Введение.

Стандартом Фортрана не предусмотрены средства графического вывода, построения диалогов.

Воспользовавшись приведенными в данной работе материалами, программист сможет организовать дружественный интерфейс (в виде диалоговых окон) между пользователем и выполняющими вычисления процедурами. Такие окна содержат управляющие элементы, поля ввода и вывода данных. Диалоги Фортрана обеспечивают полноценный интерфейс для большинства приложений. Но все же в случае сложной структуры интерфейса следует обратиться к другим программа (типа Delphi), предназначенных специально для разработки интерфейсов, так как создание такого интерфейса не целесообразно делать на Фортране.

Одновременно с диалогами в проектах QuickWin поддерживается многооконный графический вывод, управляемый из программы, из диалогов, из сопровождающего графические окна меню или процедурами, запускаемыми при нажатии на заданные клавиши клавиатуры или кнопки мыши. Имеющиеся в Фортране графические процедуры позволяют отображать растровые графические данные (в том числе и графический текст).

1. Использование диалогов.

1.1. Постановка задачи.

Исполняемый файл создадим как проект QuickWin. Такой проект создается в среде MS Developer Studio Fortran PowerStation 4.0 в результате выполнения цепочки:
File — New — Project Workspace — QuickWin Application — задать имя проекта (Name) и папку его размещения (Location) — Create.

Если осуществлять ввод данных в программе с экрана (использовать операторы print*, и read*,), то это может привести к тому, что, во-первых, такой ввод данных не нагляден, во-вторых, русский текст в DOS-окне претерпит искажения и, в-третьих, отображаемые в окне данные после перехода на следующую строку окна не могут быть отредактированы. Все эти недостатки можно устранить, создав диалог ввода данных.

1.2. Построение диалогового окна.

Диалог, после того как ясен его состав и геометрия расположения полей с данными и текстовых полей, создается в три этапа:

1) построение диалогового окна;
2) включение диалога в проект;
3) создание процедур инициализации и обработки полей диалога.

1.2.1. Проект для диалога.

Реализуем диалоговое окно, приведенное на рис.1.1.

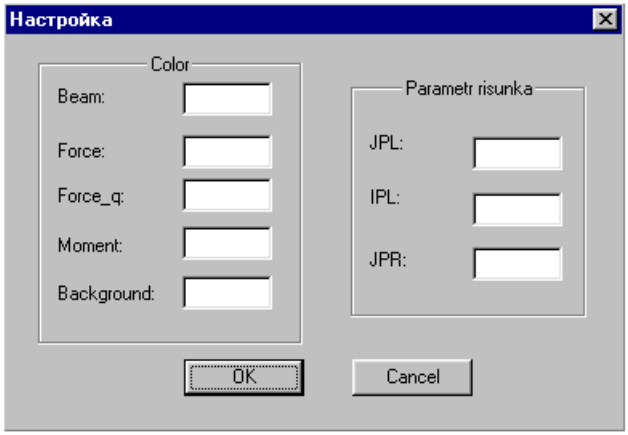


Рис.1.1. Проект диалогового окна.

Построение такого окна выполним в среде MS Developer Studio. Загрузим заготовку для диалога: Insert — Resource — выбрать Dialog — OK. В результате появится заготовка для окна с двумя кнопками — OK и Cancel.

1.2.2. Задание параметров диалога.

Настроим прежде параметры всего окна, щелкнув для этого 2 раза мышью по свободной поверхности окна. В появившемся окне Dialog Properties на вкладке General зададим:

1) в поле ID имя диалога — IDD_tun, по которому он будет идентифицироваться в программе;
2) в поле Caption название диалога — “Настройка”, которое отображается на верхней полосе диалога;
3) установим, нажав на кнопку Front, шрифт MS Sans Serif, и его размер, например 8;
4) в полях XPos и YPos координаты верхнего левого угла диалога, которые он имеет в дочернем окне QuickWin, например 150 и 100 (координаты задаются в текстовых столбцах и рядах).

Выберем затем вкладку Styles и оставим стоять галочки в пунктах Titlebar и System menu.

Остальные поля этой и других вкладок оставим без изменений (рис.1.2.).

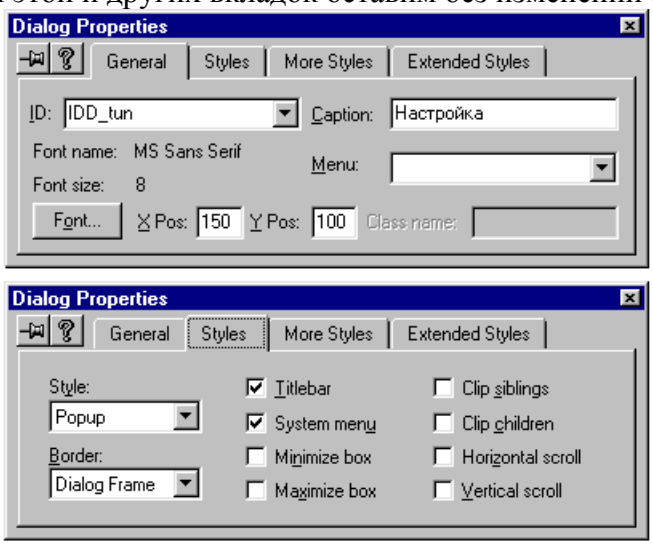


Рис.1.2. Параметры диалогового окна.

Закроем окно Dialog Properties, нажав, например, на клавишу Enter; увеличим привычным образом размеры диалогового окна; нажмем Ctrl+T и посмотрим получившееся окно. Нажмем Esc, с тем чтобы затем продолжить построение.

1.2.3. Задание и обработка статического текста.

Используя меню с управляющими элементами (рис.1.3.), зададим текстовые поля, группы и поля ввода и редактирования данных.



Рис.1.3. Управляющие элементы.

Задание текстового поля (Static text) выполняется после выбора управляющего элемента ***Aa*** и последующей прорисовки прямоугольника для предполагаемого текста в свободной области диалога. Прорисовка выполняется мышью при нажатии ее левой

клавише. (Можно также после выбора элемента управления просто “ударить” мышью по предполагаемому его месту размещения в диалоге.) Выделим получившейся прямоугольник, ударив по нему мышью, и изменим его положение и размеры, пользуясь мышью или клавиатурой. Так, увеличит размер по оси *x* можно, нажимая Shift+→. “Ударим” теперь по тексту 2 раза мышью и зададим в окне Text Properties во вкладке General свойство Caption текста: “Beam:” (рис.1.1.). Остальные свойства оставим без изменений.

Аналогичным образом зададим все иные тексты, приведенные на рис 1.1. Все тексты, поскольку они неизменяемы, будут иметь один и тот же задаваемый по умолчанию идентификатор (свойство ID): IDC_STATIC (или можно задать свой идентификатор: IDC_Col1).

Напомним, что для отмены действия можно использовать Ctrl+Z.

Задаваемый подобным образом текст не может быть изменен в диалоге.

1.2.4. Обработка редактируемых полей.

Выберем контрольный элемент ***ab***| и, применяя ту же технику, что и при создании статического текста, разместим в диалоге 8 редактируемых полей (Edit Box). Используем первое поле для ввода цвета балки, второе — для ввода цвета стрелок сил и т.д. (см. рис.1.1.) Присвоим полям имена: первому — *Col1_ed*, второму — *Col2_ed* и т.д. Имя поля задается в связанном с ним окне Edit Properties изменения свойств поля, которое появляется, если по полю дважды щелкнуть мышью. По умолчанию ID-имя первого поля имеет значение IDC_EDIT1, которое мы изменим на *Col1_ed*. Аналогичные операции повторим и для остальных редактируемых полей.

Перемещение по редактируемым полям и другим элементам управления диалога осуществляется при помощи мыши, клавиш Tab, Shift+Tab, Enter (Ввод и перемещение) и других клавиш.

Редактируемые поля содержат символьные данные. Для передачи из программы в символьные поля числовых данных выполняются преобразования “число — строка”. При вводе числовых данных из символьного редактируемого поля диалога выполняются преобразования “строка — число”.

Изменение значения поля при открытом диалоге можно выполнить из программы, применив функцию DLGSET. Функция DLGSET имеет логический тип и вернет .TRUE. в случае успеха. Например, после вызова

```
l4=dlgset(dlg,idc_col1_ed,c4)
```

в поле *Col1_ed* появится значение c4.

Передача данных из поля в программу выполняется в результате вызова функции DLGGET. Например, вызова

```
l4=dlgget(dlg,idc_col1_ed,c4)
```

обеспечит передачу значения поля *Col1_ed* в символьную переменную *string*. Функция DLGGET имеет логический тип и вернет .TRUE. в случае успеха.

Напомним, что последующее преобразование “строка — число” выполняется так:

```
read(c4,fmt4)cvb
```

Преобразование “число — строка” моно выполнить так:

```
write(c4,fmt4)cvb.
```

1.2.5. Задание и обработка группы.

Задание группы (Group Box) выполняется после выбора управляющего элемента  и последующей прорисовки прямоугольника для предполагаемой группы, объединяя в группу элементы статического текста (“Beam”, “Force”, “Force_q”, “Moment”, “Background”) и редактируемых полей, соответствующих своим текстам. Прорисовка выполняется мышью при нажатии ее левой клавише. Выделим получившейся прямоугольник, ударив по нему мышью, и изменим его положение и размеры, пользуясь мышью или клавиатурой. “Ударим” теперь по тексту 2 раза мышью и зададим в окне Group Box Properties во вкладке General

свойство Caption текста: “Color” (рис.1.1.). Выберем затем вкладку Styles и в пункте Horizontal alignment: поставим Center. Остальные свойства оставим без изменений.

Аналогичным образом зададим остальную группу. Все группы, поскольку они неизменяемы, будут иметь один и тот же задаваемый по умолчанию идентификатор (свойство ID): IDC_STATIC (или можно задать свой идентификатор: IDC_Col).

Задаваемая подобным образом группа не может быть изменена в диалоге.

1.2.6. Кнопки OK и Cancel.

Кнопки OK и Cancel появляются в каждом вновь создаваемом диалоге, и с ними связаны задаваемые по умолчанию процедуры. Можно изменить как ID-имена этих кнопок, так и названия. Разумеется, каждая из этих кнопок может быть удалена из диалога.

1.2.7. Меню диалога.

Помимо приведенного на рис.1.3. меню с управляющими элементами (Controls) при работе с диалогом можно пользоваться меню Dialog (рис.1.4.).



Рис.1.4.Меню диалога.

Это меню позволяет изменять положение и геометрию выбранных полей диалога, задавать сетку в проектируемом окне, линейку измерения размеров, просматривать (Ctrl+T) диалог.

Если выбрано одно поле, то его можно отцентрировать как по вертикали (Ctrl+F9), так и по горизонтали (Ctrl+Shift+F9). Если выбраны несколько полей, то они могут быть:

- выровнены влево (Align Left-Ctrl+←);
- выровнены вправо (Align Left-Ctrl+→);
- выровнены вверх (Align Top-Ctrl+↑);
- выровнены вниз (Align Bottom-Ctrl+↓);
- отцентрированы по вертикали и горизонтали;
- сделаны одной ширины (Make Same Width);
- сделаны одной высоты (Make Same Height);
- сделаны одного размера (Make Same Size).

Меню диалога можно убрать, нажав, например, на крест (×) (рис.1.4.). При необходимости можно восстановить, найдя в главном меню пункт Toolbars и поставив галочку после выбора Toolbars рядом с пунктом Dialog.

Аналогичным образом можно активизировать и ранее закрытое меню Controls.

1.2.8. Меню диалога.

При сохранении диалога создается файл ресурсов, которому мы присвоим имя iin.rc и который расположим в папке, содержащей исходный код. При сохранении диалога в папке с файлом iin.rc появляются или обновляются файлы resource.h и resource.fd, содержащие параметры диалога и других ресурсов.

Файл ресурсов может содержать диалоги, меню и другие ресурсы, однако пока в нем только один диалог — IDD_tun. Файл iin.rc, чтобы программа получила к нему доступ, необходимо вставить в проект. Выполним в FPS цепочку: Insert — File into Project — задать Resource files для типа файлов — выбрать файл iin.rc — Add.

Выполним компиляцию проекта. В среде MS Developer Studio — в окне работы с проектом (View window) — появится вкладка Resource view, переместившись в которую можно просмотреть и состав файла ресурсов iin.rc, и отдельно каждый из его компонентов. Существующее диалоговое окно будет загружено в среду MS Developer Studio, если дважды щелкнуть мышью по его ID-имени, отображаемому на вкладке Resource view. Появившееся окно доступно для редактирования.

На вкладке File View после компиляции проекта появится подраздел Dependencies, содержащий ссылку на файл resource.fd. Отметим, что пользователь должен воздержаться от ручного редактирования этого файла.

1.2.9. Работа с диалогом в программе.

программу реализуем по следующей схеме:

- выполним инициализацию диалога;
- отобразим, обратившись к функции DLGMODAL, диалог на экран;
- введем данные;
- нажмем кнопки OK и Cancel;
- при выборе OK или Cancel закроем диалог и завершим программу.

Функция DLGINIT имеет логический тип и вернет .TRUE. в случае успеха. Функция DLGINIT выполняет инициализацию диалога.

Функция DLGMODAL типа INTEGER(4) в случае успеха отображает диалог на экране и передает управление диалогу. В случае неудачи функция вернет –1. Функция имеет синтаксис:

result=DLGMODAL(*dlg*)

dlg — переменная типа *dialog*, связанная посредством функции DLGINIT с конкретным диалогом.

Подпрограмма DLGEXIT закрывает диалог и имеет синтаксис:

CALL DLGEXIT(*dlg*)

Закрытому по средством DLGEXIT диалогу можно передать управление, применив функцию DLGMODAL.

1.3. Управляющие элементы построенного диалога.

Таблица 1.1. Управляющие элементы для данного диалога (рис.1.1.)

Управляющий элемент	*Описание*
Статический текст	Изменяемый из программы текст
Редактируемое поле	Элемент, значение которого можно изменить с клавиатуры
Группа	Элемент, используемый для объединения элементов, в группу взаимодействующих элементов

1.4. Процедуры для работы с данным диалогом.

В FPS интерфейсы функций размещены в модуле DIALOGM.MOD.

Таблица 1.2. Процедуры для данного диалога (рис.1.1.)

Процедура	*Тип*	*Назначение*
DLGEXIT	Подпрограмма	Закрывает открытый диалог
DLGGET	LOGICAL(4)	Передает из диалога в программу значение компонента управляющего элемента
DLGINIT	“	Инициализация диалога
DLGMODAL	INTEGER(4)	Отображает диалог на экране и передает управление диалогу
DLGSET	LOGICAL(4)	Устанавливает значение компонента управляющего элемента

2. Вывод графических данных.

Графические процедуры могут быть использованы в проектах QuickWin (многооконный графический проект).

2.1. Графический дисплей.

В современных ЭВМ используются растровые дисплеи. Графические возможности растрового дисплея определяются размером экрана, зерном экрана, частотой регенерации и разрешением.

Размер экрана может варьироваться от 14 до 21 дюйма и более.

Вывод изображения выполняется в результате подсветки электронным лучом отдельных регулярно расположенных точек экрана. Такая точка является наименьшим элементом графической информации и называется *пикселем*.

Число таких точек по горизонтали и вертикали экрана определяется его *разрешением*. Стандартными являются разрешения 640×480 (первая цифра указывает на число точек по горизонтали, вторая — по вертикали), 800×600, 1024×768, 1280×1024 и более высокие разрешения. Управление разрешением осуществляется графическим адаптером.

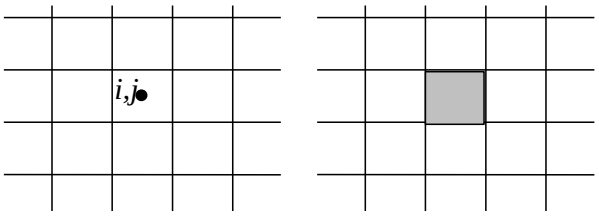
Луч последовательно “пробегает” все точки экрана, выполняя заданную программой для каждой точки подсветку. Затем цикл обхода повторяется. Полный цикл обхода всех точек экрана приводит к воспроизведению изображения и называется *регенерацией изображения*. Число регенерации в секунду называется *частотой регенерации*. Низкая частота регенерации приводит к заметному для человеческого глаза миганию изображения. Хорошим показателем можно считать частоту регенерации не менее 60 Гц.

Расстояние между соседними точками экрана определяет его *зерно*. Размеры зерна варьируются от 0,31 до 0,21 мм и менее. Понятно, что чем меньше зерно, тем выше качество изображения.

2.2. Растровое изображение.

В графике экран растрового дисплея представляется в виде прямоугольной сетки на дискретной плоскости с шагом по осям *x* и *y*, равным единице. Такая модель называется *растровой плоскостью* (рис.2.1.,а). Массив прямоугольных ячеек плоскости называется *растровым массивом*.

Каждый квадрат сетки соответствует одному пикселю экрана. Точка инициализации пикселя находится в центре квадрата сетки. Инициализация точки растровой плоскости с координатами (*i,j*) соответствует закраска квадрата сетки, в центре которого эта точка расположена (рис.2.1.,б).



а б

Рис.2.1. Модель растрового экрана:
а — растровая плоскость;
б — инициализация точки растровой плоскости.

Растровое изображение создается путем закраски ячеек растрового массива в тот или иной цвет. Растровое изображение можно сравнить с изображением на листе клетчатой бумаги, получаемым в результате закраски отдельных клеточек листа.

Каждому пикселю растровой плоскости могут быть независимым образом заданы цвет, интенсивность и другие характеристики.

2.3. Видеоадаптер.

Данные, подлежащие воспроизведению на экране монитора, формируются программой, функционирующей на центральном процессоре ЭВМ. Далее они передаются

видеоадаптеру, который размещает эти данные в видеопамяти, преобразовывает и передает устройству управления лучом электронно-лучевой трубке.

Таким образом, *видеоадаптер* является устройством сопряжения центрального процессора ЭВМ и устройства отображения.

Видеоадаптеры могут обеспечить разные режимы отображения данных, которые разделяются на *текстовые* и *графические*. Причем, как правило, видеоадаптер поддерживает несколько графических режимов. Режим отображения данных на экране называется *видеорежимом*. В MS Developer Studio видеорежим устанавливается при задании конфигурации видеоокна функцией SETWINDOWCONFIG.

Различные видеорежимы отличаются количеством выводимых на экран данных и цветов. Единицами измерения количества выводимых данных являются: в текстовом режиме — символ (литера), в графическом — пиксель.

2.4. Видеоокно и окна вывода.

В FPS в проектах с графикой вывод и текстовых и графических данных выполняется в *видеоокно*, параметры которого устанавливаются функцией SETWINDOWCONFIG. В проектах со стандартной графикой получаемое изображение отображается на полном экране без меню и вертикальной и горизонтальной полос прокрутки экрана. Затем видеоокно получает необходимые для него меню и полосы прокрутки.

В QuickWin каждое создаваемое (дочернее) видеоокно всегда располагается внутри обрамляющего окна. Размеры и положение обрамляющего и любого дочернего видеоокна могут быть изменены не только пользователем, но и из программы.

Вывод текстовых данных в видеоокно выполняется операторами PRINT, WRITE и подпрограммой OUTTEXT, чтение — оператором READ.

Внутри окна могут быть созданы прямоугольные текстовые области, называемые *окнами вывода*. По умолчанию окном вывода является все видеоокно, на которое и направляется как графический, так и текстовый вывод. С этим окном связана физическая система координат. Единицей измерения значений текстовых координат является знакоместо, а графических — пиксель. Число создаваемых окон вывода произвольно.

Замечание. Далее, как правило, будем употреблять слово “окно” вместо более громоздкого “видеоокно”.

В каждом программном компоненте, где вызываются графические процедуры, необходимо выполнить ссылку на модуль MSFLIB (USE MSFLIB). Этот модуль содержит интерфейсы к графическим процедурам, необходимые для работы с графикой именованные константы и определения производных типов данных.

2.5. Задание конфигурации видеоокна.

установка конфигурации окна не является обязательной процедурой, но при необходимости может быть выполнена функцией

flag4=SETWINDOWCONFIG(*wc*)

wc — переменная производного типа *windowconfig*, содержащая параметры окна. Тип *windowconfig* определен в модуле MSFLIB.

Можно вообще не использовать SETWINDOWCONFIG. В этом случае окно выберет наилучшее из возможных разрешений и размер шрифта 8×16. Число доступных цветов зависит от видеоадаптера.

2.6. Системы графических координат. Окно вывода.

Растровая плоскость расположена в *физической системе координат*, определяемой техническими средствами. Начало физической системы координат — левый верхний угол окна. Ось абсцисс направлена слева направо; ось ординат — сверху вниз (рис.2.2.). Физические координаты целочислены (пиксели). Используемый при задании координат тип данных — INTEGER(2).

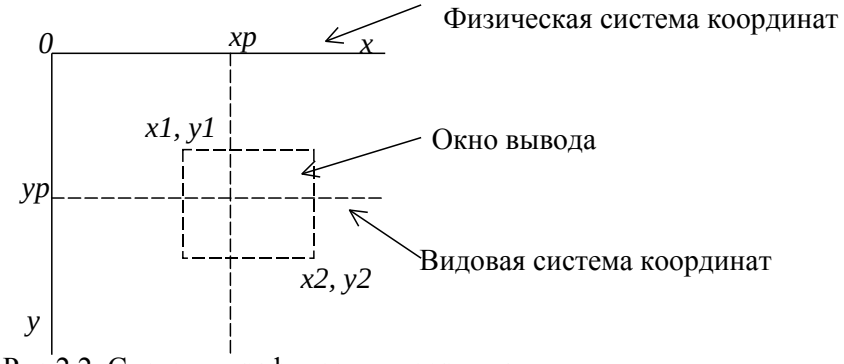


Рис.2.2. Системы графических координат.

Точка начала отсчета координат пикселей может быть перемещена в любую точку окна *xp, yp*. Эта процедура называется *заданием видового порта*. Так, после перемещения точки отсчета в центр видеоокна минимальные координаты пикселя при разрешении 800×600 будут равны (-400,-300), а максимальные — (399,299).

Видовой порт (видовую систему координат) и одновременно окно вывода можно задать одной подпрограммой

CALL SETVIEWPORT(*x1,y1,x2,y2*)

x1,y1 — координаты верхнего левого угла окна вывода;

x2,y2 — координаты нижнего правого угла окна вывода.

Тип параметров *x1,y1,x2,y2*— INTEGER(2).

Подпрограмма переопределяет графический видовой порт и задает окно вывода. То есть начало видовой системы координат после вызова

CALL SETVIEWPORT(*x1,y1,x2,y2*)

будет совпадать с верхним левым углом окна вывода.

Координаты (*x1,y1*) и (*x2,y2*) задаются в физической системе координат. Все последующие преобразования окна вывода посредством функции SETWINDOW выполняются в видовой, а не в физической системе координат.

Ранее рассмотрели системы с целочисленными координатами, диапазон изменения которых определяется разрешением дисплея и заданным окном. В FPS, однако, есть возможность задать окно вывода, связав его с *оконной системой координат* (ОСК), в которой используются вещественные координаты. Задаваемые в оконной системе координаты называются *оконными координатами*. ОСК может совпадать с *мировой системой координат*, т.е. с системой, в которой мы работаем с реальными объектами.

Окно вывода, в котором можно работать с вещественными координатами, задается в видовой системе координат функцией

result2=SETWINDOW(*finvert,wx1,wy1,wx2,wy2*)

finvert — параметр типа LOGICAL(2), определяющий направление координат (если *finvert* равен .TRUE., то ось *y* увеличивается от нижней границы видового порта к верхней; если *finvert* равен .FALSE., то ось *y* увеличивается от верхней границы видового порта к нижней, т.е. так же, как в видовой системе координат).

wx1,wy1 — верхний левый угол окна в ОСК;

wx2,wy2 — нижний правый угол окна в ОСК.

Тип параметров *wx1,wy1,wx2,wy2*— REAL(8). Тип функции — INTEGER(2). Функция возвращает отличное от нуля значение при успешном открытии окна и 0 в противном случае.

Функция SETWINDOW определяет ОСК, в которой координаты выводимых объектов ограничены значениями параметров *wx1,wy1,wx2,wy2*. В общем случае начало координат ОСК может находиться за пределами окна вывода.

В ОСК могут быть использованы для вывода графических элементов только функции, имена которых завершаются символами_W (например, LINETO_W).

Задание ОСК выполняется относительно текущего видового порта.

2.7. Очистка и заполнение экрана цветом фона.

Видеоокно, текущий видовой порт и текущее текстовое окно вывода очищаются после применения подпрограммы

CALL CLEARSCREEN (*area*)

area — определенная в модуле MSFLIB константа, указывающая на тип очищаемой области и принимающая значения:

$GCLEARSCREEN — очистка всего видеоокна;

$GVIEWPORT — очистка текущего видового порта;

$GWINDOW — очистка текущего текстового окна, заданного по средством SETTEXTWINDOW.

Помимо очистки заданная область заполняется цветом текущего фона.

2.8. Управление цветом.

2.8.1. Цветовая палитра VGA.

В режиме VGA с 16 цветами и разрешением 640× 480 пикселей по умолчанию устанавливается приведенная в табл.2.1. палитра цветов.

Таблица 2.1.Шестнадцатицветовая палитра VGA.

№	*Цвет*	*№*	*Цвет*
0	Черный	8	Серый
1	Синий	9	Светло-синий
2	Зеленый	10	Светло-зеленый
3	Голубой	11	Светло-голубой
4	Красный	12	Светло-красный
5	Фиолетовый	13	Светло-фиолетовый
6	Коричневый	14	Желтый
7	Белый	15	Ярко-белый

2.8.2. Управление цветом фона.

Задание текущего цвета фона по его номеру в цветовой палитре выполняется функцией

result4=SETBCOLOR(*index*)

index — параметр типа INTEGER(4), равный номеру цвета устанавливаемого фона.

Функция возвращает значение типа INTEGER(4), равное номеру цвета старого фона. По умолчанию устанавливается черный цвет фона.

Если же после вызова функции SETBCOLOR вызвать подпрограмму CLEARSCREEN, то выполниться очистка заданной области и ее заполнение заданным цветом.

2.8.3. Управление цветом графических примитивов.

Текущий цвет выводимых графических элементов: отрезка прямой, прямоугольника, многоугольника, эллипса, дуги, сектора — может быть задана функцией

result2=SETCOLOR(*index*)

index — параметр типа INTEGER(2), равный номеру устанавливаемого цвета.

2.9. Текущая позиция графического вывода.

Графические примитивы *текст* и *отрезок прямой* выводятся, начиная от текущей позиции. После перехода в графический режим текущая позиция находится в центре видеоокна. Изменение текущей позиции выполняется подпрограммой

CALL MOVETO_W (*wx,wy,wt*)

wx,wy — оконные координаты новой позиции типа REAL(8).

wt — оконные координаты предшествующей позиции (переменная типа *wxycoord*). Тип *wxycoord* определен в модуле MSFLIB.

При изменении текущей позиции изображение не меняется. Подпрограмма MOVETO_W изменяет текущую позицию в оконной системе координат.

2.10. Графические примитивы, используемые в данной работе.

Таблица 2.2.Процедуры вывода графических элементов.

Процедуры	*Отображаемые графические элементы*
LINETO_W	Отрезок прямой
RECTANGLE_W	Прямоугольник
POLYGON_W	Многоугольник
ELLIPSE_W	Эллипс или окружность
FLOODFIL_W	Заполнение (заливка) замкнутой области заданным цветом и по заданному шаблону

Для графического примитива могут быть установлены следующие характеристики:

- тип линии (например, сплошная, пунктирная);
- цвет линии;
- способ вывода(для отрезков, прямоугольников и многоугольников);
- шаблон и цвет заполнения (для замкнутых геометрических фигур).

Кроме того, вывод примитива может выполняться на заранее установленном фоне, т.е. после заливки окна вывода заданным цветом. По умолчанию цвет фона — черный, цвет линий — белый.

2.10.1. Вывод отрезка прямой линии.

Функция вывода отрезка прямой линии:

result2=LINETO_W(*wx,wy*)

wx,wy — оконные координаты конечной точки отрезка — REAL(8).

Функция возвращает значение типа INTEGER(2), которое, если отрезок нарисован, отлично от нуля или равно нулю в противном случае.

Отрезок выводится от текущей позиции до конечной точки, задаваемой параметрами *wx,wy*. При выводе отрезка используется текущий цвет, задаваемый функцией SETCOLOR. По умолчанию отрезки отображаются текущим цветом и выводятся сплошной линией

В случае успешного выполнения LINETO_W текущей становится соответствующая конечной точке позиция (*wx,wy*).

Замечание. Если область, границы которой образованы выведенными функцией LINETO_W отрезками прямых, заполняется посредством FLOODFIL_W, то для отображения границ области необходимо использовать сплошной тип линий.

2.10.2. Вывод прямоугольника.

Вывод прямоугольника выполняется функцией

result2=RECTANGLE_W(*control, wx1,wy1,wx2,wy2*)

control — флаг заполнения прямоугольника — INTEGER(2);

wx1,wy1 — оконные координаты левого верхнего угла прямоугольника;

wx2,wy2 — оконные координаты правого нижнего угла прямоугольника. Тип параметров *wx1,wy1,wx2,wy2*— REAL(8).

Функция возвращает значение типа INTEGER(2), которое, отлично от нуля при успешном выполнении или равно нулю в противном случае.

Границы прямоугольника рисуются с использованием текущего цвета, текущего способа вывода и текущего типа линии.

Параметр *control* может принимать значения определенных в модуле MSFLIB именованных констант $GFILLINTERIOR и $GBORDER.

Если *control* равен $GFILLINTERIOR, то осуществляется заполнение прямоугольника с использованием текущего цвета и шаблона заполнения. По умолчанию осуществляется сплошное заполнение замкнутой области.

Если *control* равен $GBORDER, то выводятся только границы прямоугольника с использованием текущих цветов, способа вывода и типа линии.

2.10.3. Вывод многоугольника.

Произвольной формы многоугольник выводиться функцией

result2= POLYGON_W (*control, wppoints, cpoints*)

Параметр *control* принимает те же значения и имеет тот же смысл и тип, что и одноименный параметр функции RECTANGLE_W.

wppoints — массив типа *wxycoord* из элементов, задающих ломанную линию;

cpoints — параметр типа INTEGER(2), задающий число точек ломанной линии.

Тип *wxycoord* определен в модуле MSFLIB.

Функция возвращает значение типа INTEGER(2), которое, отлично от нуля при успешном выполнении или равно нулю в противном случае.

Границы многоугольника вычерчиваются с использованием текущего цвета, текущего способа вывода и текущего типа линии.

Параметр *cpoints* указывает число используемых точек массива *wppoints*. Последняя выводимая точка массива всегда соединяется отрезком прямой с первой точкой.

2.10.4. Вывод эллипса и окружности.

Изображение эллипса и окружности выполняется функцией

result2= ELLIPSE_W (*control, wx1,wy1,wx2,wy2*)

control — флаг заполнения эллипса — INTEGER(2);

wx1,wy1 — оконные координаты левого верхнего угла ограничивающего эллипс прямоугольника;

wx2,wy2 — оконные координаты правого нижнего угла ограничивающего эллипс прямоугольника. Тип параметров *wx1,wy1,wx2,wy2*— REAL(8).

Функция возвращает значение типа INTEGER(2), которое, отлично от нуля при успешном выполнении или равно нулю в противном случае.

Границы эллипса вычерчиваются с использованием текущего цвета. Такие атрибуты, как способ вывода и тип линии, для эллипса не устанавливаются. Вывод эллипса всегда выполняется с применением сплошного типа линии.

Параметр *control* принимает те же значения и имеет тот же смысл и тип, что и одноименный параметр функции RECTANGLE_W и POLYGON_W.

2.11. Заполнение замкнутых областей.

Замкнутые области могут быть образованы одним графическим примитивом (прямоугольник, эллипс, многоугольник) или сочетанием нескольких, например отрезки прямых.

Как было сказано выше, графические примитивы: прямоугольник, эллипс, многоугольник — могут быть заполнены текущим цветом в процессе их построения, если управляющий заполнением параметр *control* равен $GFILLINTERIOR. При этом цвет заливки и границы примитива совпадают. Помимо этой возможности, любую замкнутую область, а не только отдельный примитив, можно заполнить текущим, установленным функцией SETCOLOR цветом, использовав функцию

result2= FLOODFIL_W (*wx,wy,bcolor*)

wx,wy — параметры типа REAL(8), задающие оконные координаты стартовой точки заполнения;

bcolor — параметр типа INTEGER(2), задающий номер цвета границы.

Функция возвращает отличное от нуля значение при успешном выполнении или 0 в противном случае. Последнее происходит, если заполнение не может быть завершено, например если стартовый пиксель (пиксель, в котором расположена стартовая точка) имеет цвет *bcolor* или если стартовая точка лежит за пределами окна вывода. Возвращаемое функцией FLOODFIL_W значение имеет тип INTEGER(2).

При заполнении области используется текущий цвет и текущая маска заполнения.

Если стартовая точка находится внутри области, то заполняется сама область. Если стартовая точка находится вне области, то заполняется пространство вне области. При задании стартовой точки на границе области заполнение не производится.

Заполнение осуществляется во всех направлениях, прекращаясь при достижении пикселей, окрашенных в цвет *bcolor* — цвет границы области.

Замечание. Заполнение области будет выполнено успешно, если область не имеет разрывов и для отображения ее границ использован сплошной тип линии.

3. Приложения QUICKWIN.

3. 1. Возможности QUICKWIN.

В Microsoft Fortran PowerStation (FPS) исполняемое приложение может быть создано как проект QuickWin — многооконный проект с графикой. В среде MS Developer Studio Fortran PowerStation 4.0 такой проект создается после выполнения цепочки: File — New — Project Workspace — QuickWin Application — задать имя проекта (Name) и папку его размещения (Location) — Create. В программных единицах, которые используют процедуры QuickWin, должна быть ссылка на модуль MSFLIB.MOD (в случае FPS):

USE MSFLIB

Работая с QuickWin, можно:

- скомпилировать консоль-приложение как простое приложение для Windows;
- минимизировать и максимизировать создаваемые в QuickWin окна, в которые выполняется передача данных;
- вызывать любую из приведенных в пункте 2 графическую процедуру;
- загружать и сохранять образы;
- выбирать, копировать и вставлять текст и графику;
- обнаруживать и обрабатывать удары мыши;
- выводить на принтер графические данные;
- программировать меню;
- создавать собственные иконы;
- открывать несколько дочерних окон.

Используя диалоговые окна и возможности QuickWin, можно создавать приложения с полноценным текстовым и графическим интерфейсом.

3. 2. Операции над окнами QUICKWIN.

3.2.1. Виды окон QUICKWIN.

После запуска приложения QuickWin создается *обрамляющее окно* (frame window). С обрамляющим окном в модуле MSFLIB связана целочисленная константа QWIN$FRAMEWINDOW, используемая в некоторых процедурах QuickWin, например в SETWSIZEQQ, для идентификации обрамляющего окна.

В обрамляющем окне могут быть расположены одно или несколько *дочерних окон* (child windows). Любое дочернее окно QuickWin появляется либо после вызова функции SETWINDOWCONFIG, либо после выполнения первой процедуры ввода/вывода (В/В), например подпрограммы CLEARSCREEN, заполняющей окно цветом фона. В приложениях QuickWin может быть до 40 дочерних окон. Причем в это число входят также дочерние окна, подсоединенные к устройствам, которые были закрыты оператором CLOSE. Такие окна недоступны для операции В/В данных.

3.2.2. Создание дочернего окна.

В общем случае дочернее окно создается оператором OPEN со спецификатором FILE=’USER’, например

```
open(100, file='user', title=' Изгиб балки ')
```

Спецификатор TITLE= является необязательным и используется для задания заголовка дочернего окна. Каждое новое дочернее окно подсоединяется к устройству В/В с уникальным номером, задаваемым спецификатором UNIT= (в данном случае можно было использовать и такую запись UNIT=100).

Однако оператор OPEN может в приложении отсутствовать. В этом случае в QuickWin создается одно дочернее окно, подсоединенное к всегда существующим устройствам 0,5 и 6. Создаваемое таким образом окно имеет заголовок Graphic 1.

3.2.3. Активизация дочернего окна.

Дочернее окно считается *активным*, если в него выполняется графический В/В, например процедурами LINETO_W, RECTANGLE_W и т.п.

3.2.4. Закрытие устройства дочернего окна.

Из программы устройство, к которому присоединено дочернее окно, закрывается оператором CLOSE. Дочернее окно можно оставить открытым (видимым) и после выполнения этого оператора, если в нем задать спецификатор STATUS='KEEP'. По умолчанию оператор CLOSE выполняется со спецификатором STATUS='DELETE', который может быть задан и явно, например:

```
close(2)                          ! Закрываем и устройство, и дочернее окно
close(unit=2, status='delete')    ! Закрываем и устройство, и дочернее окно
close(unit=2, status='keep')      ! Закрываем только устройство; дочернее окно
                                  !  открыто
```

3.2.5. Изменение размеров и позиции обрамляющего и дочернего окна.

Помимо параметров в QuickWin можно задать размеры и положение окна. Это выполняется функцией SETWSIZEQQ, имеющей синтаксис:

result4=SETWSIZEQQ(*unit, winfo*)

unit — выражение типа INTEGER(4), задающее устройство, к которому подсоединено окно. По умолчанию окно подсоединено к устройствам 0,5 и 6. Для установки размеров обрамляющего окна в качестве номера устройства следует использовать определенную в модуле MSFLIB именованную константу QWIN$FRAMEWINDOW.

winfo — переменная производного типа *qwinfo*, задающая физические координаты левого верхнего угла окна, его высоту и ширину. Производный тип *qwinfo* определен в модуле MSFLIB.

```
type (qwinfo) winfo;
winfo.type=1;
 или
winfo.type=2;
 или
winfo.type=3;
 или
winfo.type=4;
```

где 1 — это QWIN$MIN — выполняется минимизация окна;

2 — это QWIN$MAX — выполняется максимизация окна;

3 — это QWIN$RESTORE — минимизированное окно возвращается к прежним размерам и позиции;

4 — это QWIN$SET — положение окна и его размеры устанавливаются по значениям других компонентов *qwinfo*;

Функция возвращает значение типа INTEGER(4), равное нулю в случае успеха и отличное от нуля при неудаче.

Замечание. Позиция и размеры обрамляющего окна задаются в пикселях экрана. Позиция и размеры дочернего окна задаются в единицах высоты и ширины неграфической литеры — знакоместа.

3. 3. Изменение системного меню.

По умолчанию приложение QuickWin сопровождается приведенным на рис.3.3. меню.

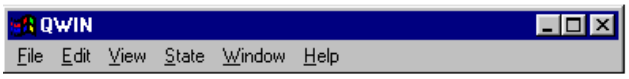


Рис.3.3.Меню QuickWin.

рассмотрим устройство меню. Оно состоит из пунктов; каждый пункт имеет подпункты, которые отображаются после воздействия (мышью или с клавиатуры) на пункт меню. Так пункт меню File имеет подпункты Print, Save, Exit (Ctrl+C). Пункты меню и подпункты имеют номера. Так, пункт File имеет номер 1, пункт Edit — номер 2 и т.д. Так же изменяются и номера подпунктов: подпункт Print имеет номер 1, Save — номер 2, а Exit — номер3. С каждым подпунктом меню связана установленная по умолчанию процедура, например с подпунктом Print связана процедура WINPRINT, запускаемая при выборе этого подпункта и открывающая диалоговое окно работы с принтером.

При инициализации можно создать свое собственное меню, которое в процессе работы программы может быть изменено. Также в процессе работы программы можно изменить и задаваемое по умолчанию меню. Функции, изменяющие меню и его свойства, приведены в табл.3.1. Все функции имеют тип LOGICAL(4).

Таблица 3.1.Функции, изменяющие меню и его свойства.

Функция	*Назначение*
APPENDMENUQQ	Добавляет пункт или подпункт меню
DELETEMENUQQ	Удаляет пункт или подпункт меню
INSERTMENUQQ	Вставляет пункт или подпункт меню
MODIFYMENUFLAGSQQ	Изменяет свойства пунктов или подпунктов меню, применяя одно из приведенных в табл.3.2. действий
MODIFYMENUROUTINEQQ	Изменяет связанную с пунктом меню или его подпунктом подпрограмму
MODIFYMENUSTRINGQQ	Изменяет название пункта или подпункта меню
SETWINDOWMENUQQ	Создает список имеющихся дочерних окон

Таблица 3.2.Изменение свойств пунктов и подпунктов меню.

Действие	*Результат*
$MENUGRAYED	Пункт или подпункт меню неактивен и светло-серого цвета
$MENUDISABLED	Пункт или подпункт меню неактивен, но обычного серого цвета
$MENUENABLED	Пункт или подпункт меню становится активным (доступным)
$MENUSEPARATOR	Вывод разделительной линии между подпунктами меню
$MENUCHECKED	Простановка галочки рядом с пунктом или подпунктом меню
$MENUUNCHECKED	Удаление галочки, стоящей рядом с пунктом или подпунктом меню

При необходимости можно в одной функции задать несколько действий, связав их логической операцией .OR., например:

$MENUCHECKED.OR. $MENUENABLED.

С каждым пунктом пользовательского меню можно связать либо заданную в QuickWin (табл.3.3.), либо пользовательскую подпрограмму, которая будет запускаться при выборе подпункта.

Таблица 3.3.Заданные процедуры QuickWin.

Имя процедуры	*Действие*
WINPRINT	Вывод содержимого окна на принтер
WINSAVE	Запись изображения в файл
WINEXIT	Выход из QuickWin
WINSELGRAPH	Выбор графического изображения из текущего окна
WINCOPY	Копирование выбранных элементов в буфер
WINCLEARPASTE	Очистка буфера
WINSIZETOFIT	Освобождение дочернего окна от передвижных шкал
WINFULLSCREEN	Переключение в режим полного экрана
WININDEX	Вывод содержания имеющейся справочной информации (Help)
WINUSING	Вывод информации о том, как использовать Help
NUL	Отсутствие процедуры

3. 4. Инициализация меню и обрамляющего окна.

Если программу QuickWin снабдить внешней функцией INITIALSETTINGS, то эта функция будет автоматически вызвана при запуске приложения. В этой функции можно задать начальные параметры обрамляющего окна (размеры и положение) и создать свое собственное меню взамен устанавливаемого по умолчанию меню QuickWin (см.рис.3.3.).

Функция INITIALSETTINGS может включать приведенные в табл.3.1. функции, которые изменяют меню QuickWin, и функцию SETWINSIZEQQ, задающую размеры и позицию обрамляющего окна QuickWin. Напомним, что параметры обрамляющего окна задаются в пикселях, а размеры и позиция дочернего окна — в единицах высоты и ширины знакоместа.

Пользовательское меню будет создано, если первой среди всех изменяющих меню функций выполнить функцию APPENDMENUQQ, добавляющую первый пункт меню, например:

```
l4 = appendmenuqq(1, $MENUENABLED,   'Файл'C,  NUL )
```

Если INITIALSETTINGS не создает пользовательское меню, то будет загружено приведенное на рис.3.3. меню. Причем выполняемые в INITIALSETTINGS попытки удалить или вставить пункты (подпункты) меню будут проигнорированы.

Функция INITIALSETTINGS имеет тип LOGICAL(4) и должна вернуть .TRUE., если удалось выполнить начальные установки QuickWin, и .FALSE. — в противном случае.

Если функция INITIALSETTINGS в программе отсутствует, то QuickWin запускает свою функцию инициализации, которая всегда возвращает.TRUE..

3. 5. Вывод стандартного окна сообщений.

При необходимости, обратившись к функции MESSAGEBOXQQ, в приложении можно вывести стандартное окно сообщений. Функция имеет синтаксис

response=MESSAGEBOXQQ(*msg, caption, mtype*)

msg — выводимое сообщение; должно быть СИ-строкой.

caption — текст, выводимый в заголовке окна сообщений; должен быть СИ-строкой.

mtype — именованная константа, определяющая объекты (вид кнопок или икон) или атрибутов окна сообщений. Можно, использовать логическую операцию .OR., комбинировать несколько констант, например сочетание MB$YESNO.OR.MB$ICONQUESTION задает кнопки *Да* и *Нет* и икону с вопросительным знаком. Примеры констант *mtype* приведены в табл.3.4.

Таблица 3.4.Примеры констант mtype функции MESSAGEBOXQQ.

Именованная константа	*Выводимые объекты*
MB$ICONEXCLAMATION	Икона с восклицательным знаком
MB$ICONSTOP	Икона со знаком ×
MB$ICONQUESTION	Икона с вопросительным знаком
MB$YESNO	Кнопки *Да* и *Нет*
MB$YESNOCANCEL	Кнопки *Да, Нет* и *Отмена*

Отметим, что в случае русского Windows имена кнопок выводятся на русском языке (кроме названия кнопки OK).

3. 6. Копирование графики окна QUICKWIN.

В дочернем окне QuickWin можно выделить и скопировать в буфер присутствующее в нем графическое изображение. Для этого используются подпункты SelectGraph и CopyGraph пункта меню *График*. После выбора подпункта SelectGraph следует, используя мышь или Shift и стрелки клавиатуры, выделить график отмеченным прямоугольником.

Далее выбранная часть окна может быть скопирована в буфер (подпункт CopyGraph меню *График* или Ctrl+Ins). Затем содержимое буфера можно вставить, например, в создаваемый вами в среде Word документ или в стандартный редактор рисунков Paint Brash. Вставка производиться после выполнения Shift+Ins.

3. 7. Использование мыши.

3.7.1. Связанные с мышью события.

В QuickWin по умолчанию мышь используется для доступа к меню и управления обрамляющим и дочерними окнами (изменение их положения, размеров…). Все эти управляющие воздействия выполняются за пределами дочернего окна. Внутри дочернего окна мышью можно обеспечить выход из режима “Полный экран”, выделение графики и текста с целью их последующего копирования в буфер и переключение с одного дочернего окна на другое.

Однако QuickWin можно создать подпрограммы, которые фиксируют и соответствующим образом реагируют на те или иные происходящие с мышью события. Перечень таких событий приведен в табл.3.5.

Таблица 3.5.События, которые могут происходить с мышью.

Событие	*Описание*
MOUSE$LBUTTONDOWN	Нажата левая кнопка мыши
MOUSE$LBUTTONUP	Отпущена после нажатия левая кнопка мыши
MOUSE$LBUTTONDBLCLK	Двойной удар левой кнопкой мыши
MOUSE$RBUTTONDOWN	Нажата правая кнопка мыши
MOUSE$RBUTTONUP	Отпущена после нажатия правая кнопка мыши
MOUSE$RBUTTONDBLCLK	Двойной удар правой кнопкой мыши
MOUSE$MOVE	Перемещение мыши

Замечания:

1. Программы могут фиксировать лишь те события, которые произошли, когда мышь находилась в пределах дочернего окна.
2. При двойном ударе кнопкой мыши происходят 4 события: BUTTONDOWN, затем BUTTONUP, далее BUTTONDBLCLK и вновь BUTTONUP. Причем событие BUTTONDBLCLK происходит лишь в том случае, если временная разница между первым и вторым ударами мыши не превышает порогового значения, установленного для мыши в панели управления Windows. В противном случае повторно происходит событие BUTTONDOWN.
3. Перечисленные события задаются в виде именованных, определенных в модуле MSFLIB констант.

4. Программы, обрабатывающие происходящие с мышью события, имеют приоритет над двойным ударом мышью при выходе из режима “Полный экран”. Всегда можно покинуть этот режим, нажав Alt+Enter. В то же время при выборе текста и графики дочернего окна QuickWin эти подпрограммы имеют более низкий приоритет, чем процедуры выбора текста или графики.

3.7.2. Блокирующая функция WAITONMOUSEEVENT.

Блокирующая функция WAITONMOUSEEVENT приостанавливает выполнение программы до тех пор, пока не произойдет заданное связанное с мышью событие. Функция имеет синтаксис:

resul4=WAITONMOUSEEVENT (*mouseevents*, *keystate*, *x*, *y*)

mouseevents — события, задающиеся в виде именованных констант (см. табл.3.5.). Тип *mouseevents* — INTEGER(4).

keystate — этот параметр передает состояние кнопок мыши и клавиш Shift и Ctrl клавиатуры во время события. Параметр является комбинацией приведенных в табл.3.6. именованных констант. Комбинация составляется по средством логической операции .OR.

Таблица 3.6.Возможные константы, образующие параметр keystate.

Константа	*Описание*
MOUSE$KS_LBUTTON	Нажата левая кнопка мыши
MOUSE$KS_RBUTTON	Нажата правая кнопка мыши
MOUSE$KS_SHIFT	Нажата и удерживается во время события клавиша Shift клавиатуры
MOUSE$KS_CONTROL	Нажата и удерживается во время события клавиша Ctrl клавиатуры

x, *y* — *x* и *y* координаты мыши во время события.

Параметр *mouseevents* является входным (имеет вид связи IN), а параметры *keystate*, *x*, *y* — выходные, с видом связи OUT.

Функция возвращает значение типа INTEGER(4), равное в случае успеха именованной константе, описывающей заданное событие, или MOUSE$BADEVENT — константе обозначающей, что заданное событие не поддерживается.

Функция не возвращает значение до тех пор, пока не произойдет заданное событие.

4. Задача.

4. 1. Задание.

Создание интерфейса для определения исходных данных и наглядного представления результатов счета (графически) на примере задачи расчета балки на изгиб. Использовать приведенную выше теорию для реализации поставленной задачи.

В составленной программе должна вычисляться функция прогиба и выводиться на экран в виде рисунка балки в деформированном состоянии с прорисовкой опор и внешней нагрузки. На рисунке балка должна быть изображена так, чтобы максимальное значение прогиба не превышало $W_{max} = 0.1$, которое можно задать (нормировка).

4. 2. Расчетная схема.

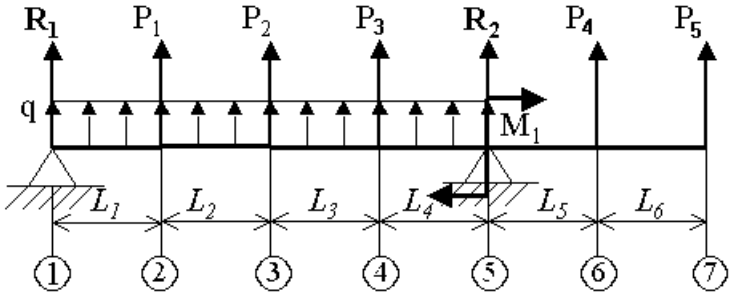


Рис.4.1. Расчетная схема.

4. 3. Исходные данные.

Значения силовых факторов и длины участков задаем числами в системе СИ (см. свой вариант).

Форму прогиба зададим выражением (универсальное уравнение прогиба):

$$W(x) = W_0 + \theta_0 x + \frac{1}{EJ}\left\{ R_1 \frac{x^3}{6} + q\frac{x^4}{24} + P_1\frac{(x-L_1)^3}{6} + P_2\frac{(x-L_1-L_2)^3}{6} + P_3\frac{(x-L_1-L_2-L_3)^3}{6} + \right.$$

$$+ R_2\frac{(x-L_1-L_2-L_3-L_4)^3}{6} - q\frac{(x-L_1-L_2-L_3-L_4)^4}{24} + M\frac{(x-L_1-L_2-L_3-L_4)^2}{2} + \quad (1)$$

$$\left. + P_4\frac{(x-L_1-L_2-L_3-L_4-L_5)^3}{6} \right\}$$

Из граничных условий $W(0) = W_0 = 0, \qquad W(L_1+L_2+L_3+L_4) = 0$ (2)

Находим

$$0 = W(L_1+L_2+L_3+L_4) = \theta_0(L_1+L_2+L_3+L_4) + \frac{1}{EJ}\left\{ R_1\frac{(L_1+L_2+L_3+L_4)^3}{6} + \right.$$

$$\left. + q\frac{(L_1+L_2+L_3+L_4)^4}{24} + P_1\frac{(L_2+L_3+L_4)^3}{6} + P_2\frac{(L_3+L_4)^3}{6} + P_3\frac{(L_4)^3}{6} + P_4\frac{(-L_5)^3}{6} \right\}$$

$$\theta_0 = \frac{1}{EJ(L_1+L_2+L_3+L_4)}\left\{ R_1\frac{(L_1+L_2+L_3+L_4)^3}{6} + q\frac{(L_1+L_2+L_3+L_4)^4}{24} + \right.$$

$$\left. + P_1\frac{(L_2+L_3+L_4)^3}{6} + P_2\frac{(L_3+L_4)^3}{6} + P_3\frac{(L_4)^3}{6} + P_4\frac{(-L_5)^3}{6} \right\} \quad (3)$$

Найдем коэффициент нормировки прогиба:

прогиб (1) представим через безразмерный прогиб

$$W(x) = \frac{B}{(L_1+L_2+L_3+L_4+L_5+L_6)}\overline{W} \qquad \Rightarrow \qquad B = \frac{W(x)\cdot(L_1+L_2+L_3+L_4+L_5+L_6)}{\overline{W}} \quad (4)$$

окончательно коэффициент нормировки B получим из условия максимального прогиба

$$B = \frac{W_{max} \cdot (L_1 + L_2 + L_3 + L_4 + L_5 + L_6)}{\overline{W}_{max}} = \frac{0.1 \cdot (L_1 + L_2 + L_3 + L_4 + L_5 + L_6)}{\overline{W}_{max}}, \quad (5)$$

где безразмерный максимальный прогиб ($\overline{W}_{max}$) ищется путем перебора значений по модулю.

В формулу для прогиба (1) входят неизвестные реакции опор R_1 и R_2, определим их из условий

$$\sum M_{(1)} = 0:$$

$$P_1 L_1 + P_2 (L_1 + L_2) + P_3 (L_1 + L_2 + L_3) + R_2 (L_1 + L_2 + L_3 + L_4) + P_4 (L_1 + L_2 + L_3 + L_4 + L_5) +$$
$$+ P_5 (L_1 + L_2 + L_3 + L_4 + L_5 + L_6) + q \frac{(L_1 + L_2 + L_3 + L_4) \cdot (L_1 + L_2 + L_3 + L_4)}{2} - M = 0$$

$$R_2 = \frac{1}{(L_1 + L_2 + L_3 + L_4)} \left\{ - P_1 L_1 - P_2 (L_1 + L_2) - P_3 (L_1 + L_2 + L_3) - P_4 (L_1 + L_2 + L_3 + L_4 + L_5) - \right.$$
$$\left. - P_5 (L_1 + L_2 + L_3 + L_4 + L_5 + L_6) - q \frac{(L_1 + L_2 + L_3 + L_4) \cdot (L_1 + L_2 + L_3 + L_4)}{2} + M \right\} \quad (6)$$

$$\sum M_{(5)} = 0:$$

$$P_4 L_5 + P_5 (L_5 + L_6) - P_1 (L_2 + L_3 + L_4) - R_1 (L_1 + L_2 + L_3 + L_4) - P_2 (L_3 + L_4) +$$
$$- P_3 L_4 - q \frac{(L_1 + L_2 + L_3 + L_4) \cdot (L_1 + L_2 + L_3 + L_4)}{2} - M = 0$$

$$R_1 = \frac{1}{(L_1 + L_2 + L_3 + L_4)} \left\{ P_4 L_5 + P_5 (L_5 + L_6) - P_1 (L_2 + L_3 + L_4) - P_2 (L_3 + L_4) + \right.$$
$$\left. - P_3 L_4 - q \frac{(L_1 + L_2 + L_3 + L_4) \cdot (L_1 + L_2 + L_3 + L_4)}{2} - M \right\} \quad (7)$$

4. 4. Структура оболочки.

После загрузки программы должно появиться окно программы со строкой меню. В качестве пунктов и подпунктов меню можно предложить:

Таблица 4.1. Вариант меню.

n/n	*Пункт*	*Подпункт*
1	Файл	Новый… Открыть… ——————— Сохранить Сохранить как… ——————— Печать ——————— Exit
2	Геометрия…	Диалоговое окно
3	Усилия…	Диалоговое окно
4	Настройка…	Диалоговое окно
5	Рисунок	Диалоговое окно
6	Вид	Полный экран (Full Screen)
7	График…	SelectGraph CopyGraph ——————— SaveWin
8	Help	Содержание... Как использовать Help...

9	О программе…	Диалоговое окно

4. 5. Структура программы.

4.5.1. Головная программа.

Головная программа используется для запуска проекта. В ней содержатся команды создания обрамляющего окна с именем “Изгиб балки” в шкале, максимизация размеров обрамляющего окна, цикл ожидания события (нажатие правой кнопки мыши).

4.5.2. Логическая функция создания меню.

В проект может быть включена функция создания собственного меню. Функция нигде явно не вызывается, но выполняется автоматически при запуске программы. Если при ее выполнении обнаруживаются ошибки, то загружается стандартное меню (см. рис.3.3.).

Все пункты меню описываются с помощью команды добавления пункта меню. Среди параметров команды можно выделить следующие:

- порядковый номер пункта или подпункта (подпункты имеют номер своего пункта);
- статус пункта;
- текст в виде СИ-строки — название пункта или подпункта;
- имя выполняемой подпрограммы, имеющей только один логический параметр для контроля выполнения. Подпрограммы могут быть стандартными (см. табл.3.3.) или подпрограммами пользователя. В последнем случае их имя должно быть указано в операторе EXTERNAL функции создания меню.

Все действия по настройке параметров задачи, счета и т.д. выполняются через пункты меню.

4.5.3. Обмен данными между программными единицами.

Такой обмен удобно выполнять с помощью модулей. В модуле описываются все глобальные объекты, которым можно задать начальные значения. Эти объекты становятся доступными под их именами после включения в программную единицу сразу после заголовка оператора

USE’имя модуля’.

4.5.4. Изменение исходных данных.

Исходные данные — параметры задачи и оболочки — могут меняться от запуска к запуску. Для сохранения последних введенных данных может использоваться текстовый файл, имеющий строго установленную структуру. Чтение данных из файла может выполняться или при запуске программы, или при выполнении соответствующего подпункта меню (Открыть…), или непосредственно перед изменением или использованием данных. Запись данных в файл может осуществляться или сразу после их редактирования, или по команде подпункта меню (Сохранить или Сохранить как…), или перед закрытием программы.

4.5.6. Редактирование данных.

Редактирование данных может быть осуществлено с помощью диалоговых окон, имеющих соответствующие поля. Диалоговое окно может быть раскрыто при выборе соответствующего пункта меню. При выполнении подпрограммы, установленной для такого пункта, выполняется инициализация диалогового окна, устанавливаются свойства полей и событий. Например, изменение какого-либо данного могут фиксироваться (программно) как в процессе редактирования, так и после закрытия диалогового окна. События могут выполняться при нажатии кнопок путем запуска подпрограмм, имеющих строго фиксированные параметры. Некоторые события устанавливаются по умолчанию.

Подробно как создаются диалоговые окно было рассмотрено ранее в главе 1.

Осталось нерассмотренным только диалоговое окно с рисунком, его можно создать так:

- создаем рисунок балке в среде Word (см.рис.4.1.) и сохраняем его в формате BMP.
- выполняем цепочку: Insert — Resource — Bitmap — Import… — выбираем свой файл BMP — Import
- выполняем цепочку: Insert — Resource — Dialog — вставляем Picture — нажать 2 раза левой кнопкой мыши на рисунок после чего появится окно

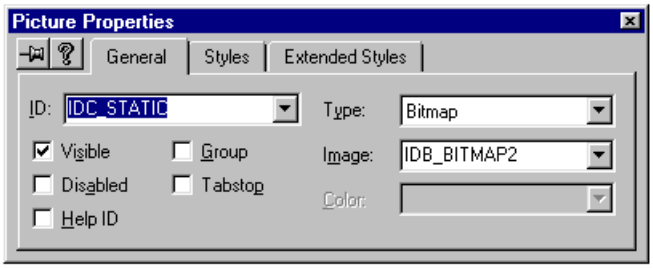


Рис.4.2.

4.5.7. Графическое представление результатов счета.

Данная подпрограмма содержит в себе команды создания графического (дочернего) окна с именем “График” в шкале, максимизация размеров графического окна. Далее для графического вывода используются функции, описанные подробно в главе 2. Затем вызов подпрограмм рисования стрелок, изображающих внешние нагрузки.

4. 6. Программа.

Имя проекта: BEAM

- File — New — Project Workspace — QuickWin Application — Name: BEAM и папку его размещения (Location) — Create.
- Головная программа: кнопка New Source File, правая кнопка мыши — Insert File into Project — BEAM — head1

```
$real:8
PROGRAM Iz_b_1_h
    USE MSFLIB
    implicit none
            type (qwinfo)qw;
            integer i4,ix,iy
    ! раскрытие обрамляющего окна с максимальными размерами:
      OPEN(100, file='user', title=' Изгиб балки ')
    qw.type=2; i4=setwsizeqq(qwin$framewindow,qw);
    i4=setwsizeqq(100,qw)
    ! выход из этога окна при помощи мышки (событие-нажата правая кнопка мыши):
      do while (.TRUE.)
            i4 = waitonmouseevent(MOUSE$RBUTTONDOWN, i4, ix, iy)
            if(.not.i4==MOUSE$BADEVENT)exit
      end do
    end
```

- Перед созданием меню необходимо разработать структуру файла TXT. В файле должны храниться:

а)геометрические параметры:

m_1, m_2, m_3, m_4, m_5, m_6 в формате f6.2

б)числовые значения нагрузок жесткости и толщины балки:

P1,P2,P3,P4,P5,MM1,q,EJ,tol в формате f10.4

в)параметры построения графика:

относительный максимальный прогиб — wm в формате f6.2

координаты порта вывода рисунка в пикселях (левый угол и правый угол) — jpl,ipl,jpr в формате i5

цвета построения cvb,cvs,cvf,cvq,cvm в формате i4

- Записанным данным необходимо задать значения по умолчанию, причем они должны быть доступны из любой подпрограммы. Поэтому эти переменные обязательно описываем в модуле: New Source File, правая кнопка мыши — Insert File into Project — BEAM — mod_b

```
$real:8
Module dano
    ! длины участков нагружения
    real:: m1=1., m2=1., m3=1., m4=1., m5=1., m6=1.
    ! сумма некоторых длин участков нагружения
    real:: m123456, m1234
    ! реакции опор и параметр θ0
    real:: r1, r2, tet0
    ! внешняя нагрузка, жесткость балки и ее толщина
    real:: p1=1., p2=1.,p3=1.,p4=1.,p5=1., mm1=1., q=1., EJ=1., tol=1.
    ! относительный максимальный прогиб
    real:: wm=.1
    ! высоты опор и стрелок при их графической реализации
    real:: lp,lf,lm,lq
    ! нормирующий прогиб коэффициент
    real:: B
    integer:: jpl=100, ipl=100, jpr=400, ipr ,lam,laq
    integer:: cvb=0, cvs=5, cvf=15,cvq=12,cvm=3
    ! описание текстовых переменных
        character(6)::   fmt1='(f6.2)',c1
        character(10)::  fmt2='(f10.4)',c2
        character(5)::   fmt3='(i5)',c3
        character(4)::   fmt4='(i4)',c4
end module dano
```

- Меню Файл: с каждым подпунктом (см. табл.4.1.) связано выполнение подпрограммы.

New_f:
по оператору open(1,file='') открывается стандартное окно создания/открытия файла. В окне выбирается нужная папка, задается имя.txt, подтверждается создание нужного файла. Производится запись (call save_f(checked)) текущих исходных данных.
В момент запуска программы доступны только подпункты: Новый…, Открыть…, Печать, Exit. После выполнения Новый… или Открыть… (определяется файл с логическим номером 1) становятся доступны все остальные подпункты меню.
Open_f:
open(1,file='') — выбирается нужный файл и производиться чтение из него.
Save_f:
запись в файл под номером 1 текущих данных
Exit_pr:
следует запрос на сохранение (если открыт файл №1). Если да, то запись. Затем Exit.

```
$real:8
subroutine new_f(checked)
 use msflib
 logical checked
  checked=.true.
  open(1,file='');
  call save_f(checked)
  ! модификация пунктов меню
  do i=2,5;
       i4=modifymenuflagsqq(i,0,$menuenabled);
  enddo
    i4=modifymenuflagsqq(1,4,4)
    i4=modifymenuflagsqq(1,5,4)
       1 return
```

```
   end

subroutine save_f(checked)
  use dano
  logical checked
   checked=.true.
            rewind(1)
     write(1,'(6f6.2,12x,"m1,m2,m3,m4,m5,m6")')m1,m2,m3,m4,m5,m6
     write(1,'(9f10.4,3x, "P1,P2,P3,P4,P5,MM1,q,EJ,tol")')P1,P2,P3,P4,P5,MM1,q,EJ,tol
     write(1,'( f6.2,24x,"wm"      )')wm
     write(1,'( 3i5, 20x,"jpl,ipl,jpr"   )')jpl,ipl,jpr
     write(1,'( 5i4, 25x,"cvb,cvs,cvf,cvq"      )')cvb,cvs,cvf,cvq,cvm
end

subroutine open_f(checked)
 use msflib
 logical checked
   checked=.true.
   open(1,file='');
   call read_f
   ! модификация пунктов меню
             do i=2,5; i4=modifymenuflagsqq(i,0,$menuenabled);enddo
      i4=modifymenuflagsqq(1,4,4)
      i4=modifymenuflagsqq(1,5,4)
   end

subroutine read_f
  use dano
            rewind(1)
     read(1,*)m1,m2,m3,m4,m5,m6
     read(1,*)P1,P2,P3,P4,P5,MM1,q,EJ,tol
     read(1,*)wm
     read(1,*)jpl,ipl,jpr
     read(1,*)cvb,cvs,cvf,cvq,cvm
end

subroutine exit_pr(checked)
 use msflib
 logical  checked, exist_f
 integer i4
   inquire(1,named=exist_f)
   if(exist_f) then
      i4 = messageboxqq('Сохранить файл?'C,&
         '    BEAM 'C, MB$yesnocancel)
             if(i4==MB$IDYES ) call  save_f(checked)
             if(i4==MB$IDcancel ) return
            endif
   i4 =  SETEXITQQ(QWIN$EXITNOPERSIST)
   stop
end
```

- Меню Геометрия…: диалоговое окно

```
$real:8
$debug
! оформление диалогова окна "Геометрия...":
subroutine geometr(checked)
  use msflib
  use dialogm
  use dano
  include 'resource.fd'
```

```
 type (dialog) dlg
 logical checked, l4
 integer:: i4
  checked=.true.
  l4=dlginit(idd_geom,dlg)
            write(c1,fmt1)m1;      l4=dlgset(dlg,idc_dl_ed1,c1)
            write(c1,fmt1)m2;      l4=dlgset(dlg,idc_dl_ed2,c1)
            write(c1,fmt1)m3;      l4=dlgset(dlg,idc_dl_ed3,c1)
            write(c1,fmt1)m4;      l4=dlgset(dlg,idc_dl_ed4,c1)
            write(c1,fmt1)m5;      l4=dlgset(dlg,idc_dl_ed5,c1)
            write(c1,fmt1)m6;      l4=dlgset(dlg,idc_dl_ed6,c1)
            write(c2,fmt2)EJ;      l4=dlgset(dlg,idc_g_ed,c2)
            write(c2,fmt2)tol;     l4=dlgset(dlg,idc_t_ed,c2)
            write(c2,fmt2)wm;      l4=dlgset(dlg,idc_wm_ed,c2)

            i4=dlgmodal(dlg);       if(i4==idcancel)return

            l4=dlgget(dlg,idc_dl_ed1,c1); read(c1,fmt1)m1;
            l4=dlgget(dlg,idc_dl_ed2,c1); read(c1,fmt1)m2;
            l4=dlgget(dlg,idc_dl_ed3,c1); read(c1,fmt1)m3;
            l4=dlgget(dlg,idc_dl_ed4,c1); read(c1,fmt1)m4;
            l4=dlgget(dlg,idc_dl_ed5,c1); read(c1,fmt1)m5;
            l4=dlgget(dlg,idc_dl_ed6,c1); read(c1,fmt1)m6;
            l4=dlgget(dlg,idc_g_ed,c2);    read(c2,fmt2)EJ;
            l4=dlgget(dlg,idc_t_ed,c2);    read(c2,fmt2)tol;
            l4=dlgget(dlg,idc_wm_ed,c2); read(c2,fmt2)wm;

          call dlgexit(dlg)
          call save_f(checked)
          Call coef
          Call gg
end
```

- Меню Усилия…: диалоговое окно

```
$real:8
$debug
! оформление диалогова окна "Усилия...":
subroutine stren(checked)
 use msflib
 use dialogm
 use dano
 include 'resource.fd'
 type (dialog) dlg
 logical checked, l4
 integer:: i4
  checked=.true.
  l4=dlginit(idd_stren,dlg)
            write(c2,fmt2)p1;      l4=dlgset(dlg,idc_s_ed1,c2)
            write(c2,fmt2)p2;      l4=dlgset(dlg,idc_s_ed2,c2)
            write(c2,fmt2)p3;      l4=dlgset(dlg,idc_s_ed3,c2)
            write(c2,fmt2)p4;      l4=dlgset(dlg,idc_s_ed4,c2)
            write(c2,fmt2)p5;      l4=dlgset(dlg,idc_s_ed5,c2)
            write(c2,fmt2)mm1;     l4=dlgset(dlg,idc_m_ed,c2)
            write(c2,fmt2)q;       l4=dlgset(dlg,idc_q_ed,c2)

            i4=dlgmodal(dlg);       if(i4==idcancel)return

            l4=dlgget(dlg,idc_s_ed1,c2);   read(c2,fmt2)p1;
            l4=dlgget(dlg,idc_s_ed2,c2);   read(c2,fmt2)p2;
            l4=dlgget(dlg,idc_s_ed3,c2);   read(c2,fmt2)p3;
```

```
          l4=dlgget(dlg,idc_s_ed4,c2);   read(c2,fmt2)p4;
          l4=dlgget(dlg,idc_s_ed5,c2);   read(c2,fmt2)p5;
          l4=dlgget(dlg,idc_m_ed,c2);    read(c2,fmt2)mm1;
          l4=dlgget(dlg,idc_q_ed,c2);    read(c2,fmt2)q;

        call dlgexit(dlg)
        call save_f(checked)
        Call coef
        Call gg
end
```

- Меню Настройка…: диалоговое окно

```
$real:8
! оформление диалогова окна "Настройка...":
subroutine tuning(checked)
 use msflib
 use dialogm
 use dano
 include 'resource.fd'
 type (dialog) dlg
 logical checked, l4
 integer:: i4
  checked=.true.
  l4=dlginit(idd_tun,dlg)

          write(c4,fmt4)cvb;    l4=dlgset(dlg,idc_col1_ed,c4)
          write(c4,fmt4)cvs;    l4=dlgset(dlg,idc_col2_ed,c4)
          write(c4,fmt4)cvf;    l4=dlgset(dlg,idc_col3_ed,c4)
          write(c3,fmt3)jpl;    l4=dlgset(dlg,idc_par2_ed,c3)
          write(c3,fmt3)ipl;    l4=dlgset(dlg,idc_par3_ed,c3)
          write(c3,fmt3)jpr;    l4=dlgset(dlg,idc_par4_ed,c3)
          write(c4,fmt4)cvq;    l4=dlgset(dlg,idc_col4_ed,c4)
          write(c4,fmt4)cvm;    l4=dlgset(dlg,idc_col5_ed,c4)

          i4=dlgmodal(dlg);      if(i4==idcancel)return

          l4=dlgget(dlg,idc_col1_ed,c4);        read(c4,fmt4)cvb;
          l4=dlgget(dlg,idc_col2_ed,c4);        read(c4,fmt4)cvs;
          l4=dlgget(dlg,idc_col3_ed,c4);        read(c4,fmt4)cvf;
          l4=dlgget(dlg,idc_par2_ed,c3);        read(c3,fmt3)jpl;
          l4=dlgget(dlg,idc_par3_ed,c3);        read(c3,fmt3)ipl;
          l4=dlgget(dlg,idc_par4_ed,c3);        read(c3,fmt3)jpr;
          l4=dlgget(dlg,idc_col4_ed,c4);        read(c4,fmt4)cvq;
          l4=dlgget(dlg,idc_col5_ed,c4);        read(c4,fmt4)cvm;

        call dlgexit(dlg);
        call save_f(checked)
        Call coef;
        Call gg
end
```

Замечание. Здесь после нажатия на кнопку OK любого диалогового окна происходит пересчет прогиба и его построение заново. Для этого вызываем подпрограмму построения прогиба Call gg.

- Меню Рисунок: диалоговое окно

```
$real:8
! оформление диалогова окна "Рисунок":
subroutine ris(checked)
 use msflib
 use dialogm
```

```
 use dano
 include 'resource.fd'
 type (dialog) dlg
 logical checked, l4
 integer:: i4
   checked=.true.
   l4=dlginit(idd_ris,dlg)
            i4=dlgmodal(dlg)
          call dlgexit(dlg)
end
```

- Меню Вид, График и Help: см. табл.3.3. — заданные процедуры QuickWin.
- Меню О программе…: диалоговое окно.

```
$real:8
! оформление диалогова окна "Изгиб балки !":
subroutine wind(checked)
 use msflib
 use dialogm
 use dano
 include 'resource.fd'
 type (dialog) dlg
 logical checked, l4
 integer:: i4
   checked=.true.
   l4=dlginit(idd_dp,dlg)
            i4=dlgmodal(dlg)
          call dlgexit(dlg)
end
```

- Подпрограмма инициализации меню:

```
! инициализация меню:
LOGICAL(4) FUNCTION InitialSettings()

USE MSFLIB
implicit none
          LOGICAL(4)  l4
          EXTERNAL  new_f, open_f, save_f, exit_pr, geometr, stren, tuning, ris, wind

l4 = appendmenuqq(1, $MENUENABLED,   'Файл'C,            NUL      )
l4 = appendmenuqq(1, $MENUENABLED,   'Новый...'C,        new_f    )
l4 = appendmenuqq(1, $MENUENABLED,   'Открыть...'C,      open_f   )
l4 = appendmenuqq(1, $MENUSEPARATOR, 'sep'C,             NUL      )
l4 = appendmenuqq(1, $MENUGRAYED,    'Сохранить'C,       save_f   )
l4 = appendmenuqq(1, $MENUGRAYED,    'Сохранить как...'C, new_f    )
l4 = appendmenuqq(1, $MENUSEPARATOR, 'sep'C,             NUL      )
l4 = appendmenuqq(1, $MENUENABLED,   'Печать'C,          WINPRINT     )
l4 = appendmenuqq(1, $MENUSEPARATOR, 'sep'C,             NUL      )
l4 = appendmenuqq(1, $MENUENABLED,   'Exit'C,            exit_pr  )

l4 = appendmenuqq(2, $MENUgrayed,    'Геометрия...'C,     geometr   )

l4 = appendmenuqq(3, $MENUgrayed,   'Усилия...'C,         stren  )

l4 = appendmenuqq(4, $MENUENABLED,   'Настройка...'C,         tuning  )
l4 = appendmenuqq(5, $MENUENABLED,   'Рисунок'C,         ris  )

l4 = appendmenuqq(6, $MENUENABLED,    'Вид'C,             NUL     )
l4 = appendmenuqq(6, $MENUENABLED,   'Full Screen'C,  WINFULLSCREEN)

l4 = appendmenuqq(7, $MENUENABLED,    'График...'C,         nul  )
```

```
l4 = appendmenuqq(7, $MENUENABLED,   'SelectGraph'C, WINselectgraphics)
l4 = appendmenuqq(7, $MENUENABLED,   'CopyGraph'C, WINcopy)
l4 = appendmenuqq(7, $MENUSEPARATOR, 'sep'C,            NUL      )
l4 = appendmenuqq(7, $MENUENABLED,   'SaveWin'C, WINsave)


l4 = appendmenuqq(8, $MENUENABLED,   'Help'C,          NUL)
l4 = appendmenuqq(8, $MENUENABLED,   'Содержание...'C, WINDEX)
l4 = appendmenuqq(8, $MENUENABLED,   'Как использовать Help...'C, WINUSING)

l4 = appendmenuqq(9, $MENUENABLED,   'О программе...'C, wind)

          InitialSettings = l4
return
end
```

- Подпрограмма вычислений:

```
! вычисление реакций опор (R1,R2) и начального угла поворота (tet0):
Subroutine coef
  use dano
   real::m12,m123,m12345,m56,m234,m34
   real::f,t,ft
   real(8)::W
                    m123456=m1+m2+m3+m4+m5+m6
                    m1234=m1+m2+m3+m4
                    m12=m1+m2;   m123=m1+m2+m3;      m12345=m1234+m5
                    m56=m5+m6;   m234=m1234-m1;      m34=m3+m4
r1=(p4*m5+p5*m56-mm1-p1*m234-p2*m34-p3*m4-0.5*q*m1234**2)/m1234
r2=(mm1-0.5*q*m1234**2-p1*m1-p2*m12-p3*m123-p4*m12345-p5*m123456)/m1234
tet0=-(r1*m1234**3/6.+q*m1234**4/24.+p1*m234**3/6.+p2*m34**3/6.+&
                    p3*m4**3/6.+p4*(-m5)**3/6.)/(m1234*EJ)

!поиск максимального значения прогиба:
    f=0;B=1
            do t=0.,m123456,.01
                    ft=dabs(W(dfloat(t))); if(ft>f)f=ft
            enddo
! нормировка прогиба:
    B=Wm/f*(m1+m2+m3+m4+m5+m6)
end

! запись функции прогиба (W):
real(8) function W(t)
  use dano
  real(8)::t
                W=tet0*t+(r1*t**3/6.+q*t**4/24.)/EJ
          if(t>=m1)                                  W=W+p1*(t-m1)**3/(EJ*6.)
          if(t>=m1+m2)                    W=W+p2*(t-m1-m2)**3/(EJ*6.)
          if(t>=m1+m2+m3)                 W=W+p3*(t-m1-m2-m3)**3/(EJ*6.)
          if(t>=m1+m2+m3+m4)              W=W+(r2*(t-m1-m2-m3-m4)**3/6.-&
                                                                  q*(t-m1-m2-m3-m4)**4/24.+&
                                                                  mm1*(t-m1-m2-m3-m4)**2/2.)/EJ
    if(t>=m1+m2+m3+m4+m5)  W=W+p4*(t-m1-m2-m3-m4-m5)**3/(EJ*6.)
   W=B*W
end

! запись функции угла поворота (alfa):
real(8) function alfa(t)
use dano
real(8)::t
```

```
            alfa=tet0+(r1*t**2/2.+q*t**3/6.)/EJ
        if(t>=m1)                       alfa=alfa+p1*(t-m1)**2/(EJ*2.)
        if(t>=m1+m2)                alfa=alfa+p2*(t-m1-m2)**2/(EJ*2.)
        if(t>=m1+m2+m3)             alfa=alfa+p3*(t-m1-m2-m3)**2/(EJ*2.)
        if(t>=m1+m2+m3+m4)          alfa=alfa+(r2*(t-m1-m2-m3-m4)**2/2.-&
                                                q*(t-m1-m2-m3-m4)**3/6.+&
                                                mm1*(t-m1-m2-m3-m4)/1.)/EJ
   if(t>=m1+m2+m3+m4+m5)  alfa=alfa+p4*(t-m1-m2-m3-m4-m5)**2/(EJ*2.)
 alfa=b*alfa
 alfa=datan(alfa)
end
```

- Подпрограммы графических построений:

```
$real:8
$debug
! постороение графика прогиба:
subroutine gg
 use msflib
 use dano
        integer,parameter::m=101
        integer(4)::i4
        real(8):: lxp,lyp ! высота в пикселях по x и по y
        real(8):: x,y,al
        real(8):: dx,dy,w,so
! задание переменных производного типа:
        type (qwinfo)qw; type(wxycoord)wxy,pxy(3)
! задание высот окна, опор,стрелок:
                lp=1;  lf=0.05*m123456;lq=0.02*m123456; lm=0.05*m123456
! длины:
     m123456=m1+m2+m3+m4+m5+m6;        m1234=m1+m2+m3+m4
! дочернего окна, где рисуется график (задается его конфигурация):
 close(2)
 ! название этого окна:
        OPEN(2, file='user', title=' График ')
 ! задаем размеры (распахиваем максимально) и позицию этого окна:
        qw.type=2;  i4=setwsizeqq(100,qw)
 ! задаем цвет экрана:
        ii=setbkcolor(cvf)
 ! открываем видовой порт (видовую систему координат):
        CALL SETVIEWPORT (0, 0, 800, 600)
 ! отчистка и заполнение экрана текущим цветом (cvf):
        CALL CLEARSCREEN ($GCLEARSCREEN)
 ! вычислем размеры видео окна:
                dx=1.5*m123456
                dy=2*(wm*m123456+lp)
                lxp=jpl+jpr;      lyp=lxp*dy/dx;  ipr=ipl+lyp
 ! открываем видовой порт (по размерам графика):
        CALL SETVIEWPORT (jpl, ipl, jpr, ipr)
 ! задаем цвет текущий цвет рисования:
        ii=setcolor(cvb)
 ! открываем окно вывода, в котором можем работатьс вещ. координатами:
        ii=setwindow(.true.,-5d-2*m123456,dy/2.,1.1d0*m123456,-dy/2.)
 ! граница окно вывода (обрамляем прямоугольником):
        ii= rectangle_w(2,-5d-2*m123456,dy/2.,1.1d0*m123456,-dy/2.)
 ! задаем текущую позицию графического вывода, т.е. ставим карандаш в точку начала рисования
        CALL MOVETO_W (0d0, W(0d0),wxy )
! рисуем среднюю линию балки:
                        ! шаг рисования
                        h=0.01*m123456
                        do i=1,m
```

```
                    ! вычисляем точки
                        x=i*h; al=alfa(x)
                        y=W(x)
                    ! соединям найденные точки линией
                        ii=lineto_w(x,y)
                    end do
! рисуем верхнюю линию балки:
                    al=alfa(0d0); y=W(0d0)+tol/2.*dcos(al)
                    CALL MOVETO_W (-tol/2.*dsin(al), y,wxy )
                    do i=1,m
                        x=i*h; al=alfa(x);
                        y=W(x)+tol/2.*dcos(al)
                        ii=lineto_w(x-tol/2.*dsin(al),y)
                    end do
! рисуем правый торец балки:
                ii=lineto_w(x+tol/2.*dsin(al),y-2.*tol/2.*dcos(al))
! рисуем нижнюю линию балки:
                    do i=m-1,0,-1
                        x=i*h; al=alfa(x)
                        y=W(x)-tol/2.*dcos(al)
                        ii=lineto_w(x+tol/2.*dsin(al),y)
                    end do
! рисуем левый торец балки:
                     ii=lineto_w(x-tol/2.*dsin(al),y+2.*tol/2.*dcos(al))
! рисуем опоры:
                    ! задаем ширину опоры
                     so=.05*m123456
                    ! неподвижная опора
                        x=0; al=alfa(x); y=W(x)-tol/2.*dcos(al)
                        x=x+tol/2.*dsin(al)
                            pxy.wx=(/x,x+so/2,  x-so/2/)
                            pxy.wy=(/y,y-.85*so,y-.85*so/)
                        ii=polygon_w(2,pxy,3)
                    ! "земля" под опорой
                    call moveto_w(x-0.75*so,y-0.85*so,wxy);      ii=lineto_w(x+0.75*so,y-0.85*so)
                    call moveto_w(x-0.5*so,y-0.85*so,wxy);       ii=lineto_w(x-0.75*so,y-1.05*so)
                    call moveto_w(x-0.25*so,y-0.85*so,wxy);      ii=lineto_w(x-0.5*so,y-1.05*so)
                    call moveto_w(x,y-0.85*so,wxy);              ii=lineto_w(x-0.25*so,y-1.05*so)
                    call moveto_w(x+0.25*so,y-0.85*so,wxy);      ii=lineto_w(x,y-1.05*so)
                    call moveto_w(x+0.5*so,y-0.85*so,wxy);       ii=lineto_w(x+0.25*so,y-1.05*so)

                    ! подвижная опора
                        x=m1234; al=alfa(x); y=W(x)-tol/2.*dcos(al)
                        x=x+tol/2.*dsin(al)
                            ii=ellipse_w(2,x-0.1*so,y,x+0.1*so,y-0.2*so)
                            ii=ellipse_w(2,x-0.1*so,y-0.65*so,x+0.1*so,y-0.85*so)
                        CALL MOVETO_W (x, y-0.2*so,wxy )
                      ii=lineto_w(x,y-0.65*so)
                    ! "земля" под опорой
                    call moveto_w(x-0.75*so,y-0.85*so,wxy);      ii=lineto_w(x+0.75*so,y-0.85*so)
                    call moveto_w(x-0.5*so,y-0.85*so,wxy);       ii=lineto_w(x-0.75*so,y-1.05*so)
                    call moveto_w(x-0.25*so,y-0.85*so,wxy);      ii=lineto_w(x-0.5*so,y-1.05*so)
                    call moveto_w(x,y-0.85*so,wxy);              ii=lineto_w(x-0.25*so,y-1.05*so)
                    call moveto_w(x+0.25*so,y-0.85*so,wxy);      ii=lineto_w(x,y-1.05*so)
                    call moveto_w(x+0.5*so,y-0.85*so,wxy);       ii=lineto_w(x+0.25*so,y-1.05*so)
! рисуем стрелки:
! распределенная нагрузка
                        do x=0.,m1234,0.1*m1234;if(q/=0)call strela_q(x);enddo
                    ! момент
```

```
                    if(mm1/=0) call  mom()
                ! сосредоточенные силы:
                    if(p1/=0) call strela_f(m1,p1);
                    if(p2/=0) call strela_f(m1+m2,p2)
                    if(p3/=0) call strela_f(m1+m2+m3,p3);
                    if(p4/=0) call strela_f(m1234+m5,p4)
                    if(p5/=0) call strela_f(m1234+m5+m6,p5)
end
```

Подпрограммы рисования стрелок (учтено направление сил и моментов, относительно исходной расчетной схемы, где приняты положительные направления):

```
$debug
$real:8
subroutine mom()
use msflib
use dano
        real(8)::x
        real(8)::al
            type(wxycoord)trxy(3)
            ii=setcolor(cvm)
            lam=1;if(mm1<0)lam=-1
          x=m1234;  al=alfa(x); y=W(x)+tol/2.*dcos(al)
             x=x-tol/2.*dsin(al)
             trxy.wx=(/x+lam*(2*lm/3.-0.05*lm),x+lam*(lm-0.05*lm),  x+lam*(2*lm/3.-0.05*lm)/)
             trxy.wy=(/y+lm,y+0.85*lm,y+0.7*lm/)
             ii=polygon_w(3,trxy,3)
             ii=rectangle_w(3,x-lam*5d-2*lm,y+9d-1*lm,x+lam*(2*lm/3.-5d-2*lm),y+8d-1*lm)
                ii=rectangle_w(3,x-5d-2*lm,y+8d-1*lm,x+5d-2*lm,y)
end

subroutine strela_f(x1,p)
 use msflib
 use dano
            real(8)::al,x,w
         type(wxycoord)wxy(3)
                    lap=1;if(p<0)lap=-1
        ii=setcolor(cvs);x=x1
         al=alfa(x); y=W(x)+tol/2.*dcos(al)
                    x=x-tol/2.*dsin(al)
                    wxy.wx=(/x-0.2*lf, x+0.2*lf, x/)
                    wxy.wy=(/y+lap*2*lf/3., y+lap*2*lf/3., y+lap*lf/)
                    ii=polygon_w(3,wxy,3)
                    ii=rectangle_w(3,x-5d-2*lf,y+lap*2*lf/3.,x+5d-2*lf,y)
end

subroutine strela_q(x1)
 use msflib
 use dano
            real(8)::al,x
         type(wxycoord)wxy(3)
        ii=setcolor(cvq);x=x1
                    laq=1;if(q<0)laq=-1
            al=alfa(x); y=W(x)+tol/2.*dcos(al)
                    x=x-tol/2.*dsin(al)
                    wxy.wx=(/x-0.2*lq, x+0.2*lq, x/)
                    wxy.wy=(/y+laq*2*lq/3.,y+laq*2*lq/3.,y+laq*lq/)
                    ii=polygon_w(3,wxy,3)
                    ii=rectangle_w(3,x-5d-2*lq,y+laq*2*lq/3.,x+5d-2*lq,y)
end
```

4. 7. Графический результат счета.

Здесь

$L_1 = L_2 = L_3 = L_4 = L_5 = L_6 = 1м$

$P_1 = P_2 = P_3 = P_4 = P_5 = 1H$

$h = 0.2м - толщина \quad балки$

$w_{\max} = 0.1$

$EJ = 1Hм^2$

$q = 1H/м$

$M = 1Hм$

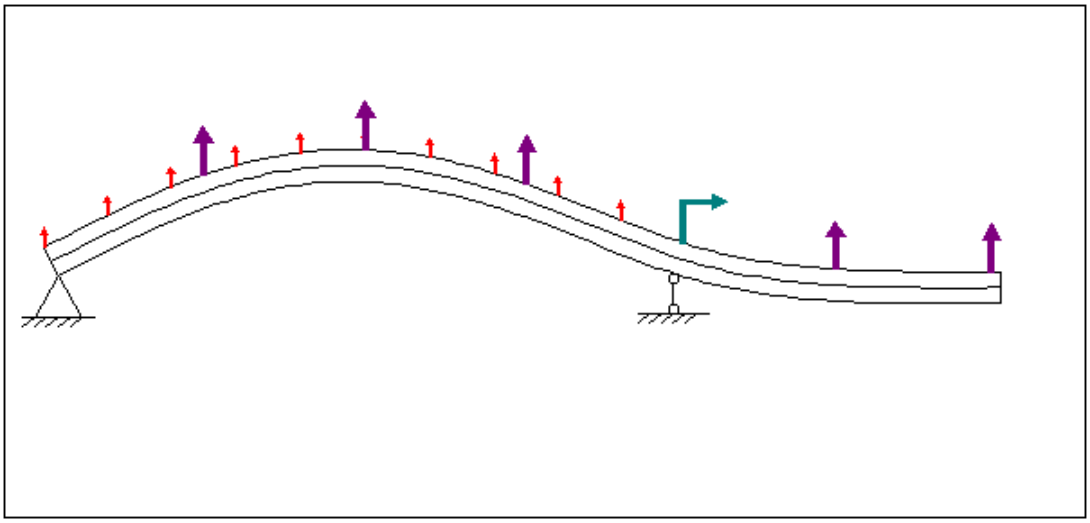


Рис.4.3.Прогиб балки.