Программы и программное обеспечение

Программа - это данные, предназначенные для управления конкретными компонентами системы обработки информации (СОИ) в целях реализации определенного алгоритма.

Обратить внимание: программа - это данные.

Один из *основных принципов машины фон Неймана* - то, что и программы, и данные хранятся в одной и той же памяти. Сохраняемая в памяти программа представляет собой некоторые коды, которые могут рассматриваться как данные.

Возможно, *с точки зрения программиста программа - активный компонент, она выполняет некоторые действия.*

Но *с точки зрения процессора команды программы - это данные, которые процессор читает и интерпретирует.*

С другой стороны *программа - это данные с точки зрения обслуживающих программ*, например, с точки зрения компилятора, который на входе получает одни данные - программу на языке высокого уровня (ЯВУ), а на выходе выдает другие данные - программу в машинных кодах.

Программное обеспечение (ПО) - совокупность программ СОИ и программных документов, необходимых для их эксплуатации

Необходимые свойства ПО:

1 Необходимость **документирования.** По определению программы становятся ПО только при наличии документации. По Бруксу ошибкой в ПО является ситуация, когда программное изделие функционирует не в соответствии со своим описанием, следовательно, ошибка в документации также является ошибкой в программном изделии.

2 **Эффективность**. ПО, рассчитанное на многократное использование пишется и отлаживается один раз, а выполняется многократно.

3 **Надежность**. В том числе:

Тестирование программы при всех допустимых спецификациях входных данных

Защита от неправильных действий пользователя

Защита от взлома - пользователи должны иметь возможность взаимодействия с ПО только через легальные интерфейсы.

"Ошибки в системе возможны из-за сбоев аппаратуры, ошибок ПО, неправильных действий пользователя. Первые - неизбежны, вторые - вероятны, третьи - гарантированы".

4 **Возможность сопровождения**. Возможные цели сопровождения - адаптация ПО к конкретным условиям применения, устранение ошибок, модификация.

Этапы подготовки программы

При разработке программ, а тем более - сложных, используется **принцип модульности:**

разбиения сложной программы на составные части, каждая из которых может подготавливаться отдельно.

Модульность является основным инструментом структурирования программного изделия, облегчающим его разработку, отладку и сопровождение.

Программный модуль - программа или функционально завершенный фрагмент программы, предназначенный для хранения, трансляции, объединения с другими

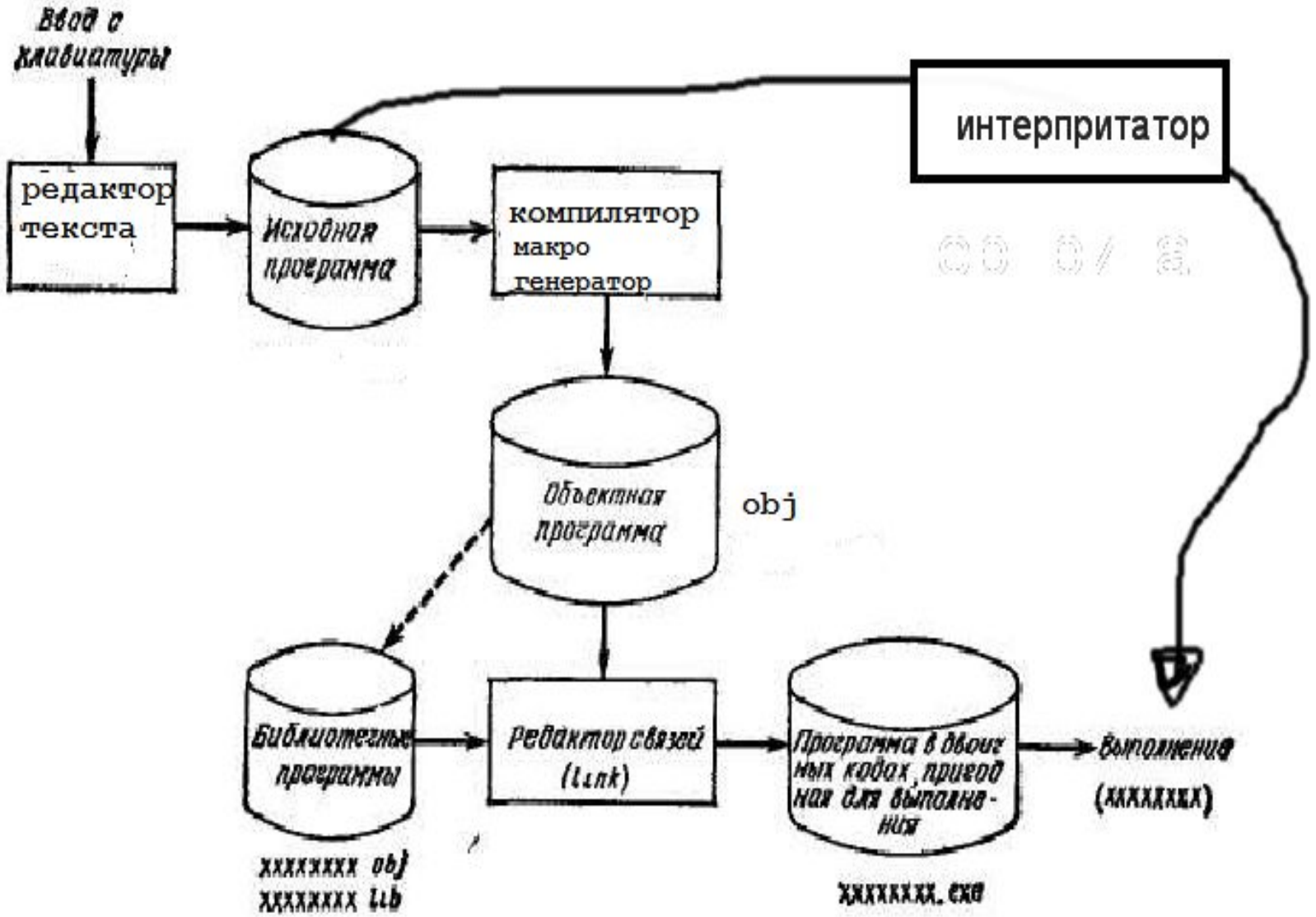
При выборе модульной структуры должны учитываться следующие основные соображения:

Функциональность - модуль должен выполнять законченную функцию

Несвязность - модуль должен иметь минимум связей с другими модулями, связь через глобальные переменные и области памяти нежелательна

Специфицируемость - входные и выходные параметры модуля должны четко формулироваться

Этапы «создания» программы



Программа пишется в виде *исходного модуля*, на рисунке - файл Исходная Программа.

Исходный модуль - программный модуль на исходном языке, обрабатываемый транслятором и представляемый для него как целое, достаточное для проведения трансляции.

Первым (не обязательным) этапом подготовки программы является обработка ее ***Макропроцессором*** (или Препроцессором).

Макропроцессор обрабатывает текст программы и на выходе его получается новая редакция текста. В большинстве систем программирования **Макропроцессор** совмещен с транслятором, и для программиста его работа и промежуточный ИМ' "не видны".

Следует иметь в виду, что **Макропроцессор** выполняет обработку текста - он "не понимает" операторов языка программирования.

Так, если **Макропроцессор** заменил в программе некоторый текст A на текст B, то транслятор уже видит только текст B, и не знает, был этот текст написан программистом "своей рукой" или подставлен Макропроцессором.

Следующим этапом является трансляция.

Трансляция ***- преобразование программы, представленной на одном языке программирования, в программу на другом языке программирования, в определенном смысле (логическом, функциональном) равносильную первой.***

Как правило, выходным языком транслятора является машинный язык целевой вычислительной системы. (Целевая ВС - та ВС, на которой программа будет выполняться.)

Машинный язык - язык программирования, предназначенный для представления программы в форме, позволяющей выполнять ее непосредственно техническими средствами обработки информации.

Трансляторы - общее название для программ, осуществляющих трансляцию.

Они подразделяются на **Ассемблеры** и **Компиляторы** - в зависимости от исходного языка программы, которую они обрабатывают. Ассемблеры работают с Автокодами или языками Ассемблера, Компиляторы - с языками высокого уровня.

Автокод* - *символьный язык программирования, предложения которого по своей структуре в основном подобны командам и обрабатываемым данным конкретного машинного языка.

Язык Ассемблера* - *язык программирования, который представляет собой символьную форму машинного языка с рядом возможностей, характерных для языка высокого уровня (обычно включает в себя макросредства).

Язык высокого уровня - язык программирования, понятия и структура которого удобны для восприятия человеком.

Объектный модуль - программный модуль, получаемый в результате трансляции исходного модуля.

Поскольку результатом трансляции является модуль на языке, близком к машинному, в нем уже не остается признаков того, на каком исходном языке был написан программный модуль

Это создает принципиальную возможность создавать программы из модулей, написанных на разных языках.

Специфика исходного языка, однако, может сказываться на физическом представлении базовых типов данных, способах обращения к процедурам/функциям и т.п. Для совместимости разно языковых модулей должны выдерживаться

Большая часть объектного модуля - команды и данные машинного языка именно в той форме, в какой они будут существовать во время выполнения программы.

Однако, программа в общем случае состоит из многих модулей. Поскольку транслятор обрабатывает только один конкретный модуль, он не может должным образом обработать те части этого модуля, в которых запрограммированы обращения к данным или процедурам, определенным в другом модуле.

Такие обращения называются **внешними ссылками**. Те места в объектном модуле, где содержатся внешние ссылки, транслируются в некоторую промежуточную форму, подлежащую дальнейшей обработке.

Говорят, что **объектный модуль представляет собой программу на машинном языке с неразрешенными внешними ссылками.**

Разрешение внешних ссылок выполняется на следующем этапе подготовки, который обеспечивается **Редактором Связей** (Компоновщиком).

Редактор Связей соединяет вместе все объектные модули, входящие в программу.

Поскольку Редактор Связей "видит" уже все компоненты программы, он имеет возможность обработать те места в объектных модулях, которые содержат внешние ссылки. Результатом работы Редактора Связей является загрузочный модуль.

Загрузочный модуль - программный модуль, представленный в форме, пригодной для загрузки в оперативную память для выполнения.

Загрузочный модуль сохраняется в виде файла на внешней памяти. Для выполнения программа должна быть перенесена (загружена) в оперативную память.

Иногда при этом требуется некоторая дополнительная обработка (например, настройка адресов в программе на ту область оперативной памяти, в которую программа загрузилась). Эта функция выполняется **Загрузчиком**, который обычно входит в состав операционной системы.

Возможен также вариант, в котором редактирование связей выполняется при каждом запуске программы на выполнение и совмещается с загрузкой. Это делает **Связывающий Загрузчик**. Вариант связывания при запуске более расходный, т.к. затраты на связывание тиражируются при каждом запуске. Но он обеспечивает:

- большую гибкость в сопровождении, так как позволяет менять отдельные объектные модули программы, не меняя остальных модулей;
- экономию внешней памяти, т.к. объектные модули, используемые во многих программах не копируются в каждый загрузочный модуль, а хранятся в одном экземпляре.

Вариант интерпретации подразумевает прямое исполнение исходного модуля.

***Интерпретация - реализация смысла некоторого синтаксически законченного текста, представленного на конкретном языке*.**

Интерпретатор читает из исходного модуля очередное предложение программы, переводит его в машинный язык и выполняет.

Все затраты на подготовку тиражируются при каждом выполнении, следовательно, интерпретируемая программа принципиально менее эффективна, чем транслируемая.

Однако, интерпретация обеспечивает удобство разработки, гибкость в сопровождении и переносимость.

Этапы «создания» программы

Не обязательно подготовка программы должна вестись на той же вычислительной системе и в той же операционной среде, в которых программа будет выполняться. Системы, обеспечивающие подготовку программ в среде, отличной от целевой называются **кросс-системами**.

В кросс-системе может выполняться вся подготовка или ее отдельные этапы:

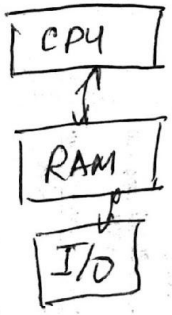
Макро обработка и трансляция

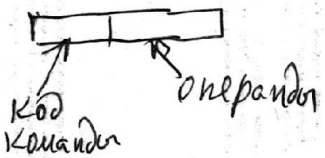
Редактирование связей

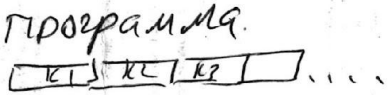
Отладка

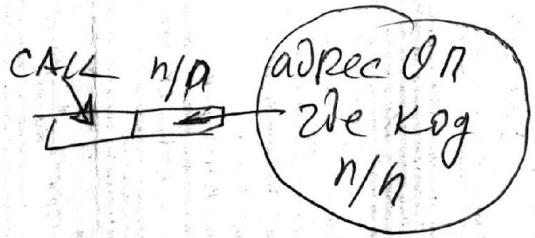
Типовое применение кросс-систем - для тех случаев, когда целевая вычислительная среда просто не имеет ресурсов, необходимых для подготовки программ, например, встроенные системы.

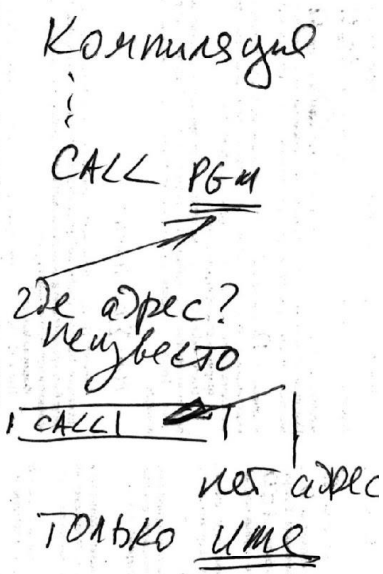
Программные средства, обеспечивающие отладку программы на целевой системе можно также рассматривать как частный случай **кросс-системы**.

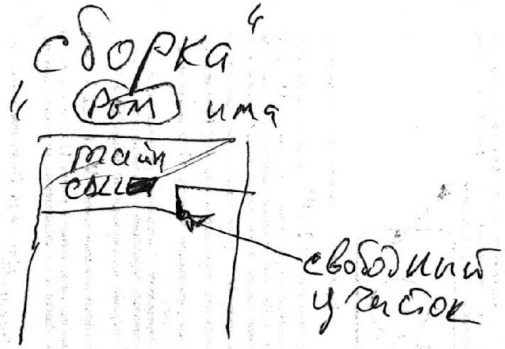

CPU
RAM
I/O

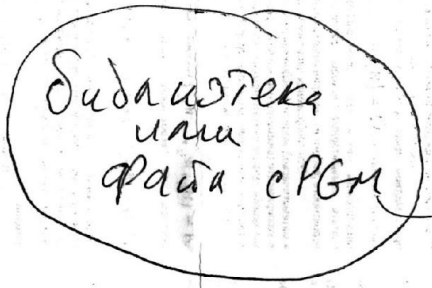

код команды
операнды

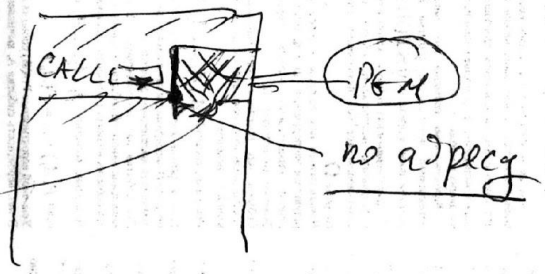

программа
K1
K2


CALL n/p
адрес ОП где код п/п


Компиляция
CALL PGM
где адрес? неизвестно
CALL
нет адрес
только имя


сборка
PGM
имя
свободный участок


библиотека или файл cPGM


CALL
PGM
по адресу

Процесс получения рабочей программы

Кратко рассмотрим процесс получения загрузочного модуля (рабочей программы) по технологии **"статическая сборка"** для системы MINGW gfortran**.**

1 шаг. Создают файл ***исходной программы*** *(SOURCE файл).* Обычно для этого применяют программу, называемую *текстовым редактором,* например **notepad++ имя_файла.тип**

Для удобства работы, имя типа файла выбирают по требованию системы программирования.

Для фортрана выбираются типы - f77, for, f95 (в зависимости от версии).

2 шаг. Транслируют исходную программу (файл), используя *программу - **компилятор** языка программирования фортран.*

Компилятор языка программирования "читает" исходную программу из SOURCE файла. Переводит текст программы в набор машинных команд, которые записываются в выходной файл.

Полученный таким образом файл представляет собой *файл **объектной программы** (OBJ файл).*

Протокол процесса перевода команд исходной программы компилятор может выводить на консоль и/или помещает в специальный "протокольный" файл, обычно называемый "листингом" (файл **«листинг»** - LST файл).

Многие компиляторы позволяют частично "переработать" исходный текст программы перед началом трансляции (перевода).

Программист имеет возможность использовать

3 шаг. **Связывают** (или преобразовывают внутренний формат объектного файла) файл объектной программы с библиотечными программами и другими программами с помощью ***редактора связей,*** и таким образом формируют файл ***загрузочного*** *модуля (EXE файл).*

Этот файл содержит "полный" набор машинных команд в такой форме, при которой сразу возможна их загрузка в оперативную память и выполнение. Обычно протокол "связывания" объектных модулей может быть помещен в файл протокола, или в файл "карты памяти" (MAP - файл).

4 шаг. Выполняют загрузочный модуль.

Вызывают системный загрузчик операционной системы, который копирует содержимое исполняемого файла в оперативную память ЭВМ и настраивает центральный процессор на выполнение машинных команд, содержащихся в загрузочном модуле.

В зависимости от принятой технологии, указанные выше шаги, выполняются в **"консольном режиме"** или при помощи специализированных программ - **интегрированных сред разработки (IDE).**

Процесс получения рабочей программы в консольном режиме

Вызов консоли операционной системы можно осуществить, например, так - "ПУСК" -" ВЫПОЛНИТЬ" - "CMD" или WIN-R и cmd

Далее будем считать, что в операционной системе установлено приложение IDE Code Blocks с MINGW

http://www.codeblocks.org/downloads/binaries

в стандартную папку C:\Program Files\CodeBlocks\.

Напомним, что вызов "внешней" программы выполняется в общем случае так:

путь_к_папке\ИМЯ_ПРОГРАММЫ параметры.... опции....

Если не указывается <**путь_к_папке**> - предполагается текущая папка или запись пути к папке в переменной «окружения ОС» **path**

Например, если программа fp.exe расположена в d:\test\, то её вызов:

d:\test\fp параметры (если необходимо)

Процесс получения рабочей программы в консольном режиме

Для удобства работы <путь_к_папке> для часто вызываемым программам можно не указывать. Для этого он фиксируется средствами операционной системы (команда **set patch**) или составляется простой процедурный файл, в котором указывается путь и параметры программы компилятора, сборщика и т.п..

Пример простого **процедурного файла fll.bat:**

```
echo on
"C:\Program Files\CodeBlocks\MinGW\bin\gfortran"
-Wall -static -s -o %1.exe %1.o -L. -l%myLIB%
```

Пример "вызова" процедуры: **fll fpgm**

Значение переменной %myLIB% (значение таких переменных задаётся командой set myLIB=значение) подставляется в текстовую строку вместо имени %myLIB%. Вместо имени %1 подставляется имя первого параметра командной строки (в примере это fpgm).

Полученная строка передается консоли на выполнение. Часто такие простые процедуры программист помещает в папку с исходными файлами программы.

Далее рассмотрим основные процедурные файлы для системы программирования MingGW gfortran.

Создание и редактирование исходного модуля.

Редактор текста (или текстовый редактор) выбирается исходя из удобства работы. "Классический" текстовый редактор notepad выбирать не рекомендуется, используйте notepad++

Текстовый редактор, интегрированный в файловый менеджер FAR может быть рекомендован для подготовки исходных файлов.

Вызов менеджера FAR производится аналогично "консоли". В FAR для выполнения:

редактирования существующего файла - **«подсветить» и F4,**

создание НОВОГО файла - **Shift - F4.**

Процесс получения рабочей программы в консольном режиме

Компиляция программы – процедура fc.bat

Вызов:

fc имя_файла_исходного_модуля_тип_которого_f95

тип не указывается

При успешной компиляции - создание файла объектного модуля имя_файла_исходного_модуля_тип_которого_**o**

Для примера, ниже дается краткий текст процедуры.

Процедура (упрощенный вариант):

```
"C:\Program Files\CodeBlocks\MinGW\bin\gfortran" -Ofast
                                    -Waliasing –Wall -static -s -c %1.f95 %2
```

Пример вызова для компиляции файла pgm.f95 из текущей папки (процедура находится так же в текущей папке):

fc pgm

Процесс получения рабочей программы в консольном режиме

В текст процедуры можно поместить дополнительные команды CMD. Например:

```
@echo off
if "%1"=="" goto err
@echo компиляция фортран программы из файла %1.f95
"C:\Program Files\CodeBlocks\MinGW\bin\gfortran" -Ofast -Waliasing -Wall
                                                    -static -s -c %1.f95 %2
if errorlevel 1 goto err1
@echo компиляция успешна, создан файл объектного модуля %1.o
goto end
:err
@echo Внимание!!! Не указан файл с исходным кодом программы %1.f95 .....
goto end
:err1
@echo Внимание!!! Обнаружены ошибки компиляции программы %1.f95 .....
goto end
:end
```

Изменяя параметры вызова компилятора, можно выполнить самые разнообразные действия по его настройке. В частности, получить список всех возможных "опций" - ключ (параметр) -V.

Пример процедуры получения "ассемблерной" программы см. fca.bat, получение протокола компиляции - fc1.bat.

Процесс получения рабочей программы в консольном режиме

"Сборка" программы - fl.bat

Вызов:

fl имя_файла_объектного_модуля_тип_которого_f95

(тип не указывается)

Примечание - в этой процедуре необходимо указывать файл, содержащий "главную" программу.
Допускается в один исходный файл помещать "главную" программу и подпрограммы.

В случае когда программный проект содержит много процедур, размещенных в разных исходных файлах, имена всех файлов объектных модулей необходимо указывать следующими (вторым, третьим и т.д.) параметром команды вызова (в процедуре предусмотрен только второй параметр).

Если файлов объектных модулей много, то они могут быть объедены в "библиотеку объектных модулей".

Процесс получения рабочей программы в консольном режиме

Работа с пользовательской библиотекой объектных модулей.

Задать имя библиотеки - setMYlib.bat

Вызов: setMYlib <имя_биб>

До перезагрузки консоли для всех ниже описанных процедур устанавливается имя библиотеки **<имя_биб>.** Согласно требованиям системы MINGW, это имя соответствует библиотечному файлу lib<имя_биб>.a, который будет создан/изменён в текущей папке.

Добавить/заменить объектный файл в библиотеке - flib.bat

Вызов: flib <имя_объектного_модуля>

Примечание: процедура выводит список файлов, которые находятся в библиотеке. Закомментируйте последнюю строчку в процедурном файле, если Вам это ненужно.

Собрать" программу используя библиотеку - fll.bat

Вызов: flib <имя_объектного_модуля_"главной"_программы>

Подробнее смотрите в описании лабораторной работы.

*****