Модули. Структура модулей

- **Модульное программирование** – это метод разработки программ по частям.
- **Модуль** – это автономно компилируемая программная единица, включающая в себя различные компоненты раздела описаний и некоторые исполняемые операторы.

Модуль имеет следующую структуру:

```
UNIT < имя >;
    INTERFACE
        < интерфейсная часть >
    IMPLEMENTATION
        < исполняемая часть >
    BEGIN
        < инициирующая часть >
END.
```

Заголовок модуля и связь модулей друг с другом

- Заголовок модуля состоит из зарезервированного слова **UNIT** и следующего за ним **имени модуля**.

- Например: **Unit GLOBAL;**

Запомните!

- **Имя модуля** должно совпадать с именем дискового **файла**, в который помещается исходный текст модуля.

- Например: Модуль **GLOBAL** должен быть сохранен под именем **GLOBAL.PAS**

- Имя модуля служит для связи с другими модулями и программами. Эта связь устанавливается специальным предложением:

  USES <список модулей >;

- Например: **USES Crt, Graph, Global;**

Интерфейсная часть

- Интерфейсная часть открывается зарезервированным словом **INTERFACE**.
- **В этой части модуля содержится** объявление всех глобальных объектов модуля (**типов, констант, переменных и подпрограмм**), которые должны стать доступными основной программе и другим модулям.

- При объявлении глобальных подпрограмм в интерфейсной части указывается только их **заголовок.**

Пример

```
Unit Global;
Interface
uses crt;
const n=100;
type  Vector=array [1..n] of integer;
procedure wwod (Razm: integer; Name:
   char; var s: Vector);
procedure wywod (Razm: integer; Name:
   char; h: Vector);
...
```

- Если теперь в новой программе написать предложение **Uses Global**, то в программе станут доступными тип **Vector** и процедуры **wwod** и **wywod.**

Исполняемая часть

- Начинается зарезервированным словом **IMPLEMENTATION** и содержит описания подпрограмм, объявленных в интерфейсной части.
- В ней могут быть объявлены локальные для модуля объекты.

Пример (продолжение)

```
Implementation
procedure wwod;
   var i:integer;
Begin
clrscr;
Writeln ( 'Введите элементы массива ‘ );
for i := 1 to Razm do
begin
   write ( Name, ‘ [ ‘ , i , ‘ ]=‘ );
   readln ( s[i] );
End;
end;
```

```
procedure wywod;
var i:integer;
Begin
clrscr;
for i:=1 to Razm do
Writeln (Name, ‘ [ ‘ , i , ‘ ]= ', h [ i ] );
readln;
end;
...
```

Инициирующая часть

- Инициирующая часть завершает модуль.
- Она может отсутствовать.
- Но обязательно в конце модуля должно быть написано слово **END** и поставлена точка.

- Если инициирующая часть есть в модуле, то начинается она с зарезервированного слова **BEGIN**.
- После него размещаются исполняемые операторы, содержащие некоторый фрагмент программы.
- Эти операторы исполняются до передачи управления основной программе и обычно используются для подготовки к ее работе.

- Не рекомендуется делать инициирующую часть пустой, лучше ее опустить

Компиляция модулей

1. Сохранить программу в файле под именем модуля
2. Выбрать в меню **Компайл** пункт **Куда** и изменить его на значение **Disk**
3. Откомпилировать ее в одном из трех режимов **COMPILE (Alt+F9)**, **MAKE (F9)** или **BUILD** (в меню **Compile**).

Пример 2

- Создать модуль, содержащий подпрограммы для вычисления математических функций – tg(x), ctg(x), a^x
- Для создания используем математические формулы:

tg(x)=sin(x)/cos(x)

ctg(x)=cos(x)/sin(x)

$$a^x = e^{x \cdot \ln a}$$

```
Unit func1;
Interface
    function tg (x: real) : real;
    function ctg (x: real) : real;
    function ax (a: real; x: real) : real;
Implementation
    function tg;
    begin
    if cos(x)<>0 then tg:=sin(x)/cos(x)
    else writeln ('Значение tg не определено!');
    end;
```

```
    function ctg;
    begin
    if sin(x)<>0 then ctg:=cos(x)/sin(x) else
    writeln (‘Значение ctg не определено!’);
    end;
    function ax;
    begin
    if a>0 then ax:=exp (x*ln(a)) else
if a=0 then ax:=0 else writeln(‘a - отрицательно’);
    end;
end.
```

Использование созданного модуля

- Даны действительные x и y.Вычислить значение выражения:

$$F = \frac{tg(x+y) - ctg(x-2y)}{2^x + 4^y + x^y + y^x}$$

```
Program test;
Uses crt, func1;
Var x, y, f:real;

Begin
Clrscr;
Writeln (‘x=’);
Readln(x);
Writeln (‘y=’);
Readln(y);
```

```
F:= (tg(x+y) – ctg(x-2*y)) / (ax(2, x) + ax(4, y) +
ax(x, y)+ ax(y, x)) ;
Writeln('F=', f:15:3);
Readln
end.
```

Библиотеки подпрограмм

- Модули могут использоваться для организации библиотеки подпрограмм.
- Часто возникает ситуация, когда один и тот же алгоритм используется при решении самых разных задач.
- В таких случаях желательно использовать уже готовую подпрограмму как часть любой другой программы.

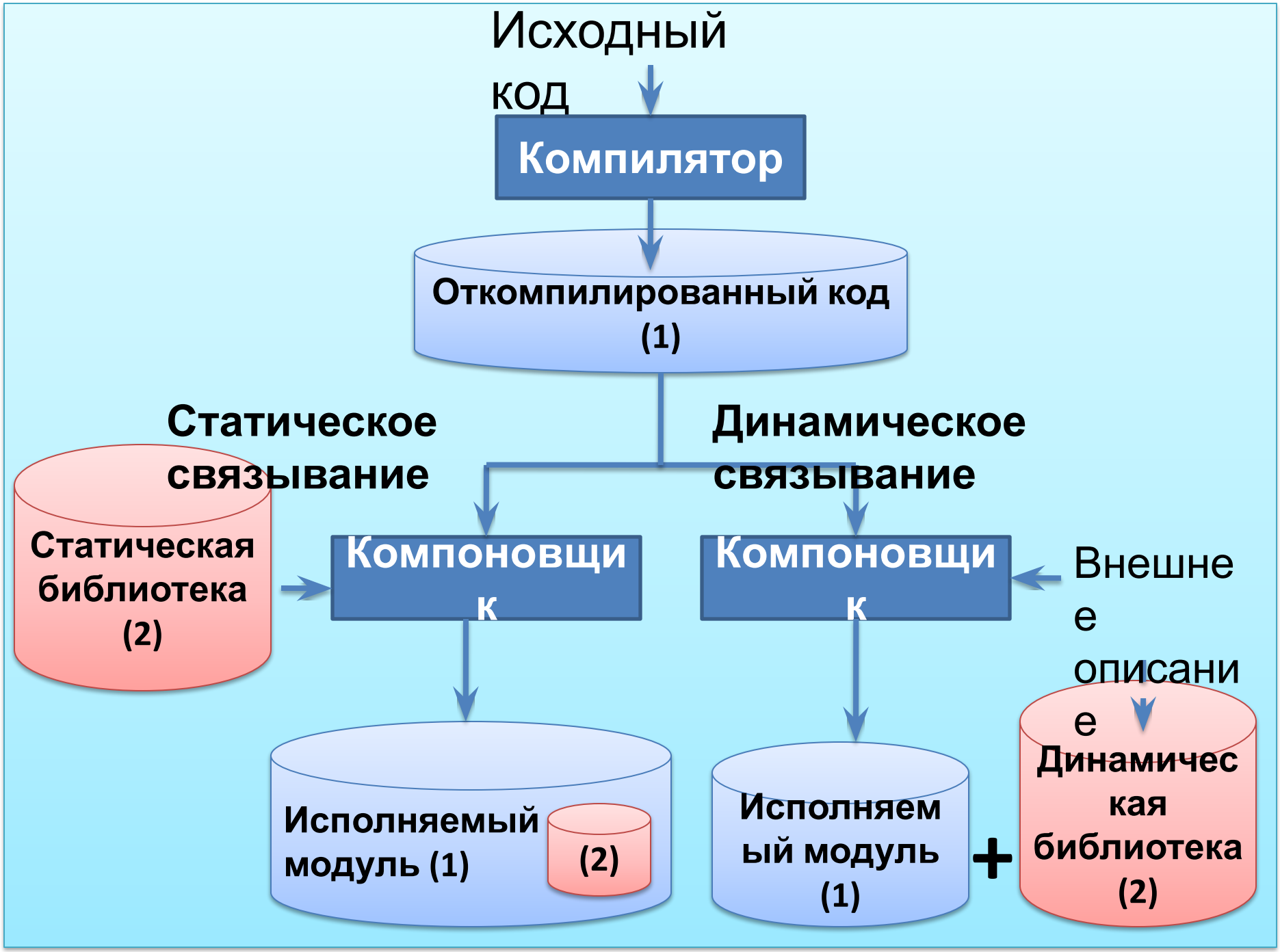
- Набор подпрограмм принято называть **библиотекой подпрограмм.**
- Библиотеки подпрограмм делятся на

❑ **библиотеки статического вызова** (статические библиотеки)

❑ **библиотеки динамического вызова** (динамические библиотеки).

- После компиляции подпрограммы ***статической библиотеки*** компоновщик добавляет ее откомпилированный код к исполняемой программе.
- Получившийся в результате исполнительный модуль содержит код программы и всех используемых подпрограмм.

- В случае ***динамической компоновки*** компоновщик просто использует информацию о подпрограмме для настройки соответствующих таблиц в исполняемом файле.
- Когда исполняемый модуль загружается в память, операционная система загружает также все необходимые динамические библиотеки и заполняет внутренние таблицы программы адресами библиотечных подпрограмм в памяти, после чего программа запускается на исполнение.


Исходный код
Компилятор
Откомпилированный код (1)
Статическое связывание
Динамическое связывание
Статическая библиотека (2)
Компоновщик
Компоновщик
Внешнее описание
Исполняемый модуль (1)
(2)
Исполняемый модуль (1)
+
Динамическая библиотека (2)