Window – Place

Среда разработки многооконных (многозадачных) приложений в среде контекстно-зависимой трехмерной графики OpenGL с поддержкой наложенных графических фрагментов и виртуальных процедур языка C++.

Реферат:

Window-Place – пакет программ[1], образующий стандартную контекстную среду программирования на языке C++ для реализации трехмерной научной графики с использованием стандартных операций OpenGL. Объектно-ориентированный комплекс Window-Place полностью обеспечивает интерфейс между программой, операционной системой Windows и внешними устройствами ЭВМ: компьютерными часами и интервальным таймером; внутренними растровыми и системными векторными шрифтами; графическим терминалом; клавиатурой и указателем «мышь»; другими внешними устройствами (*измерительной телеметрии*). Производный класс **Window** обеспечивает многооконный интерфейс на основе SDK Windows, создает независимый пользовательский интерфейс для работы с часами и таймером; а в моменты активности окна Window обеспечивает опросы и ожидание информации от клавиатуры, и непосредственно управляет переадресацией прерываний от указателя «мышь» до базового класса или наложенных страниц Place. Базовый для Window объект **Place** обеспечивает поддержание контекстно-зависимой среды программирования трехмерной графики на основе OpenGL, а также устанавливает режимы и контролирует параметры трехмерной графики и плоских прорисовок для текстовых отчетов, меню и справок. Виртуальные процедуры для масштабирования наложенных страниц, обработки прерываний от мыши и др., создают интуитивно понятное и визуально естественное поведение графических площадок и текстовых страниц. Однако, полиморфизм производных классов для прикладных пользовательских вычислительных объектов, допускает подмену всех базовых виртуальных функций, что может быть полезным для ускорения вычислений за счет отмены или упрощения изначально встроенных в Window-Place процедур для комплексного управления процессами визуализации.

Тип ЭВМ: IBM-PC 386.
Язык: Code::Blocks-10.5, MinGW-C++, OpenGL.
ОС: Win32, Linux-Wine
Объём программы *(исходного текста)*: 120 Кб
тестовые примеры *(исходного текста)*: 20 Кб

[1] © *Храмушин В. Н.* Роспатент № 2010615850, 2010.09.08. Заявка СахГУ № 2010614191, 2010.07.13.

Window – Place

Обобщенный класс объектов и операций трехмерной контекстной графики OpenGL с виртуальным управлением C++

Window-Place – пакет контекстно-зависимых процедур и виртуальных функций для поддержки программирования в графической среде **OpenGL**. Многооконный интерфейс на базе класса **Window** дополняется графическими и текстовыми фрагментами/площадками **Place** с помощью стековых уровней/слоев для активных изображений в пределах исходного контура окна **Window**.

Основные процедуры и функции (перечисление):

```
Window::Window()        // окно создается и визуализуется конструктором
void Locate( int X,int Y,int W,int H ) // простая переустановка окна
byte WaitKey()          // ожидание нового символа с клавиатуры
byte ScanKey()          // запрос символа без остановки программы
byte ScanStatus()       // запрос сопутствующего кода из буфера
```

и виртуальные процедуры для работы с интервальным таймером:

```
virtual void Timer()  // виртуальная процедура обработки прерываний
void SetTimer( DWORD mSec, bool(*)()=NULL );      // запуск таймера
void KillTimer()      // полный сброс и остановка интервального таймера
```

Базовый объект **Place** образует геометрическое поле контекстной графики OpenGL для стандартного окна **Window**, а также предназначен для наложения на него обособленных графических или текстовых площадок для представления трехмерной графики, текстовых страниц или управляющих элементов. С классом **Place** связаны виртуальные функции масштабирования, обработки прерываний от «мыши» и все операции со шрифтами и текстами.

```
Place* Place::Act   /// Точка контекстной привязки к активной площадке
Place::Place( byte = PlaceOrtho | PlaceAbove )        // конструктор
Place& Capture( Window *Win=Act->Base )        // связь с окном Window
void Area( int X,int Y,int W,int H )      // местоположение площадки
virtual void Active() // активизация конкретного графического контекста
void Active( void(*)( int W,int H ) )      // внешнее масштабирование
void Clear()          // расчистка ограниченной графической площадки
void Save()           // сохранение текущего изображения в памяти
void Rest()           // восстановление фрагмента экрана из памяти
void Show()           // проявление на экране фрагмента из буфера
void Free()           // принудительное освобождение площадки от окна
virtual void MousePress( int stat, int x, int y )      // процедуры
void MouseMove ( void(*Pass)(      int, int ) )
void MousePress( void(*Push)( int, int, int ) )
SIZE Alfabet( int size=16, const char* font="Courier",
                  byte weight=FW_DONTCARE, byte italic=false )
SIZE AlfaBit( byte *=NULL )          // старый русский растр: 8,14,16 x 8
SIZE AlfaRect( const char *str )    //     определение размеров текста
int  Text( Course Dir,Real x,Real y,Real z,const char*,...)
void Print( int x,int y,const char *fmt, ... )     // если y/x<=0, то
void Print( const char *fmt, ... )                 // — снизу/справа
```

Производный класс Window на базе Place

```
class Window: Place                    // стандартное окно Windows для OpenGL
```

Название:

Оконный интерфейс Window для OpenGL в среде Microsoft Windows.

Описание:

Класс **Window** открыто наследует все элементы базового класса – графической площадки **Place**, и замыкает на себя комплекс процедур для непосредственного доступа к внешней периферии: графическому экрану и клавиатуре, с поддержкой операций для доступа к интервальному таймеру и компьютерным часам.

Объект **Window** автоматически создает новое независимое окно в среде Microsoft Windows, с которым изначально связывается стандартное фоновое графическое поле **Place** в качестве открытого базового класса.

Для выбора размеров окна относительно полного графического экрана ЭВМ можно использовать макросы с обращением к функциям Win32:

```
#define  Xpm( X ) ( GetSystemMetrics( SM_CXSCREEN ) * Real( X )/100.0 ) // %%X
#define  Ypm( Y ) ( GetSystemMetrics( SM_CYSCREEN ) * Real( X )/100.0 ) // %%Y
```

Информацию о размерностях графического экрана на момент создания нового окна Windows хранится во внутренних константах объекта Window:

```
const int ScreenWidth, ScreenHeight   // полные размеры экрана ЭВМ
```

Доступны также и другие внутренние данные присоединенного базового класса **Place**, которые определяют геометрические характеристики и задают стандартные операции для охватывающего окна **Window** в целом (*описание в* **Place**).

При создании самого первого окна **Window** в программе устанавливается внутренняя статическая ссылка Window::**First**, служащая началом списка активных окон.

Активность **Window** определяется наличием хотя бы одной присоединенной страницы для работы с графикой и текстом, адрес которой должен содержаться в вершине стека наложенных площадок Window::(**Place**)**Peak**. При создании нового окна **Window** эта ссылка указывает его собственный базовый класс **Place**.

Если ссылка **Window**::(**Place**)**Peak** обнуляется, это означает закрытие основного окна **Window**, и дополнительные действия с ним становятся недопустимыми (*невозможными*).

В неявном конструкторе класса Window использованы следующие значения по умолчанию:

```
Window::Window( char* Title=NULL,
                int X=0,int Y=0, int Width=0,int Height=0 )
```

что определяет простое окно без рамки с графическим полем 640x480;

Если указан заголовок **Title**, то создается стандартное окно Windows с активной рамкой и верхним заголовком с управляющими кнопками.

Если заголовка нет (**Title**=*null*), то создается простое окно фиксированного размера без активной рамки. Размеры такого окна невозможно изменить извне, что снимает необходимость перерисовки по внешним прерываниям.

Числовые параметры **X**, **Y**, **Width** и **Height** определят местоположение и размеры полного графического поля внутри Window.

X, **Y** – положительные величины определяют местоположение левого верхнего угла { 1,1 } нового окна Window, отрицательные величины – задают соответствующие отступы от правой и нижней границы графического экрана ЭВМ. Нулевые значения **X**, **Y** – ставят окно на четверть отступа сверху и треть – справа.

Width и **Height** – ширина и высота выделяемого окна Windows. Нулевые значения заменяются величинами 640x480 – соответственно; отрицательные или слишком большие значения приводят к установке максимальных размерностей окна в пределах всего графического экрана ЭВМ. Обрамляющие рамки Windows добавляются к заданным размерам Width и Height.

Для динамического изменения размеров и местоположения окна Window предназначена процедура **Locate**, числовые параметры **X**, **Y**, **Width** и **Height** интерпретируются также, как и в вышеописанном конструкторе:

```
void Window::Locate( int X, int Y, int Width, int Height );
```

Отсчеты местоположения и размеров окна могут быть заданы в процентах относительно экрана ЭВМ с помощью функций – макросов: **Xpm**(X) и **Ypm**(Y). При определении реальных параметров окна, по необходимости смещаются контрольные отсчеты местоположения – **X**, **Y** в пользу поддержания максимально возможных величин – **Width** и **Height**.

Название:

Подборка основных процедур для работы с клавиатурой и – удовлетворения внутренних очередей Windows

Описание:

Три функции для управления программой с помощью клавиатуры связаны с конкретным окном **Window**, и все посылки с клавиатуры сохраняются в его кольцевом буфере до момента выборки внутри в программы:

```
byte Window::WaitKey();       // ожидание нового символа с клавиатуры
byte Window::ScanKey();       // запрос символа без остановки программы
byte Window::ScanStatus();    // запрос сопутствующего кода из буфера
```

Функции **`WaitKey`** и **`ScanKey`** выбирают по одному символу из буфера. Функция **`ScanStatus`** считывает признаки сопутствующих <***Shift***>, <***Alt***> и <***Ctrl***> клавиш, которые были использованы в момент успешного ввода символа в буфер клавиатуры, и могут принимать следующие значения/маски:

```
RIGHT=1,  LEFT=2,   SHIFT=3,   // 0x03
LCTRL=4,  RCTRL=8,  CTRL=12,   // 0x0C
L_ALT=16, R_ALT=32, ALT=48.    // 0x30
```

Как и все клавиатурные функции, **`ScanStatus()`** ожидает завершения всех внутренних операций и прерываний Windows, попадающих в выполняемую программу. Учитывая, что реально **`ScanStatus()`** не оказывает никакого воздействия на среду исполняемой программы, она может быть использована в цик-

лах сложных математических расчетов для оживления графического окна и поддержания активного (*параллельного*) интерфейса с клавиатурой, мышкой и графическим экраном ЭВМ.

Предусмотрены команды/функции клавиатуры, выполняемые независимо от текущего состояния основной программы:

<***Ctrl***+C> – нормальное завершение с исполнением всех деструкторов;

<***Alt***+*LeftMouse-move*> перемещение окна по экрану ЭВМ.

Все три процедуры **`WaitKey`**(), **`ScanKey`**() и **`ScanStatus`**() доступны для контекстного вызова без явного указания окна **Window**, что приведет к его динамическому подключению по ссылке из активной площадки: **Place::Act**. …

Название:

Комплекс процедур интервального таймера

Описание:

С каждым окном **Window** может быть связан только один виртуальный таймер, получающий управление по заданному интервалу времени в такте выборки прерываний на исполнение внутренних очередей программы в Windows. Прерывания таймера отслеживают достигнутое время индивидуально для каждого окна **Window**, и в случае необходимости ожидания повторные вызовы внешних процедур обработки прерываний от таймера блокируются.

С окном **Window** связаны три программы для работы с таймером, включая виртуальную процедуру Timer, в которой производится фоновая предустановка, настройка и масштабирование контекстной среды OpenGL, с последующим вызовом процедуры обработки прерываний по адресу void(*__inTime__)(), задаваемому при установке таймера: **SetTimer**.

```
void Window::SetTimer( DWORD mSec, bool(*inTime)() )
virtual void Window::Timer(); // виртуальный прародитель прерываний
void Window::KillTimer()      // сброс — установка нулевого интервала
```

Первым параметром **SetTimer** задается интервал времени **mSec** в миллисекундах. Адрес внешней функции обработки прерываний таймера — **inTime**. Для запуска интервального таймера обязательно указание непустых параметров: **mSec** и **inTime**. Если заданный интервал меньше реального времени исполнения внешней процедуры обработки прерывания, то пропущенные шаги по времени не восполняются.

Встроенная виртуальная процедура Window::**Timer**() требует избыточного времени на последовательное пересохранение всех наложенных площадок **Place**, с их последующим восстановлением. Таким образом, при получении управления по таймеру графическое окно временно освобождается от всех наложенных площадок, затем, в очищенной и подготовленной к безопасному использованию контекста OpenGL, происходит обращение в адрес процедуры bool(*__inTimer__)().

Чтобы отказаться от этих процедур и ускорить обработку прерываний, виртуальная процедура **Timer** может быть заменена одноименной виртуальной процедурой в производном (охватывающем) классе от **Window**, что одновременно отключит вызов внешней процедуры bool(*__inTimer__)(), которая, по необходимости, может снова получить управление прямым вызовом только из виртуально переопределенной процедуры **Timer**.

Внешний внеклассовый обработчик прерываний **inTimer** возвращает значение «**true**», если внутри процедуры произведено изменение изображения в фоновом графическом буфере. В этом случае будет вызвана быстрая перерисовка изображения с помощью Place::Show(), т.е. без сохранения рисунка, выполняемого Refresh();. Если же **inTimer** не работает с графикой, или самостоятельно обновляет активное изображение, то для ускорения работы **inTimer** должна вернуть «**false**».

Следующие глобальные переменные и функции дают доступ к использованию компьютерных часов:

```
DWORD   StartTime,          // время начала исполнения программы от запуска Windows
        GetTime(),                  // в миллисекундах = timeGetTime = GetTickCount
        ElapsedTime();                      //! опрокидывание через ~49,7 суток
```

Базовый класс: Place – контекстная графическая и текстовая среда наложенных страниц

`class` **`Place`** // графическая площадка/страница на поверхности окна Window

Название:

Основной графический объект, обеспечивающий контекстную графику и стандартные текстовые операции средствами **OpenGL** на специально выделенных площадках в поле **Window**.

Описание:

С базовым классом связываются все контекстные операции OpenGL, а также системнозависимые утилиты для позиционирования и сохранения растровых полей; выбора шрифтов и представления текстовых строк в графическом и страничном форматах; обработки прерываний от указателя «мышь» и др.

```
Place* Place::Act   /// Точка контекстной привязки к активной площадке
Place::Place( byte = PlaceOrtho | PlaceAbove ) // конструктор
```

```
struct Window;              // родительский класс определяет рабочее окно Windows
struct Place                // базовый класс графической площадки/текстового листа
{ Window *Base;               // опорный (для Place) контекст окна Window в Windows
  Place *Prev;                // наличие нижней площадки, вершина Peak - в Window
  int  *Img;                  // временное хранилище фонового графического образа
  int pX,pY,Width,Height;     // положение и размеры на родительском окне Window
  HFONT hFont;                // шрифт сохранятся подключенным к hDC Windows
  byte *BitFont;              // временная установка старого растра из DispCCCP
  int FontBase,               // индекс растрового списка TT/Win шрифта
      FontX,FontY,            // текущая позиция печати по экрану как по листу
      FontWidth,FontHeight,   // заданные ширина и высота условного TT-символа
      MouseState;             // внутренний статус мышки
  byte Signs;                 // особые режимы/признаки управления страницей Place
static Place *Act;            //! Адрес активной страницы - контекстной среды
```

…

Конструктор новой площадки **Place** создает чистую заготовку, предварительно связанную с исходным контекстом программы Window::**First**, что необходимо для доступа к контексту внутренних настроек графической площадки. В качестве обязательного параметра при конструкторе указывается маска битов для установки режимов использования новой наложенной площадки:

```
enum{ PlaceAbove=0x80,   // сохранность фоновой подложки
      PlaceOrtho=0x40 }// ортогональность координат внутри страницы
```

Бит **`PlaceAbove`** x80 указывает на необходимость включения алгоритмов автоматического контроля и восстановления фонового изображения под графической площадкой.

Бит **`PlaceOrtho`** x40 используется внутри виртуальной процедуры `Place::`**`Active`** и указывает на установку режима автоматического ортогонального масштабирования графической площадки, при котором внутрь видимого поля включается представление пространственного куба с граничными размерами: `X[-1:1]; Y[-1:1]; Z[-1:1].`

Если бит **PlaceOrtho** выключен, то в качестве физических границ площадки устанавливаются ее растровые размерности [pX,pY, **Width,Height**]. Вертикальный отсчет ведется снизу – вверх. pX, pY – местоположение площадки внутри Window. Естественный режим удобен для работы с текстами, для которых обычно известны растровые размеры шрифтов (**FontWidth**, **FontHeight**) и полей печатаемых строк (**AlfaRect**(*str*){ *cx,cy* }).

Первой утилитой для встраивания новой площадки Place в управляющие списки активного окна Window является процедура Capture. В продолжение построения графической и текстовой среды необходимо выбрать шрифт из списка TrueType имен в Windows, установить местоположение и размеры площадки и др.

Освобождение площадки от связанного окна, при сохранении ее внутренних свойств, выполняется с помощью процедуры **Free**().

Название:

Управление контекстной графической средой Place

Описание:

Наложенные графические площадки обеспечивают независимый интерфейс для управления фрагментами растрового поля в окне Window, обеспечивающие привычную среду представления фрагментарной графики для OpenGL.

```
//!                определения и процедуры управления наложенными фрагментами экрана
//             реализуется создание, позиционирование, активизация и очистка фрагментов
Place& Place::Capture( Window=Act->Base )  // привязка к активному Window
void Place::Free()    // истинная свобода исключает всякие связи и обязанности
```

Процедуры **Capture** и **Free** подключают и освобождают площадку **Place** в управляющих списках базового окна **Window**. Каждая площадка может быть связана только с одним окном **Window**, а повторный вызов процедуры **Capture** выполняет соответствующее переподключение к другому окну **Window**.

~~В результате подключения Place::**Capture**(&Window) адрес новой площадки устанавливается на вершину наложенных фрагментов Window::**Peak**, забирая предыдущее значение адреса во внутреннюю ссылку Place::**Prev**.~~

В результате подключения Place::**Capture**(&Window) адрес новой площадки встраивается в стековый список строго над текущей активной площадкой Place::**Act**, забирая предыдущее значение адреса в свою внутреннюю ссылку Place::**Prev.** Это автоматически обеспечивает возврат к исходной странице при освобождении площадки утилитой Place::**Free**(). Собственно вершиной стека, одновременно определяющей порядок наложения графических фрагментов **Place**, является адрес в структуре базового окна Window::**Peak**.

Если требуется определение размеров и местоположения наложенной площадки в отсчетах количества символов, то необходима предварительная установка шрифтов с помощью процедур **Alfabet** или **AlfaBit**, иначе такие размеры будут определяться в отсчетах точек растра.

```
void Place::Area( pX,pY,Width,Height )    // местоположение и размерности
```

Установка местоположения и размеров наложенного в окне Window графического фрагмента.

Если pX, pY > 0 – отсчеты местоположения выполняются от левого–верхнего угла Window, иначе – от правого нижнего.

Если Width, Height > 0 – размеры площадки устанавливаются в количестве символов предустановленного шрифта. Если Width или Height = 0, то размеры площадки определяются по краю соответствующей границы окна Window. Если Width, Height < 0 – размеры площадки определяются в растровых отсчетах, с установкой правой правой системы координат – ось Y – снизу вверх.

Если шрифт предварительно не устанавливался, то размеры площадки определятся по тому же самому алгоритму – с размерами шрифта – в одну точку [1x1].

Если площадка создавалась с указанием режима PlaceAbove, то в процедуре Area выполняется сохранение фонового изображения.

По завершении процедуры Area происходит установка фокуса графического контекста с помощью вызова Place::Active.

```
virtual void Place::Active()              // установка контекста Place
void Place::Active( void(*)( int W, int H ) )   // - внешнее масштабирование
```

Установка масштабов и фокуса графического контекста **Place**. Присоединяемая процедура inActive(W,H) получает управление от виртуальной процедуры Active с установленным режимом масштабирования `GL_PROJECTION`, который не должен переключаться.

Если в конструкторе **Place** указано требование ортогональных координат (**`PlaceOrtho`** x40), то внешняя функция inActive получает из виртуальной процедуры `Place::Active` предустановленные ортогональный объем, полностью вмещающий в себя единичный куб с граничными размерами: `X[-1:1]; Y[-1:1]; Z[-1:1]`. Ось X направлена слева-направо, Y – снизу-вверх, Z – из экрана на наблюдателя. Это нейтральная разметка для единичной матрицы, к которой применимо простое и вполне адекватное перемасштабирование. Так вызов glOrtho(0,1, 0,1,-1,1) переключит масштаб на вмещение куба: : `X[0:1]; Y[0:1]; Z[-1:1]`.

Если в конструкторе **Place** опущен бит **`PlaceOrtho`**, то внешняя процедура масштабирования получает исходное поле со следующими границами: `X[0:`**Width**`]; Y[0:`**Height**`]; Z[-1:1]` (ось Y направлена снизу-вверх), что удобно для работы с растровыми изображениями и текстами.

Ничто не мешает подменить в охватывающем родительском классе виртуальную процедуру Place::**Active**, что позволит, к примеру, сократить время масштабирования и исключить излишний вызов ViewPort с последующим восстановлением исходной среды средствами OpenGL.

```
void Place::Clear()     // расчистка ограниченной графической площадки
void Place::Save()      // сохранение текущего изображения в памяти
void Place::Rest()      // восстановление фрагмента экрана из памяти
void Place::Show()      // проявление на экране фрагмента из буфера
void Place::Free()      // принудительное освобождение страницы от окна
void Place::Refresh()   // обновление картинки с сохранением ее в памяти
```

Название:

Транзакции обработки прерываний от указателя «мышь»

Описание:

Два варианта прерываний от указателя «мышь» предусматривают передачу управления при свободном движении над конкретной площадкой, и при движении с нажатыми кнопками:

```
virtual void Place::MouseMove( int X, int Y )          // свободное движение
virtual void Place::MousePress( int But, int X, int Y )  // — с кнопкой But
```

Обе процедуры включаются в работу только при условии предварительного подключения внешних процедур обработки прерываний:

```
void Place::MouseMove  ( void(*inPass)(          int X, int Y ) )
void Place::MousePress( void(*inPush)( int But, int X, int Y ) )
```

При вызове внешних независимых процедур обработки прерываний от указателя мышь: inPass и inPush, происходит предварительное переключение окна Window, сохранение текущего графического контекста OpenGL, а ссылке Place::**Act** присваивается соответствующий адрес на площадку под указателем «мышь». Собственно вызов утилиты масштабирования: Active() не выполняется. По завершении прерывания восстанавливается фокус активности исходного окна Window с и графический интерфейс OpenGL, что, как правило, достаточно для безаварийного продолжения работы прерванных операций.

Название:

Подборка растровых и Truetype шрифтов

Описание:

Предусмотрена работа со стандартными шрифтами Windows, вызываемыми по их названию, а также со старинными шрифтами из коллекции **DispCCCP** в трех вариантах: **_8x08**; **_8x14**; **_8x16**, где русские буквы прорисованы тонкими линиями, а латинские — жирными.

```
SIZE& Place::Alfabet( int=16, const char*="Courier", // установка TT-шрифта
                      byte weight=FW_DONTCARE, byte italic=false )// Windows
unsigned char _8x08[],_8x14[],_8x16[]; // просто русские растровые шрифты
SIZE& Place::AlfaBit( DispCCCP )  // установка растрового шрифта
SIZE& Place::AlfaRect( char* )    // растровые размерения надписи
```

Процедура **AlfaRect** выдает размеры растрового представления строки, что может быт использовано, например, для предварительной расчистки.

```
void Print( int X, int Y, const char *_fmt,  ...  )   // лист  y/x<=0 — снизу/справа
void Print( const char *_fmt,  ... )                  // контекстная печать
```

Две процедуры позволяют печатать тестовые строки на графической площадке, как по писчему листу, с отсчетом первой позиции печатаемой строки от верхнего-левого угла при положительных X,Y, и от правого-нижнего при отрицательных X,Y, соответственно. В процедурах **Print** допускается многократное использование символа '\n' для перехода на новую строку.

`int` **`Text`** (Course Dir, Real X, Real Y, Real Z, const char* _fmt, ...) *// 3D-подписи*

Текст печатается на графическом поле, при этом выбор **Course** показывает отступ текста от контрольной точки X,Y,Z:

Center – указанные координаты приходятся на центр надписи;

North – со смещением вверх; **South** – вниз;

West – влево, **East** – вправо.

В стандартной русской кодировке Windows-1251 имеются следующие специальные символы:

‘°’ – B0 ‘Ё’ – A8 ‘©’ – A9 ‘§’ – A7
‘±’ – B1 ‘ё’ – B8 ‘®’ – AE ‘«’ - AB
‘≠’ – 87 ‘№’ – B9 ‘™’ – 99 ‘»’ – BB ... 86

Последняя кодовая строка «0xF0 – 0xFF» из альтернативной (OEM-866) кодировки в DOS, перенесена в позицию «0x80 – 0x8F» и содержит следующие символы: ‘≡±≥≤ ⌠⌡ ÷≈°•·√ⁿ²■’, устанавливаемые функцией для старого русского растра **`AlfaBit`**(_8x08 | _8x14 | _8x16).

```
//
//!  запросы в виде наложенных на графическое поле текстовых страничек
//
byte Help( const char *N[],const char *C[],const char *P[], int
    X=-1,int Y=1 );
//   Name[0,1-3] Заголовок и строки расширенного названия
//   Text       Парное описание основных команд
//   Plus       Парное описание дополнительных команд
//    ++  определение каждого блока строк заканчивается нулевым адресом
//
struct Mlist{ short skip,lf; const char *Msg; void *dat; };
#define Mlist( L ) L,( sizeof( L )/sizeof( Mlist ) )
int TMenu( Mlist *M, int Nm, int x=1, int y=1, int ans=0 );
//
//     Mlist - Список параметров одного для запроса на терминал
//     skip    :пропуск строк             --> номер сроки
//     lf      : 0 - запрос не производится --> длина входного сообщения
//     Msg     :NULL - чистое входное поле  --> выходной формат
//     dat     :NULL & lf<>0 - меню-запрос  --> адрес изменяемого объекта
//
class TextMenu: Place // запрос текстового меню с отсрочкой полного завершения
{ int Y,X,Lx,Ly,  // местоположение на экране (++/слева-сверху, --/снизу-справа)
      K,          // номер редактируемого поля / последнего обращения
      Num;        // количество строк меню
  Mlist *M;       // собственно список меню Mlist/mlist
//void(*)(int);   // прерывание/подсказка при переходе на новый запрос из меню
  bool Up;        // признак установки меню на экране
public:           //
  TextMenu( Mlist*,int, int=1,int=1 ); ~TextMenu();
  void Active(); // локальная активизация графического контекста новой площадки
  int Answer( int=-1 ); void Back(){ Up=false; Free(); }
};
```

Приложение 1. Операции C++

Арифметические операции выполняются слева → направо,
или справа ← налево, в порядке старшинства операций:

Первичные и постфиксные

[] $\rightarrow_{16}$ *индексация массива*

() $\rightarrow_{16}$ *вызов функции*

. $\rightarrow_{16}$ *элемент структуры*

-> $\rightarrow_{16}$ *элемент указателя*

++ $\rightarrow_{15}$ *постфиксный инкремент*

-- $\rightarrow_{15}$ *постфиксный декремент*

Одноместные операции

++ $\leftarrow_{14}$ *префиксный инкремент*

-- $\leftarrow_{14}$ *префиксный декремент*

~ $\leftarrow_{14}$ *поразрядное NOT*

! $\leftarrow_{14}$ *логическое NOT*

- $\leftarrow_{14}$ *унарный минус*

& $\leftarrow_{14}$ *взятие адреса*

***** $\leftarrow_{14}$ *разыменование указателя*

(*тип*) $\leftarrow_{14}$ *приведение типа*

`sizeof` $\leftarrow_{14}$ *размер в байтах*

Мультипликативные

***** $\rightarrow_{13}$ *умножение*

/ $\rightarrow_{13}$ *деление*

% $\rightarrow_{13}$ *взятие по модулю*

Аддитивные

+ $\rightarrow_{12}$ *сложение*

- $\rightarrow_{12}$ *вычитание*

Поразрядного сдвига

<< $\rightarrow_{11}$ *сдвиг влево*

>> $\rightarrow_{11}$ *сдвиг вправо*

Отношения

< $\rightarrow_{10}$ *меньше*

<= $\rightarrow_{10}$ *меньше или равно*

> $\rightarrow_{10}$ *больше*

>= $\rightarrow_{10}$ *больше или равно*

== $\rightarrow_{9}$ *равно*

!= $\rightarrow_{9}$ *не равно*

Поразрядные

& $\rightarrow_{8}$ *поразрядное* AND

^ $\rightarrow_{7}$ *поразрядное* XOR

| $\rightarrow_{6}$ *поразрядное* OR

Логические

&& $\rightarrow_{5}$ *логическое* AND

|| $\rightarrow_{4}$ *логическое* OR

Условные

? : $\leftarrow_{3}$ *условная операция*

Присваивания

= $\leftarrow_{2}$ *присваивание*

***=** $\leftarrow_{2}$ *присвоение произведения*

/= $\leftarrow_{2}$ *присвоение частного*

%= $\leftarrow_{2}$ *присвоение модуля*

+= $\leftarrow_{2}$ *присвоение суммы*

-= $\leftarrow_{2}$ *присвоение разности*

<<= $\leftarrow_{2}$ *присвоение левого сдвига*

>>= $\leftarrow_{2}$ *присвоение правого сдвига*

&= $\leftarrow_{2}$ *присвоение* AND

^= $\leftarrow_{2}$ *присвоение* XOR

|= $\leftarrow_{2}$ *присвоение* OR

, $\rightarrow_{1}$ *запятая*

лаб. Вычислительной гидромеханики и морских исследований
почта: Khram@Mail.ru