8. ОБЛАСТЬ СУЩЕСТВОВАНИЯ ИМЕНИ

После объявления имя можно использовать. Однако оно, как правило, может быть использовано только в некоторой части программы, которая называется областью существования имени.

Текст программы можно разместить в одном файле, а можно и в нескольких различных файлах, каждый из которых содержит целиком одну или несколько функций. Для объединения в одну программу информация обо всех этих файлах помещается в так называемый файл проекта. Компилятор для каждого исходного файла создаёт объектный код (файл с расширением ".obj"). Затем все объектные файлы (вместе с библиотечными) объединяются компоновщиком в исполняемый, или загрузочный модуль, который имеет имя файла проекта и расширение ".exe". Область существования имени нужна компилятору для того, чтобы сгенерировать верный машинный код.

8.1. Компиляция, компоновка, библиотеки

Рассмотрим в связи с этим несколько подробнее саму технологию подготовки программ. Подготовка программы начинается с редактирования файла, содержащего текст этой программы, который имеет стандартное расширение ".cpp". Затем выполняется его компиляция, которая включает в себя несколько фаз: препроцессор, лексический, синтаксический, семантический анализ, генерация кода и его оптимизация. В результате компиляции получается объектный модуль – некий «полуфабрикат» готовой программы, который потом участвует в ее сборке. Файл объектного модуля имеет стандартное расширение ".obj". Компоновка (сборка) программы заключается в объединении одного или нескольких объектных модулей программы и объектных модулей, взятых из библиотечных файлов и содержащих стандартные функции и другие полезные вещи. Результат компоновки – исполняемый модуль в виде отдельного файла со стандартным расширением ".exe", который загружается в память, в результате чего программа начинает выполняться (рис. 1).

Заметим, что функции, хранящиеся в библиотеках, содержатся в переносимом формате. Это означает, что адресация памяти для различных инструкций машинного кода не полностью определена. Там содержится только информация о смещении (относительные адреса). Когда программа компонуется с функциями стандартных библиотек, эти

смещения памяти используются для создания физического адреса (абсолютного адреса).

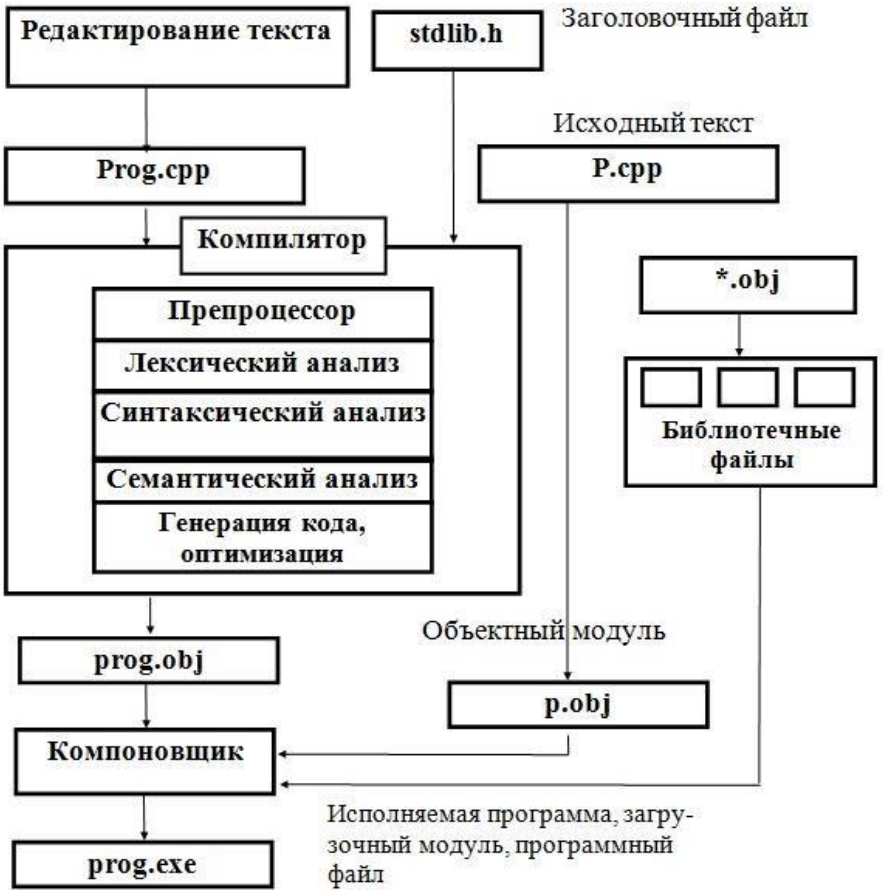

Рис. 1. Подготовка программы к выполнению

Компиляция и ее фазы

Собственно компиляция начинается с лексического анализа программы. *ЛЕКСИКА* языка программирования – это правила «правописания слов» программы, таких как константы, служебные слова, комментарии. Лексический анализ разбивает текст программы на указанные элементы. Особенность любой лексики – ее элементы представляют собой регулярные линейные последовательности символов. Например, ***ИДЕНТИФИКАТОР*** – это произвольная последовательность букв, цифр и символа '_', начинающаяся с буквы или '_'.

СИНТАКСИС языка программирования – это правила составления предложений языка из отдельных слов. Такими предложениями являются операции, операторы, определения функций и переменных. Особенностью синтаксиса является принцип вложенности (рекурсивность) правил построения предложений. Это значит, что

элемент синтаксиса языка в своем определении прямо или косвенно в одной из его частей содержит сам себя. Например, в определении оператора цикла телом цикла является оператор, частным случаем которого является все тот же оператор цикла.

СЕМАНТИКА языка программирования – это смысл, который закладывается в каждую конструкцию языка. Семантический анализ – это проверка смысловой правильности конструкции. Например, если мы в выражении используем переменную, то она должна быть определена ранее по тексту программы, а из этого определения может быть получен ее тип. Исходя из типа переменной, можно говорить о допустимости операции с данной переменной.

ГЕНЕРАЦИЯ КОДА – это преобразование элементарных действий, полученных в результате лексического, синтаксического и семантического анализа программы, в некоторое внутреннее представление. Это могут быть коды команд, адреса и содержимое памяти данных, текст программы на языке Ассемблера либо стандартизованный промежуточный код (например, так называемый P-код). В процессе генерации кода производится и его оптимизация.

Модульное программирование, компоновка

Полученный в результате трансляции ***ОБЪЕКТНЫЙ МОДУЛЬ*** включает в себя готовые к выполнению коды команд, адреса и содержимое памяти данных. Но это касается только собственных внутренних объектов программы (функций и переменных). Обращение к внешним функциям и переменным, отсутствующим в данном фрагменте программы, не может быть полностью переведено во внутреннее представление и остается в объектном модуле в исходном (текстовом) виде. Но если эти функции и переменные отсутствуют, значит, они должны быть каким-то образом получены в других объектных модулях. Самый естественный способ – написать их на том же самом C++ и откомпилировать. Это и есть принцип ***МОДУЛЬНОГО ПРОГРАММИРОВАНИЯ*** – представление текста программы в виде нескольких файлов, каждый из которых транслируется отдельно.

С модульным программированием мы сталкиваемся в двух случаях:

- когда сами пишем модульную программу;
- когда используем стандартные библиотечные функции.

БИБЛИОТЕКА ОБЪЕКТНЫХ МОДУЛЕЙ – это файл (библиотечный файл), содержащий набор объектных модулей и собственный внутренний

каталог. Объектные модули библиотеки извлекаются из нее целиком при наличии в них требуемых внешних функций и переменных и используются в процессе компоновки программы.

КОМПОНОВКА – это процесс сборки программы из объектных модулей, в котором производится их объединение в исполняемую программу и связывание вызовов внешних функций и их внутреннего представления (кодов), расположенных в различных объектных модулях. Этот этап выполняет специальная программа – LINKER.

8.2. Виды областей существования имени

Вернемся к понятию «область существования имени». Можно выделить 5 видов областей существования имени.

1. Область существования ***БЛОК***. Напомним, что блок – это фрагмент программы, заключённый в фигурные скобки { }.

Например,

```
if(a != 5){ int j
= 0; double k
= 3.5; a++;
…
}
```

Заметим, что тело любой функции является блоком.

Имя, объявленное в блоке, может быть использовано от точки, где находится его объявление, и до конца блока. Такую же область существования имеют и имена в определении функции:

```
int  f1 (int i){ return i; }
```

Имя i имеет область существования «блок». Область существования «блок» распространяется и на вложенные блоки.

2. Область существования ***ФУНКЦИЯ***. Эту область существования имеют только имена меток перехода, используемые оператором goto:

```
void f (){….
…
goto lab;
…
{… lab: ...}
…
```

}

3. Область существования ***ПРОТОТИП ФУНКЦИИ***. Прототип функции есть объявление функции, не являющееся её определением. Прототип может иметь, например, такой вид

int F(int a, double b, char* str);

Область существования «прототип» заключена между открывающей и закрывающей круглыми скобками. Иначе говоря, имена a, b, str в примере определены только внутри круглых скобок. Из этого следует, что в прототипах можно использовать для аргументов любые имена или не использовать их совсем:

int F(int, double, char*);

4. Область существования ***ФАЙЛ*** имеют имена, объявленные вне любого блока и класса. Такие имена называют глобальными. Глобальные имена определены от точки их объявления и до конца файла, где встретилось их объявление. Примером таких имён являются имена функций.

```
#include <iostream>

int a, b, c[40];              // Глобальные имена;
int f1()                      // локальное имя f1;
{int i;                       // локальное имя;
…
}
int count;                    // глобальное имя;
void f2(){ … }                // глобальное имя f2.
```

5. Область существования ***КЛАСС***.

Такую область существования имеют имена, объявленные в классах (см. раздел 17). Эти имена определены во всем классе, в котором они объявлены, независимо от точки их объявления.