Приложение С. Описание на стандартните модули

Съдържащите се в стандартната библиотека (намира се на дистрибутивната дискета под името SYSTEM.UPL) модули са следните:

- UniCRT - модул за работа с екрана в текстов режим;
- UniDOS - модул за връзка с операционната система;
- UniGRAPH - модул за използване на графичния екран.
- UniLEX - модул за лексическа обработка на текст;

Техните описателни (interface) части се намират съответно във файловете: **UNICRT.DOC**, **UNIDOS.DOC**, **UNIGRAPH.DOC** и **UNILEX**.DOC. Самите файлове са пакетирани и се намират във файла **EXAMPLES.ARC**. За да ги разпакетирате, напишете на командната линия на UniDOS следното (най-добре е да използвате две дискети):

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

където **XXX** е името на модула, чието описание ще разпакетирате.

Тези файлове се дават само за удобство. Самите описателни части са компилирани и записани в библиотеката.

С.1. Описание на модула UniCRT

Модулът UniCRT предоставя средства за удобно използване на възможностите на BIOS за работа с екрана на микрокомпютрите от фамилията Пълдин 601/601A/601M (601M е еквивалентен на 601A).

Експортират се следните константи, представящи кодовете на съответните цветове в текстов режим:

• Black =	0	Черно
• Blue =	1	Синьо
• Green =	2	Зелено
• Cyan =	3	Синьо-зелено
• Red =	4	Червено
• Magenta =	5	Виолетово
• Brown =	6	Кафяво
• LightGray =	7	Светло сиво
• DarkGray =	8	Тъмно сиво
• LightBlue =	9	Светло синьо
• LightGreen =	10	Светло зелено
• LightCyan =	11	Светло синьо-зелено
• LightRed =	12	Светло червено
• LightMagenta =	13	Светло виолетово
• Yellow =	14	Светло жълто
• White =	15	Бяло
• Blink =	128	Код за добавяне към основния цвят за получаване на мигащо изображение.

procedure ActvCrsr(size: ShortCard);

Действие: Превключва активния курсор (този, който се появява когато се чете от клавиатурата) в един от възможните размери: 0 - няма видим курсор; 1, 2, 3, 4 - курсора е с размер съответно $^{1}/_{4}$, $^{2}/_{4}$, $^{3}/_{4}$ и $^{4}/_{4}$ от пълната си големина. По премълчаване този курсор има размер $^{1}/_{4}$ от пълната си големина (1).

procedure ClrEol;

Действие: Изтрива (т.е. записва вместо тях интервал) всички символи от текущата позиция на курсора до края на реда, без да придвижва самия курсор.

procedure ClrEos;

Действие: Изтрива всички символи от текущата позиция на курсора до края на екран, без да придвижва самия курсор.

procedure ClrScr;

Действие: Изтрива екрана и поставя курсора в горния ляв ъгъл.

procedure Delay(ms: cardinal);

Действие: Изчаква указаните чрез параметъра брой милисекунди.

function GetColor: ShortCard;

Действие: Резултат от тази функция е текущият атрибут, с които се изписват символите на екрана. Той има същия смисъл, както и за процедурата SetColor.

procedure GotoXY(X, Y: ShortCard);

Действие: Придвижване на курсора на екрана на позиция с координати (X, Y). Горния ляв ъгъл на екрана е с координати (1, 1), като X координатата нараства надясно, а Y координатата надолу.

function KeyPressed: boolean;

Действие: Връща TRUE, ако от момента на последното четене от клавиатурата до момента на нейното активиране е бил натиснат клавиш. С други думи проверява дали буферът на клавиатурата не е празен.

procedure PassCrsr(size: ShortCard);

Действие: Превключва пасивния курсор (този, който се появява, когато не се чете от клавиатурата, т.е. докато програмата работи) в един от възможните размери: 0 - няма видим курсор. 1, 2, 3, 4 - курсора е с размер съответно $^1/_4$, $^2/_4$, $^3/_4$ и $^4/_4$ от пълната си големина. По премълчаване този курсор е невидим (0).

procedure PlayMelody(var Melody);

Действие: Изсвирва цяла мелодия. Мелодията е съставена от последователност от ноти (виж описанието на PlayNote). Всеки елемент на последователността е или нота (заемаща 3 байта) или команда за смяна на тембъра или край (един байт). Ако елементът е нота, то първият байт е номерът на нотата, вторият - продължителността на нотата, а третият - продължителността на паузата след нотата. Смяната на тембъра се задава със един байт, имащ стойност от $F8 до $FE, отговарящ съответно на тембър от 1 до 7. Край на мелодията е се задава с байт със стойност $FF.

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

Действие: Изсвирва нота. Note е номер на нота от 0 до 72. На номер 0 - съответства пауза, на номера от 1 до 72 съответстват нотите, като на 29 съответства нота ЛА от първа октава. Tembr е от 1 до 7 и се интерпретира като тембър. Length е продължителност на звучене на нотата в единици от 1.536 милисекунди.

function ReadKey: char;

Действие: Изчаква за натискане на клавиш и връща неговия код. Кодовете на клавишите са описани в приложението "Сведения за клавиатурата и екрана на Пълдин".

function ScrXsize: byte;

Действие: Връща хоризонталния размер на екрана, т.е. колко символа има на ред.

function ScrYsize: byte;

Действие: Връща вертикалния размер на екрана, т.е. колко са редовете.

procedure SetColor(Attribute: ShortCard);

Действие: Задава нов цвят за символите, които ще бъдат записвани впоследствие на екрана. Тази процедура не извършва нищо, ако се използва на Пълдин 601. Тя работи само на Пълдин 601A/601M и то само ако е избрано цветно изображение с 40 символа на ред. Цветовете, които могат да се използват, са описани чрез константи. Основното правило е, че цветът на буквите може да бъде само от първите 8 цвята, а цветът на фона - от всички 15 цвята. Задаването на цветен атрибут се определя по формулата 16 * цвят на буквата + цвят на фона. Ако е необходимо мигащо изображение, добавете към така получения атрибут константата Blink.

procedure TextMode(Mode: ShortCard);

Действие: Превключва екрана в един от следните режими: черно-бяло изображение с 40 символа на ред; черно-бяло изображение с 80 символа на ред; цветно изображение с 40 символа на ред. Като фактически параметър използвайте една от следните експортирани константи:

- BW40 = 0; { черно-бяло изображение 40x25 }
- BW80 = 4; { черно-бяло изображение 80x25 }
- CO40 = 5; { цветно изображение 40x25 }

function WhereX: ShortCard;

Действие: Връща текущата X позиция на курсора, т.е. номерът на колоната, на която се намира той.

function WhereY: ShortCard;

Действие: Връща текущата Y позиция на курсора, т.е. номерът на реда, на който се намира той.

C.2. Описание на модула UniDOS

Модулът UniDOS предоставя средства за удобно използване на възможностите на операционната система за работа с файловете.

Експортират се следните константи, представящи маските за обработка на атрибутите на файл:

- ReadOnly = $01; Файлът е само за четене;
- Hidden = $02; Файлът е скрит;
- SysFile = $04; Файлът е системен;
- VolumeId = $08; Обозначение за идентификатор на дискета;
- Directory= $10; Поддиректория;
- Archive = $20; Бил ли е файла архивиран;
- AnyFile = $3f; Маска за откриване на какъв да е файл.

Експортират се и следните типове, необходими за някои от процедурите и функциите:

```
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 };
```

След всяко изпълнение на процедура или функция трябва да се проверява с помощта на функцията DOSerr за настъпилите грешки (кодовете на грешките са дадени в приложението за входно изходни грешки) .

procedure ChDir(const s: string);

Действие: Задава нова текуща поддиректория.

function DiskFree(drv: byte): longint;

Действие: Връща броят на свободните байтове на носителя, поставен в устройство с номер, даден като параметър. Номерата на устройствата са 0 - текущото устройство, 1 - устройство A:, 1 - B: и т.н.

function DosErr: word;

Действие: Връща кода на грешката на последната използвана процедура или функция от този модул.

function DosVers: word;

Действие: Връща версията на операционната система. Старшият байт съдържа главната (major) версия, а младшия байт - подверсията (minor).

procedure Erase(const s: string);

Действие: Изтрива файл, чието име е дадено като параметър. Тази процедура връща код на грешка чрез IOresult, а не чрез DOSerr.

Ограничение:Файлът не трябва да бъде отворен в този момент.

procedure ErrText(var s: ComStr);

Действие: Дава текстът на грешката на последната използвана процедура или функция от този модул в параметъра **s**. Кодът на грешката трябва да бъде по-малък от 128 (т.е. грешка от операционната система). Ако е по-голям от 128, то това е код на грешка на файловата система на UniPascal. Нито една от процедурите в модула UniDOS не предизвиква грешка на файловата система на UniPascal.

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

Действие: Търси първия файл, чието име отговаря на името, зададено в Path. Attr е маската, с която трябва да се търсят файловете. Например нека искаме да намерим файл с име MYFILE. Независимо дали за операционната система той е само за четене (read only) или не е. Освен това не се интересуваме от суфикса на името. Трябва да зададем като параметър Path = 'MYFILE.*' и ATTR = ReadOnly. А ако искаме да намерим всички файлове, чиито имена започват с A, ще трябва да зададем Path = 'A*.*' и ATTR = AnyFile. В параметър-

променливата F ще бъдат върнати: името на намерения файл, неговите атрибути и т.н. Променливата F трябва да се запази (да не се променя), за да може да се използва за последващо търсене чрез процедурата FindNext. Ако търсения файл е намерен, то DOSerr ще върне 0, а ако не е намерен, ще върне номер на грешката поради която не е намерен.

procedure FindNext(var F: SearchRec);

Действие: Тази процедура се използва за търсене на следващо срещане на файл, след като вече е използвана процедурата FindFirst. Действието ѝ е аналогично на FindFirst, но не е необходимо да се задават името и атрибутите, тъй като вече са зададени при активирането на FindFirst. Като параметър F трябва да бъде зададена същата променлива (от тип SearchRec), която е била зададена при FindFirst. Например нека да отпечатаме съдържанието на директорията, т.е. искаме да реализираме командата DIR на операционната система. Тогава фрагмент от програмата, която намира имената на всички файлове е:

```
...
var F: searchrec;
begin
  ...
  FindFirst('*.*', AnyFile, F);
  while DosErr = 0 do begin       { обработваме, докато се намира }
    WriteDirEntry(f);              { отпечатване на името на файла }
    FindNext(F);                     { намиране на следващия файл }
  end { while };
  ...
end.
```

Тук не даваме текста на процедурата WriteDirEntry, а на мястото на точките се намират други части от програмата.

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

Действие: Записва в променливата **s** пътя на текущата поддиректория за устройство drv.

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

Действие: Връща в Attr атрибута на файла с име, дадено в **s**. За по удобно обработване на атрибута може да се използват експортираните константи.

Ограничение: Файлът не трябва да бъде отворен в този момент.

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

Действие: Връща в Time времето (датата и часа), когато файлът е бил модифициран за последен път. За обработка може да се използва процедурата UnpackDT.

Ограничение: Файлът не трябва да бъде отворен в този момент.

procedure MkDir(const s: string);

Действие: Създава поддиректория с име и път, зададени в **s**.

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

Действие: Пакетира времето, подготвяйки го за SetFtime.

procedure Rename(const old, new: string);

Действие: Преименуване на файл със старо име old и зададено ново име new. Тази процедура връща грешките си чрез IOresult, а не чрез DOSerr. Файлът, който ще се преименува, **не** трябва да бъде отворен в момента.

procedure RmDir(const s: string);

Действие: Унищожава зададената поддиректория.

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

Действие: Задава нов (сменя) атрибут за файла **s**.

Ограничение: Файлът не трябва да бъде отворен в този момент.

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

Действие: Задава ново (сменя) време за файла **s**.

Ограничение: Файлът не трябва да бъде отворен в този момент.

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

Действие: Разпакетира полученото чрез GetFtime време.

С.3. Описание на модула UniGRAPH

Модулът UniGRAPH предоставя процедури и функции за работа в графичен режим на микрокомпютрите от фамилията Пълдин.

Графичните екрани на микрокомпютъра Пълдин 601/601A/601M се делят според тяхната разрешаваща способност на следните 3 вида (в скоби е дадена разрешаващата способност при Пълдин 601A/601M):

- висока разрешаваща способност: 320x200(640x200), 2 цвята;
- средна разрешаваща способност: 160x200(320x200), 4 цвята;
- ниска разрешаваща способност: 80x200(160x200), 16 цвята.

Разрешаващата способност се характеризира с броя на точките, които може да има на един ред на екрана, броя на редовете, които може да има на екрана и броя на цветовете с които може да бъде нарисувана една точка. Дадените по-горе стойности за различните видове графични изображения се интерпретират така: в режим средна разрешаваща способност има 160 (320) точки на ред, 200 реда от точки и 4 възможни цвята за една точка.

Кодираната информация, която определя изображението, показвано на графичния екран на микрокомпютъра (независимо от разрешаващата си способност) се разполага в оперативната памет и заема 8 или 16 килобайта съответно за Пълдин 601 и Пълдин 601A/601M.

В режим на висока разрешаваща способност може да се задават само два цвята - черен и бял, при ниска - 16 цвята, а в режим на средна разрешаваща способност само 4 цвята, но избрани от една палитра (избор на палитрата се прави отделно от избора на цвят). Номерата на цветовете съвпадат с тези номера дадени в модула UniCRT. Палитрите са както следва:

1) палитра	черен(0),	зелен(2),	червен(4),	жълт(6);
2) палитра	син(1),	синьо-зелен(3),	виолетов(5),	бял(7);
3) палитра	сив(8),	ярко зелен(10),	ярко червен(12),	ярко жълт(14);
4) палитра ярки:	син(9),	синьо-зелен(11),	виолетов(13),	бял(15).

Независимо от разрешаващата способност координатната система е една и съща (координатите са логически). Графичната система на компютъра (намираща се в XBIOS) прави превръщането от логически във физически координати и извършва чертаенето по екрана. Горният ляв ъгъл на екрана има координати (0, 0), а долния десен ъгъл - (639, 399), т.е. в логически координати екрана е с размири (в точки) 640x400.

Изписването на символи е възможно и в графичен режим с висока или средна разрешаваща способност. Символите се изписват автоматично (при използване на

WRITE) в графичния екран, ако той е активен. От програмиста в този случай не се изискват никакви действия, т.е. механизмът е прозрачен. Освен READ/WRITE в графичен режим могат да се използват и всички процедури от модула UniCRT. Графичният курсор (който е невидим, т.е. не се индицира, както текстовия) е различен от текстовия курсор, т.е. текущата графична позиция и текущата текстова позиция са независими. Например: gotoxy(10, 10); и moveto(40, 40); ще предизвикат следното: изписващият се след това текст ще бъде от позиция (текстова) (10, 10), а чертаенето например линия ще започва от позиция (графична) (40, 40).

function ActualXsz: cardinal;

Действие: Връща физическия хоризонтален размер на екрана (брой точки на ред) и 0 ако не е в графичен режим.

function ActualYsz: cardinal;

Действие: Връща физическия вертикален размер на екрана (брой редове от точки на екран) и 0, ако не е в графичен режим.

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

Действие: Отделя място за дефиниция на Npg * 32 символа. Възможна е дефиницията на 32 последователни символа, като кодът на първия от тях трябва да бъде един от следните: $00, $20, $40, $60, $80, $a0, $c0, $e0. Паметта трябва де бъде отделена на граница на 256 байта (както се изисква от XBIOS), т.е. младшият байт на началния адрес ще съдържа 0. Дефиницията на всеки символ се състои от 8 байта (по един за всеки ред от графичното изображение на символа). Следователно дефиницията на 32 символа се състои от 256 байта, по 8 байта за всеки символ, разположени един след друг. Тази функция само отделя място за тези символи и връща началният му адрес. Зареждането на дефиницията трябва да се извърши след това (например прочитане от файл). Тъй като отделянето на памет ще се извършва с изравняване на 256 байтова граница в повечето случаи се отделя повече памет от необходимата (но излишъкът не надминава 255 байта). Ако се зададе параметъра FREE = true, то неизползваната памет (излишната за дефиниция на букви) ще се освободи от функцията AllctCharSet (по този начин ще се образува 'дупка' в динамичната памет). Ако се зададе FREE = false, то неизползваната памет няма да бъде освободена.

function AllctScreen(free: boolean): pointer;

Действие: Отделя необходимото количество памет за графичен екран (8/16 килобайта). Ако вече съществува графичен екран, не извършва нищо. При резервирането на памет за графичен екран, възможно е да бъде заета малко повече памет (със не повече от 15 байта), тъй като се прави изравняване на 16 байтова граница. Ако се зададе параметър free = true, то тези байтове се освобождават веднага за използване (аналогично на AllctCharSet). Ако параметърът free = false, то те се освобождават едва, когато се освободи паметта за графичния екран (т.е. заедно с неговата памет).

function GraphSize: cardinal;

Действие: Връща размера на паметта която е необходима за графичен екран в зависимост от компютъра. 8 килобайта при Пълдин 601 и 16 при Пълдин 601A/601M.

procedure DefaultCharSet;

Действие: Възстановява стандартните дефиниции на символите (такива, каквито са били преди изпълнението на програмата).

procedure DrawBar(X, Y: cardinal);

Действие: Рисува запълнен правоъгълник, единият връх на който се определя от текущото положение на курсора, а другият - от зададените координати, които могат да бъдат абсолютни или относителни в зависимост от зададения вид от процедурата SetCoordType.

procedure DrawCircle(rX, rY: cardinal);

Действие: Рисува елипса с център в текущото положение на курсора и радиуси: rX по оста X и rY по оста Y.

procedure DrawDisk(rX, rY: cardinal);

Действие: Рисува запълнена елипса с център в текущото положение на курсора и радиуси: rX по оста X и rY по оста Y.

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

Действие: Изпълнява зададена последователност от действия: придвижване на графическия курсор без рисуване и придвижване с рисуване. Рисунката се получава чрез последователно начертаване на отсечки. За целта трябва да са дадени указания за новото положение на курсора чрез полярни координати: направление на движението и дължина на отсечката до новата точка. Дължината се задава с числа от 1 до 16 (в разстояния между две съседни логически точки). Посоката се задава чрез ъгъл спрямо хоризонталната ос по посока на движение на часовата стрелка, чиято големина се определя с числа от 0 до 7 (в единици от по 45°). С параметъра Shape се определя една последователност от N на брой движения на курсора или придвижвания с рисуване, необходими за получаване на исканото изображение. Тези данни са разположени в N последователни байта предхождани от дума, съдържаща това число N. Числото се предполага записано със старша част, предхождаща младшата (обратно на представянето на числата в UniPascal). Всеки от следващите N байта задава едно елементарно движение (със или без чертаене на отсечка) на графичния курсор. Информацията за това действие се записва в байта, както следва: Съдържание 1 в бит 7 е указание, че при движени курсора трябва да рисува, а нула - курсора да се придвижва без рисуване. Битовете от 6 до 4 определят ъгъла, а тези от 3 до 0 - големината на отсечката (0 задава единица разстояние, 1 задава две единици, и т.н. 15 задава 16 единици).

С параметъра ROT се задава завъртане на определеното със Shape и още ненарисувано изображение на ъгъл определен с число от 0 до 7 (определен по същия начин, както при елементарните движения).

С параметъра SCALE се задава увеличаване (мащабиране) на определеното със Shape и още ненарисувано изображение чрез число от 0 до 15 (0 - означава, че изображението няма да бъде увеличавано, 1 - удвояване на дължината на всяка отсечка, и т.н. 15 - 16 пъти увеличаване на дължината на всяка отсечка).

procedure FreeScreen;

Действие: Освобождава заетата за графичен екран памет, ако заявката за създаване на графичен екран е била дадена от същата програма. В противен случай не прави нищо.

function GetVmode: word;

Действие: Връща една дума, която съдържа: текущия видеорежим в старшия байт на думата и атрибута/палитрата в младшия байт на думата.

procedure LineTo(newX, newY: word);

Действие: Рисува линия от текущото положение на курсора до дадените координати, които могат да бъдат абсолютни или относителни в зависимост от зададения вид от процедурата SetCoordType.

procedure MoveTo(newX, newY: word);

Действие: Премества графичния курсор на нова позиция. Координатите са абсолютни или относителни в зависимост от зададения вид от процедурата SetCoordType.

function Ncolors: shortcard;

Действие: Връща броя на цветовете, които е позволено да се използват, 2 за висока разделителна способност, 4 за средна, 16 за ниска, 16 за 40x25 цветен текстов екран, 2 за 40x25 или 80x25 черно-бял текстов екран.

procedure Point(X, Y: word);

Действие: Рисува точка с координати X, Y. Координатите са абсолютни или относителни в зависимост от зададения вид от процедурата SetCoordType.

function RelCoord: boolean;

Действие: Връща вида на използваните координати: TRUE при относителни и FALSE при абсолютни.

procedure SetCoordType(relative: boolean);

Действие: Задава типа на координатите, които ще се използват в различните процедури - абсолютни или относителни. Относителните координати са отместване от текущото положение на курсора. Параметър TRUE задава относителни координати.

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

Действие: Установява за ползване в графичен режим дефинираните от потребителя символи, PAGES е множество, в което трябва да бъдат зададени номерата на групите по 32 символа, чиято дефиниция ще се използва. Например: Следващия отрязък от програма ни позволява да дефинираме 3 групи от по 32 символа (група 0 - $00..$1F, група 6 - $C0..$Df и група 7 - $E0..$FF) и да ги използваме:

```
 ...
 var ChSetBuff: pointer;
begin
   ...
   ChSetBuff:= AllctCharSet(3, true {/false});
   DefineCharSet; { процедура, написана от програмиста за
   дефиниране на символите, например зареждане от дискета }
   SetCharSet([0, 6, 7], ChSetBuff);
   { от този момент за изобразяване на символите в графичен режим
   при използване на WRITE ще бъдат използвани символите,
   дефинирани от потребителя }
   ...
```

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

Действие: Установява цвят за рисуване (COLOR) и начин на рисуване (MODE). С параметъра COLOR се установява цвят (число, допустимо според установения видеорежим) за ползване от други процедури. С параметъра MODE се установява начинът, по който трябва да се рисува всяка точка в зависимост от заявеното и съществуващото състояние на тази точка върху екрана. Стойностите, които mode може да приема, са от 0 до 13. Те представляват код,

които определя функцията (N), която трябва да се приложи над предишното състояние (S) и над заявеното (P). Стойност 1 и 0 на S и P съответно означават светене и несветене на точката. Всички възможни номера на функции за рисуване се експортират като мнемомонични константи:

```
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: Включва указания видеорежим. Номерът на видеорежима (ModeNo) се експортира, като константа и може да заема стойности със следния смисъл:

- text_bw40 = 0; { черно-бял/цветен текстов екран 40x25 }
- graph_lo = 1; { графика 80(160)x200, 16 цвята }
- graph_mid = 2; { графика 160(320)x200, 4 цвята }
- graph_hi = 3; { графика 320(640)x200, 2 цвята }
- text_bw80 = 4; { черно-бял текстов екран 80x25 (601A) }

Параметърът Pallette е номер на палитрата за графика със средна разрешаваща способност, номер на цвят за графика с ниска разрешаваща способност или атрибут за текст 40x25. В останалите случаи се игнорира. Ако се указва включване на графичен режим, но не е отделена памет за него, процедурата не извършва нищо.

function WhereGRx: cardinal;

Действие: Връща текущата X координата на графичния курсор.

function WhereGRy: cardinal;

Действие: Връща текущата Y координата на графичния курсор.

С.4. Описание на модула UniLEX

Модулът UniLEX предоставя удобни, вградени в интерпретатора на Y код, процедури и функции за лексическа обработка на текст. Това е обработка, свързана с отделяне на думи, разпознаване на числа и с други възможности. Този модул се използва и от компилатора на UniPascal. Програмистът сам може да задава кои символи влизат в рамките на една дума, кои са разделители и т.н.

Основната и най-важна задача на лексическия анализ е разбиването на входния поток на отделни лексеми. Например при синтаксиса на Pascal следния входен поток

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

се преобразува във потока от лексеми:

идентификатор(x), присвояване, число(1), плюс, число(2), звезда,
идентификатор(y), минус, идентификатор(z), накл_черта, число(2)

Анализът на лексеми е по-прост в сравнение с директен анализ на входния поток (текста). Задачата за разбиване на входния поток на лексеми обикновено се решава

чрез подпрограма. Такава подпрограма работи като краен автомат, на чийто вход се подава входния поток, а изхода е поредната разпозната лексема. В интерпретатора на Y код е вграден такъв краен автомат, написан на асемблер и затова работещ значително по-бързо, отколкото ако беше написан на UniPascal. Този краен автомат е оформен като функция, наречена TokenSearch (търсене на лексема). Входните параметри задават входния поток, който тя трябва да сканира, и правилата, по които се разпознават лексемите.

function TokenSearch(var ls: LexStructure): token;

Описание: За TokenSearch входният поток се задава чрез буфер, който трябва да бъде зареден с част от входния поток (например прочетен е един ред във входния буфер). Символите във входния текст се делят на 8 основни класа (A, B, C, A_1, B_1, C_1, D, E), а думите - на три (A, B, C). Под дума в случая се разбира последователност от символи без разделители.

- клас A, B, C - от тези класове са символите, с които могат да започват думите от съответните класове - A, B, C;
- клас A_1, B_1, C_1 - от тези класове са символите, които могат да се съдържат (след първия символ) в думите от съответните класове - A, B, C;
- клас D - от този клас са всички разделители, които се пропускат (обикновено това са интервал и табулация);
- клас E - от този клас са специални символи - начало на специална дума, обработвана от програмиста отделно (обикновено това са числата - реални и цели).

При активиране на TokenSearch първо се пропускат (прескачат) всички разделители (символи от входния поток които са от клас D), ако има такива. След това се анализира (разпознава) лексемата следваща разделителите. Резултатът от работата на функцията е описан от изброимия тип TOKEN (лексема) и неговите константи означават:

- InvalidCh - първият символ (след пропуснатите разделители) в зададения входен поток не е от нито един от възможните 8 класа или не е разпозната никаква дума (символите са от клас C и C_1, но няма ключова дума от този клас в таблицата);
- UserCh - първият символ (след пропуснатите разделители) в зададения входен поток е от клас E (отделно обработвани от програмиста лексеми);
- EndBuff - няма нито един символ във входния буфер, т.е. входния буфер е празен или е достигнат неговия край, защото всички символи са били разделители и са пропуснати.
- IdentA - разпозната е дума от клас A (липсваща в таблицата от ключови думи за клас A);
- IdentB - разпозната е дума от клас B (липсваща в таблицата от ключови думи за клас B);
- Akeyword - разпозната е ключова дума от клас A (дума от клас A, намираща се в таблицата от ключови думи за клас A);
- Bkeyword - разпозната е ключова дума от клас B (дума от клас B, намираща се в таблицата от ключови думи за клас B);
- Ckeyword - разпозната е ключова дума от клас C (думите от клас C могат да бъдат само ключови думи, в противен случай те не се разпознават изобщо и се връща лексема InvalidCh). Например, ако е указано, че ':' и '=' принадлежат на клас C и C_1, но в таблицата фигурира само ':=', то двата символа могат да се разпознават само в тази комбинация и в никоя

друга. Ако няма разделители между няколко последователни символа от символния клас C_1, разпознава се максимално дългата дума. Например, ако '+' и '++' са в таблицата за ключови думи от клас C и във входния поток се срещне +++, то първото обръщение към TokenSearch разпознава ++, а второто разпознава + (не една лексема +++, нито три последователни лексеми +);

Пример: Нека сме задали символите така: **A..Z** от клас A и A_1, цифрите **0..9** от клас A_1 и клас E, '*', '+' и '-' от клас B и B_1, интервал и табулация от клас D, ',' от клас C, ':' и '=' от клас C и C_1. При така дефинираните символи могат да се разпознават следните лексеми:

- думи от клас A: последователност от символите **A..Z** и **0..9**, започваща със символ от **A..Z** и завършващи със символ различен от **A..Z**, **0..9**. Тези думи са ни добре познати, като идентификатори в езика Pascal. Например COUNTER, WORD, LIST1, A1, B52 и т.н. Ако за думите от клас A е дадена и таблица с ключови думи, то тези от тях, които са дефинирани в таблицата ще бъдат определени като ключови думи от клас A, а не като обикновени думи;
- думи от клас B: последователност от символите: * + -. Например: * + - ** ++ -- *+ +- и т.н. Аналогично на думите от клас A, може да бъде зададена таблица с ключови думи от клас B;
- думи от клас C не могат да бъдат разпознати само въз основа на определение на символите от съответен клас. За разпознаване е необходима и таблица с възможните думи от клас C. Ако в тази таблица са дефинирани ',' ':=' '==' '=', то става възможно да бъдат разпознати. Например, ако входният поток е следния: ',,,:====', ще бъдат разпознати три лексеми 'запетая', последвани от лексемите ':=' '==' '=' независимо от това, че между тях няма интервал;
- начало на специална последователност от клас E, ако бъде срещнат символ от **0** до **9**, непринадлежащ на дума от клас A;
- всички интервали и табулации ще бъдат пропуснати, понеже са дефинирани от клас D.

Нека входният поток е:

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

Тогава при всяко извикване на TokenSearch ще бъдат разпознати последователно следните лексеми (в скоби са дадени имената им):

A-äóìà(LIST1) C-äóìà(:=) C-äóìà(=) B-äóìà(++) A-äóìà(A) B-äóìà(-)
A-äóìà(B) A-äóìà(C) A-äóìà(D) C-äóìà(,) C-äóìà(,) A-äóìà(E1) E-ñèìâîë
(àêî ñå îáðàáîòè îò äðóãà ïðîöåäóðà ìîæå äà áúäå ðàçïîçíàòî öÿëîòî
÷èñëî 123) A-äóìà(C)

Правилата за разпознаване на лексемите, входният поток (буфера) и таблиците за ключови думи се задават чрез параметър от тип структура. Полетата в тази структура имат следното значение:

- IdName: указател към променлива от тип string с максимална дължина не по-малка от стойността, записана в полето IdSize. Стойността на IdNameˆ при активиране на TokenSearch е без значение. В този низ тя записва името на разпознатата дума от клас A или B;
- IdSize: максимален брой значещи символи в името на дума от клас A или B. Ако думата е по-дълга, тя се анализира цялата, но в IdNameˆ се записват само първите IdSize символи. Не се променя от TokenSearch;

- flags: байт, в който всеки бит има специфично предназначение. TokenSearch променя съдържанието само на бит 0. (Всички останали битове остават непроменени.):
  - бит 0 - съдържанието му при активирането на TokenSearch е без значение. Ако TokenSearch е достигнала края на буфера, този бит се вдига в единица. Например анализира се BEGIN. След неговото разпознаване бит 0 ще бъде 1, ако BEGIN се намира на края на буфера, и 0, ако BEGIN не е бил в края на буфера;
  - бит 1 - този бит указва разпознаването на ключовите думи от клас A или B да бъде независимо от това дали са написани с малки или главни букви на латиница. В такъв случай таблицата от ключови думи трябва да бъде дадена само с главни букви от латиницата;
  - бит 2 - този бит е аналогичен на бит 1, но действието му е за буквите от кирилицата;
  - бит 3 - ако е 0, името на разпознатата лексема от клас A или B ще бъде записано (в IdName^) така, както се намира в текста. Ако е 1, името се записва след предварително превръщане на всички букви от латиницата в главни или малки (в какви точно указва бит 5);
  - бит 4 - този бит е аналогичен на бит 3, но действието му е за буквите от кирилицата;
  - бит 5 - указва към какви трябва да бъдат превърнати буквите, участващи в името на дума от клас A или B (виж битове 3 и 4). Ако бит 5 = 1, превръщането се извършва в малки букви, ако е 0 - в главни;
  - бит 6 - не се използва, но стойността му трябва да е 0;
  - бит 7 - указва дали за всички символи е зададена принадлежността им към символни класове или това се отнася само за първите 128 символа (7 битов ASCII код). Ако бит 7 = 1, то таблицата за принадлежност на символите е с големина 128 байта и се използва 7 битов ASCII код, т.е. предполага се, че всички символи с код по-голям от 128 не принадлежат към нито един от класовете. Ако бит 7 = 0, използва се 8 битов ASCII код и таблицата за принадлежност е с големина 256 байта;
- buffer: указател към буфера на входния поток. Ако за буфер се използва низ (string), то указателят трябва да сочи към първият му елемент (а не към нулевия). Не се променя от TokenSearch;
- buffsz: размер на входния буфер в байтове. Не се променя от TokenSearch.
- index: индекс на началния символ в буфера, от който ще се започне сканирането. Променя се от TokenSearch и след нейното завършване е индекса на елемента, до който е стигнало сканирането;
- start: индекс на символ в буфера, от който започва текущо разпознатата лексема. Стойността му е без значение при активиране на TokenSearch;
- KeyWordNo: Ако лексемата е разпозната като ключова дума от клас A, B или C, то това е номерът на ключовата дума в таблицата от ключови думи за съответния клас (A, B, C);
- Classes: указател към масив от 128 или 256 еднобайтови елемента (в зависимост от използвания ASCII код: 7 или 8 битов - зависи от бит 7 на flags). Елемент с индекс N от този масив отговаря на символ с ASCII код N. Битовете от съответния байт указват принадлежност (1) или непринадлежност (0) към съответния клас: бит 0 за клас D, 1 - C, 2 - E, 3 - A, 4 - B, 5 - A_1, 6 - B_1, 7 - C_1. Например стойност $40_{10} = 28_{16} = 00101000_2$

на седемдесетия елемент на масива означава, че буквата F (с десетичен ASCII код 70) принадлежи на клас A и A_1;

- ClassAkw: указател към таблица за ключовите думи от клас A. Таблицата съдържа в произволен ред всички ключови думи, всяка от тях предхождана от един байт с броя на символите й. Край на таблицата е байт с нулево съдържание. Например, ако таблицата трябва да съдържа ключовите думи BEGIN, END, IF, THEN и ELSE, и е обявена като пакетиран масив от символи, то на нея може да се присвои следната стойност #5'BEGIN' #3'END' #2'IF' #4'THEN' #5'ELSE' #0;
- ClassBkw: указател към таблица за ключовите думи от клас B. Форматът е същия, както за ClassAkw^;
- ClassCkw: указател към таблица за ключовите думи от клас C. Форматът е същия, както за ClassAkw^;

За удобство се експортират константи, с които по-лесно се установяват стойностите на флаговете и принадлежността към класовете. Тези константи са следните:

- Class_D = $01; { клас D, разделители }
- Class_C = $02; { клас C (само от ключови думи) }
- Class_E = $04; { клас E обработван от програмиста }
- Class_A = $08; { клас A }
- Class_B = $10; { клас B }
- Class_A1 = $20; { клас A_1 , продължение на клас A }
- Class_B1 = $40; { клас B_1, продължение на клас B }
- Class_C1 = $80; { клас C_1, продължение на клас C }
- flag_EOB = $01; { EndOfBuffer достигнат е края на 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) }

Използването им ще покажем чрез следния пример. Нека да искаме да дефинираме всички букви от латиницата от клас A и A_1, буквите от кирилицата от клас B и B_1 и цифрите от клас E и B_1. Тогава можем да зададем всеки от байтовете-дефиниции за принадлежност на буквите от латиницата като Class_A + Class_A1, за буквите от кирилицата като Class_B + Class_B1 и за цифрите като Class_E + Class_A1.

Освен основната функция за лексически анализ модула UniLEX експортира още няколко процедури:

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

Действие: Задачата на GetInteger е да превърне цяло число (по синтаксиса на UniPascal) от символно представяне във вътрешно. По-просто казано прочита цяло число, но не от текстов файл, а от зададена променлива от тип низ. Параметрите й са следните: **S** е входен буфер със символното представяне на цяло число, **i** е началният индекс в него, **l** е максималният брой символи, които трябва да се анализират, започвайки от **i**-тия елемент. В параметъра-променлива **E** функцията връща броя на цифрите които е анализирала. Функцията IOresult ще даде стойност 0, ако числото е написано правилно (няма грешка) и стойност 144, ако има грешка в числото. Не се счита за грешка, ако в буфера има вярно представяне на число, последвано от някакъв друг символ. Например за 1234xyz се връща 1234 и **E** = 4. Грешка се индицира, ако няма символно представяне на

число (например ABC1234 или --1234). Водещите интервали се пропускат. Ако числото е реално, може да не настъпи грешка. Например 123.45 ще бъде анализирано като 123, параметърът **E** = 3. Но ако е написано .123, ще се даде грешка.

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

Действие: Аналогично на функцията GetInteger, с тази разлика, че тази функция превръща символното представяне на реално число (по синтаксиса на UniPascal) във вътрешно.

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

Действие: Обратна на GetReal. Превръща реално число от вътрешно представяне в символно. За символното представяне се изисква да има зададен от **W** брой цифри след десетичната точка. Общата дължина на резултатния низ **S** е толкова, колкото е необходимо. Тази процедура работи като стандартната процедура WRITE(R:0:W), но не над текстов файл, а над низа зададен чрез параметъра **S**.

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

Действие: Аналогично на Real2Str, но символното представяне се прави в експоненциална форма. **W** задава броя на символите (цифри, точка, порядък, знак), които трябва да съдържат символното представяне. Минималната стойност на **W** е 8, а максималната - 15 (както и при използване на WRITE(R:W)).

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

Действие: Аналогично на Real2Str, но се превръща цяло число в неговото символно представяне. В символното представяне има толкова позиции, колкото са му необходими, но максимумът е 11 (10 цифри и знак).

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

Действие: Аналогично на Int2Str, но се превръща цялото число от двойна дума в неговото символно представяне в шестнадесетична бройна система. В символното представяне има 8 символа, по един за всеки полубайт на двойната дума.

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

Действие: Аналогично на Long2Str, но се превръща цялото число от дума, а не от двойна дума. В символното представяне има 4 символа.

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

Действие: Аналогично на Long2Str, но се превръща цялото число от байт, а не от двойна дума. В символното представяне има 2 символа.

function UpCaseCyr(ch: char): char;

Действие: Работи като стандартната функцията UpCase, но освен латинските превръща в главни и малките буквите от кирилицата.

function LoCase(ch: char): char;

Действие: Работи като стандартната функцията UpCase, но вместо в главни превръща буквите в малки.

function LoCaseCyr(ch: char): char;

Действие: Работи като функцията UpCaseCyr, но вместо в главни превръща буквите в малки.

procedure UpCaseS(var s: string);

Действие: Превръща всички малки букви от латиницата, намиращи се в низа **S**, в главни.

procedure UpCaseSCyr(var s: string);

Действие: Работи като функцията UpCaseS, но освен латинските превръща в главни и малките букви от кирилицата.

procedure LoCaseS(var s: string);

Действие: Работи като функцията UpCaseS, но вместо в главни превръща буквите в малки.

procedure LoCaseSCyr(var s: string);

Действие: Работи като функцията UpCaseSCyr, но вместо в главни превръща буквите в малки.

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

Действие: Търси символ в даден буфер (низ, пакетиран масив от байтове или символи). Параметъра **CH** задава търсения символ, **S** - буфера, в който ще се търси, **I** - началния елемент, от който ще започне търсенето, **L** - максималния брой елементи, които трябва да се проверят. Ако **L** е по-голямо от нула, ще се търси от текущия елемент нататък (към края на буфера). А ако **L** е по-малко от нула - в обратна посока. Резултатът от функцията е число, показващо след колко елемента е намерен търсения символ, т.е. при 0 е намерен на място **I**, при 1 - на място **(I+1)**, и т.н. Резултатът е отрицателен, ако е търсено е било в обратна посока (т.е. ако **L** е отрицателно или с други думи знака на резултата съвпада със знака на параметъра **L**). Ако елементът не е намерен, функцията връща стойността на параметъра **L**. Най-просто резултатът от функцията трябва да бъде интерпретиран така: ако той е различен от **L**, следователно търсеният символ е намерен на място **I + резултата от функцията**.

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

Действие: Търси символ, различен от дадения, в поле от паметта. Работи аналогично на функцията ScanFor, но търси символ, не съвпадащ с дадения.

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

Действие: Сравнява съдържанието на две полета от паметта (низове, пакетирани масиви от байтове или символи). Параметърът **L** задава броя на байтове които трябва да бъдат сравнени. **S** и **D** задават съответно първото и второто поле от паметта, които ще се сравняват. **SI** и **DI** задават началния индекс, от който ще започне сравняването съответно в първото и във второто поле. Резултатът е броя на байтовете, които съвпадат от двете полета. В частност той е равен на **L** при еднакви полета и нула при се различаващи се с първите си байтове.