Вводная лекция

Артамонов Юрий Николаевич

Содержание

Машинные языки, языки ассемблера, языки высокого уровня

Программисты пишут программы на различных языках программирования, некоторые из нинх непосредственно понятны компьютеру, другие нуждаются в промежуточной стадии *трансляции*. Сотни имеющихся языков могут быть разделены на три общих типа:

- Машинные языки - система команд конкретного центрального процессора
- Ассемблерные языки (Nasm, Tasm, Masm, Fasm)
- Языки высокого уровня (C/C++, Python, Java, Php, Haskel, Ruby, Erlang, Common Lisp, Pascal, Fortran и т.д.)

Машинные языки, языки ассемблера, языки высокого уровня (продолжение)

Каждый компьютер может понимать только свой машинный язык. Он тесно связан с аппаратной частью компьютера. Машинные языки являются *машинно-зависимыми*, т.е. конкретный машинный язык может быть использован только с определенным типом компьютера. Машинные языки неудобны для восприятия человеком, поскольку представляют собой совокупность чисел (запоминать, что означает каждое число очень неудобно).

По мере распространения информационных технологий стало ясно, что программирование на машинных языках тормозит развитие компьютерной техники. Вместо последовательности чисел, непосредственно понятных компьютеру, программисты для представления элементарных операций стали применять аббревиатуры (мнемоники), которые и сформировали основу языков ассемблера. Для преобразования программ, написанных на таких языках, в машинный язык были разработаны *программы трансляторы*, называемые *ассемблерами*.

Языки высокого уровня - общее представление

С появлением языков ассемблера использование компьютеров значительно расширилось, однако программы оставались очень детальными. Даже выполнение простейших действий требовало огромного количества инструкций. За набором этих инструкций сложно было разглядеть, что делает программа. Для ускорения процесса программирования, улучшения читаемости программного кода был использован еще один уровень абстракции - были разработаны *языки высокого уровня*. В языках высокого уровня инструкции зачастую выглядят как обычный текст на английском языке с применением общепринятых математических знаков.

В свою очередь, все языки высокого уровня делятся на два класса:

- *компилируемые языки высокого уровня* (C/C++, Pascal, Fortran). Основное достоинство - это скорость работы конечной программы, ее относительная независимость от программного окружения
- *интерпретируемые языки высокого уровня* (Python, Java, PHP, Ruby, Haskel). Основное достоинство - это гибкость разработки, отладки и сопровождения.

Основные этапы разработки программы на компилируемых языках

В интерпретируемых языках команды выполняются строчка за строчкой (трансляция в машинный язык происходит построчно в диалоге с интерпретатором).

Мы будем изучать язык высокого уровня на примере компилируемых языков C/C++. Поэтому рассмотрим основные этапы разработки программы на компилируемых языках.

В типичной Unix-системе разработка программы проходит через следующие шесть стадий:

- *редактирование*
- *препроцессорная обработка*
- *компиляция*
- *компоновка*
- *загрузка*
- *исполнение*

Основные этапы разработки программы на компилируемых языках (продоложение)

На этапе *редактирования* программа пишется в обычном текстовом редакторе и сохраняется на диск с расширением «.c» В Unix широко используются редакторы Vi, Emacs, в Windows можно использовать встроенные редакторы выбранной среды (например, Visual Studio).
На этапе *препроцессорной обработки* выполняются директивы препроцессора, которые определяют, какие действия должны быть выполнены перед компиляцией программы. Обычно это добавление других файлов к компилируемому модулю или замена специальных символов в тексте программы. Препроцессор автоматически запускается компилятором перед тем, как начинается преобразование программы в машинный язык.
На этапе *компиляции* специальная программа - компилятор транслирует С-программу в код машинного языка (который также называется *объектным кодом*).

Основные этапы разработки программы на компилируемых языках (продоложение)

Типичные С-программы используют какие-либо функции из стандартных библиотек. На этапе компиляции место в объектном коде, где происходит вызов такой функции заменяется «дыркой». На этапе компоновки такие дырки заполняются определениями соответствующих функций. В результате компоновки создается *исполняемый образ*, в котором уже нет отсутствующих частей. Это собственно и есть исполняемая программа.

Например, для компиляции и компоновки программы с именем «Hello.c» в консоле среды Unix следует напечатать:

> cc Hello.c -o Hello.out

и нажать Enter. В случае корректной компиляции и компоновки появится файл Hello.out, являющийся исполняемым файлом от исходной программы на языке высокого уровня С.

Перед выполнением программа должна быть размещена в памяти, эту задачу выполняет загрузчик, что соответствует 5 этапу.

Представление о стандартном вводе/выводе

Наконец, компьютер выполняет исполняемую программу Hello.out. Для этого в строке приглашении консоля следует напечатать:
> ./Hello.out
Большинство программ на C получают или выводят данные.
За входными данными зарезервированно специальное обозначение stdin (standard input device - стандартное устройство ввода). Как правило, stdin ассоциировано с клавиатурой (хотя может быть перенаправлено и на другое устройство).
Вывод осуществляется на stdout (standard output device - стандартное устройство вывода), которое по умолчанию ассоциировано с экраном, хотя при желании также может быть перенаправлено.
Еще имеется stderr (standard error device - стандартное устройство ошибок), оно обычно также связано с экраном.

История языка C

Язык C берет своё начало от двух языков, BCPL и B. В 1967 году Мартин Ричарде разработал BCPL как язык для написания системного программного обеспечения и компиляторов. В 1970 году Кен Томпсон использовал B для создания ранних версий операционной системы Unix.

Язык C был разработан на основе B Деннисом Ричи из Bell Laboratories и впервые был реализован в 1972 году на компьютере DEC PDP-11. Известность C получил в качестве языка операционной системы Unix. Сегодня практически все основные операционные системы написаны на C и/или C++.

C сочетает в себе основные принципы языков BCPL и B, кроме того, в нем введена типизация переменных и некоторые другие важные моменты. В конце 70-х годов C превратился в то, что мы называем «традиционный C».

История языка C (продолжение)

Публикация в 1978 году книги Кернигана и Ричи «Язык программирования C» привлекла широкое внинмание к этому языку.
Применение C для разрзличных типов компьютеров привело к появлению различных вариантов языка, которые, несмотря на свою похожесть, зачастую были несовместимыми. Это явилось серьезной проблемой для разработчиков программных продуктов. Становилось ясно, что нужна стандартная версия C. В 1983 году Американским комитетом национальных стандартов в области компьютеров и обработке информации был учрежден технический комитет, перед которым стояла задача выработать однозначное, машино-независимое определение языка C. В 1989 году был окончательно одобрен разработанный стандарт, который получил название ANSI/ISO 9899:1990. Второе издание книги Кернигана и Ричи выпущено в 1988 году и описывает эту версию C, называемую ANSI C и широко используемую в мире.

История языка C++

В лаборатории Bell Бьерном Страуструпом было разработано расширение языка C, а именно C++. Язык C++ имеет целой ряд качеств, «украшающих» обычный C. Но самое главное, он предоставляет возможности для объектно-ориентированного программирования. Объекты - это в принципе многократно используемые компоненты программного обеспечения, которые моделируют реальные сущности. В настоящее время разработчики сознают, что применение объектно-ориентированного подхода способно повысить производительность рабочих коллективов в 10-100 раз.

Пример простейшей программы на C

```
/* Первая программа на C */

main()
{
   printf("Hello world!\n");
}
```

Анализ простейшей программы на C

Строка

```
/* Первая программа на C */
```

начинается символами /* и заканчивается символами */, означающими, что эта строка является *комментарием*. Комментарии не оказывают никакого влияния на работу программы, компилятор C их просто игнорирует. Комментарии помогают понять программисту, что делает его программа.

Анализ простейшей программы на C (продолжение)

Строка

```
main()
```

должна обязательно присутствовать в каждой программе. Круглые скобки после main означают, что main является «строительным блоком» программы, называемым *функцией*. Программа на C может содержать несколько функций, однако одна из функций обязательно должна быть main.

Анализ простейшей программы на C (продолжение)

Левая фигурная скобка { должна предварять *тело* каждой функции. Соответственно *правая фигурная скобка* } должна стоять в конце каждой функции. Эта пара скобок и часть программы между ними называется *блоком*. Блок - важная программная единица в языке C.

Анализ простейшей программы на C (продолжение)

Строка

```
printf("Hello world!\n");
```

дает компьютеру команду выполнить действие, а именно вывести на экран строку символов, находящуюся внутри кавычек. Такую строку называют *символьной строкой, литералом*. Вся строка, включая printf, *аргументы* внутри круглых скобок и точку с запятой (;), называют *оператором*. Каждый оператор должен заканчиваться точкой с запятой - *символом конца оператора*. Результатом выполнения является вывод на экран сообщения «Hello world!». Символы обычно печатаются именно так, как они записаны внутри кавычек. Символ $\backslash n$ не появился на экране. Обратная косая черта \ называется *ecs-символом*. Он указывает, что printf предстоит сделать нечто нестандартное.

Анализ простейшей программы на C (продолжение)

Когда встречается обратная косая черта, printf считывает следующий за ним символ и, объединяя его с обратной косой чертой, создает *ecs-код*. Ecs-код $\backslash n$ означает новую строку, результатом является перевод курсора на начало следующей строки экрана. Ниже приведены другие ecs-коды:
$\backslash t$ - горизонтальная табуляция, перемещает курсор в следующую позицию табуляции.
$\backslash r$ - возврат каретки, перемещение курсора в начало текущей строки без перемещения на следующую строку.
$\backslash a$ - звуковой системный сигнал оповещения.
$\backslash\backslash$ - обратная косая черта, выводит на экран символ $\backslash$ при помощи printf
$\backslash$" - двойные кавычки, выводит на экран символ двойных кавычек " при помощи printf.

Второй простейший пример на C

```c
/*  Программа сложения чисел */
#include <stdio.h>

main()
{
  int integer1, integer2,sum;         /* объявленние переменных */

  printf("Введите первое число\n"); /* подсказка */
  scanf("%d ", &integer1);          /* читаем первое число */
  printf("Введите второе число\n"); /* подсказка */
  scanf("%d ", &integer2);          /* читаем второе число */
  sum = integer1 + integer2;                  /* вычисляем сумму */
  printf("Сумма равна %d \n", sum); /* печатаем сумму */

  return 0; /* показывает успешное завершение программы */
}
```

Анализ второго примера

Строка

```
#include <stdio.h>
```

является директивой для препроцессора C. Строка, начинающаяся символом #, выполняется препроцессором до того, как программа начнет компилироваться. Эта специфическая строка сообщает препроцессору, что необходимо включить в программу содержание *стандартного заголовочного файла ввода-вывода* stdio.h. Этот заголовочный файл содержит информацию и объявления, используемые компилятором во время компиляции вызовов стандартных функций ввода-вывода printf, scanf.

Анализ второго примера (продолжение)

Строка

```
int integer1, integer2,sum;
```

является объявлением. Символы integer1, integer2, sum - это имена переменных. *Переменная* - это ячейка памяти, в которую можно записывать значение, предназначенное для использования программой. Объявленные переменные принадлежат типу целых чисел, что указано специальным зарезервированным словом int. Кроме int в языке C существуют и другие типы:
float - тип вещественного числа с плавающей запятой;
char - тип одного символа;

Анализ второго примера (продолжение)

Строка

```
scanf("%d ", &integer1);
```

вызывает scanf, чтобы получить от пользователя некоторое значение. Эта функция считывает данные со стандартного устройства ввода, которым обычно является клавиатура. В нашем случае scanf имеет два аргумента, %d и &integer1.
%d - управляющая строка, которая задает формат считывания, тем самым определяется тип данных, который предстоит ввести пользователю. В нашем случае спецификация %d указывает, что введенное число будет интерпретироваться как десятичное целое. Вот несколько других спецификаций:
%u - целое десятичное число без знака;
%f - десятичное число с плавающей точкой вида [-]xx.xxxx;
%e - десятичное число с плавающей точкой в экспоненциальной форме вида [-]xx.xx e[-] xx.

Анализ второго примера (продолжение)

Второй операнд scanf &integer1 начинается со знака амперсанд &. Амперсанд, когда он используется совместно с именем переменной, следующей за ним, сообщает scanf ячейку памяти, в которой хранится переменная integer1.

Строка

```
sum = integer1 + integer2;
```

вычисляет сумму переменных integer1, integer2, и ее значение присваивается переменной sum с помощью *операции присваивания* =.

Анализ второго примера (продолжение)

Строка

```
printf("Сумма равна %d \n", sum);
```

осуществляет форматированный вывод результата. В место строки «Сумма равна», где встречается управляющая строка %d, будет вставлено значение переменной sum в формате целого числа со знаком.

Строка

```
return 0;
```

передает значение 0 среде операционной системы, если программа завершена успешно.