyr(var s: string);

Действие: Эта процедура работает как процедура UpCaseSCyr, но делает обратное преобразование (прописных букв в строчные).

function ScanFor(ch: char; const s; i: natural; l: integer): integer;

Действие: Эта функция ищет символа в буфере (символьная строка, упакованный массив байтов или символов). Параметром **CH** задается искомый символ, параметром **S** - буфер, в котором будет совершаться поиск, параметром **I** - начальный элемент буфера, с которого начнется поиск, **L** - максимальное число элементов, подлежащих проверке. Если **L** больше нуля, поиск ведется с текущего элемента к концу буфера. А если **L** меньше нуля - в обратном направлении. Результатом функции является число, указывающее сколько элементов было пропущено до того, как был обнаружен искомый символ, т.е. результат 0 означает, что **I**-ый элемент буфера содержит искомый символ, 1 - **(I+1)**, и т.д. Отрицательное значение результата является указанием на то, что поиск был совершен в обратном направлении (т.е. **L** $<$ 0 или, другими словами, знак результата совпадает с знаком параметра **L**). Если функция не обнаружила искомого символа (**CH**), в качестве результата она возвращает значение параметра **L**. Таким образом, результат функции надо интерпретировать так: если он отличается от **L**, искомый символ был обнаружен в позиции с индексом, значение которого является суммой результата и **I**.

function ScanNotFor(ch: char; const s; i: natural; l: integer): integer;

Действие: Эта функция ищет символа, отличающегося от заданного параметром. Она работает аналогично функции ScanFor, только ищет символа, отличающегося от заданного символа (а не совпадающего с ним).

function Compare(l: natural; const s; si: natural; const d; di: natural): natural;

Действие: Эта функция сравнивает содержимое двух буферов (символьные строки, упакованные массивы байтов или символов). Параметром **L** задается число байтов, содержимое которых подлежит сравнению, параметрами **S** и **D** задаются первый и второй буфера. Параметрами **SI** и **DI** задаются начальные индексы элементов, соответственно, первого и второго буфера, с которых должно начинаться сравнение. Результатом является число байтов, содержимое которых совпадает. В частности, он совпадает с значением **L** при одинаковых буферах и имеет значение 0 при отличающихся еще своими первыми элементами буферах.