Эказмен по C++

Хлебников Андрей Александрович

16 сентября 2014 г.

Оглавление

Вопросы к экзамену

Структура программы на языке C++. Примеры. Этапы создания исполняемой программы.

Листинг 1: Простая программа

```
1   // my first program in C++
2
3   #include <iostream>
4   using namespace std;
5
6   int main ()
7   {
8      cout << "Hello World!";
9      return 0;
10  }
```

В листинге Простая программа приведен текст программы на языке C++, которая после исполнения выведет на экран терминала из под которого был произведен запуск текст – "Hello World!".

- `// my first program in C++`
  Это коментарий. Все строки, котрорые начинаются с двух последовательных правых «слэшей» (//) является комментариями и не имеют вляния на непосредственное поведение программы. Программист может использовать комментарии для включения в код программы текстовых пояснений.

- `#include <iostream>`
  Строки начинающиеся с «решетки»(#) являются директивами препроцессора. Выражение `#include <iostream>` говорит препроцессору, чтобы он включил стандартный файл `iostream`. Файл `iostream` является включаемым файлом стандартной библиотеки C++ отвечающий за ввод – вывод, и включен в данную программу т.к. в дальнейшем в программе будет использоваться вывод теста в стандартный поток вывода.

- `using namespace std;`
  Все элементы стандартной библиотеки C++ определены в области видимости `std`. При помощи данной конструкции мы открываем доступ к функциональности библиотеки. Данный вид записи можно встретить довольно таки часто в коде, где используется стандартная библиотека C++.

– `int main ()`
  В данной строке определяется функция `main`. Функция `main` является основной точкой входа в программу написанную при помощи языка C++. Это не значит, что перед этой функцией или после нее определять другие функции с другими именами. Просто это означает, что функция `main` будет исполнена первой при с запуске программы.

  После слова `main` идут две скобки (`()`) открывающая и закрывающая соответственно. Это означает, что функция `main` не принимает на вход никаких параметров. Есть и другие вызовыспособы вызвать функцию `main` с передачей параметров.

– `cout « "Hello world!";`
  Данная строка является выражением языка C++. `cout` – это имя стандартного потока вывода стандартной библиотеки C++ и означет последовательный ввод символов (в данном, конкретном, случае - C строку «Hello World!») в стандартный поток вывода (`cout` – обычно связан с экраном терминала).

  `cout` определен в стандартном файле `iostream` находящемся в области видимости `std`, вот поэтому выше мы при помощи выражения открыли доступ к области видимости `std`(Структура программы на языке C++. Примеры. Этапы создания исполняемой программы).

  Также выражение завершается точкой с запятой (`;`), что является конструкцией завершения описания выражения.

– `return 0;`
  Данная строка является выражением возвращающим некий результат из функции. В нашем случае возвращается 0 из функции `main`.

Процесс создания исполняемой программы можно условно разбить на 4 этапа. В любом случае, от компилятора или условий этапы могут или дополнятся либо изменяться. Примеры будут приводится с использованием компилятора `clang`

За пример, возьмем участок кода приведенный ниже:

Листинг 2: Демонстрационная программа

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int
5  main (void)
6  {
7      printf("Hello world\n");
8      return EXIT_SUCCESS;
9  }
```

1. **Препроцессор.**
   На данном этапе работа идёт только с текстовыми файлами. Здесь препроцессор объединит наш исходник и все `#include` – файлы в один большой

текстовый файл. Для нашего случая[Демонстрационная программа] это будет:

Листинг 3: Результат работы препроцессора

```
/**
Definitions in files stdio.h and stdlib.h
**/

extern int printf(const char *format, ...);

int
main(void)
{
   printf("Hello world\n");
   return 0;
}
```

Для того, чтобы посмотреть результат, необходимо исполнить команду

```
clang source.c -E -o source.pp.c
```

Результат работы препроцессора будет помещен в файл `source.pp.c`.

2. **Трансляция.**
Полученный после препроцессирования единый текстовый файл отдается на обработку транслятору. В процессе работы транслятора уже нет никакой работы со внешними файлами, путями поиска и т.п. На вход транслятору подаётся один файл с исходником, на выходе транслятора получается один файл, содержащий ассемблерный текст. А сам транслятор занимается преобразованием исходника на языке программирования в ассемблерный текст, содержащий код целевой машины на языке ассемблера.

   В нашем случае на выходе транслятора мы получим ассемблерный текст. Приведен текст, который получен в результате работы компилятора `clang`.

Листинг 4: Результат работы транслятора

```
1          .def _main;
2          .scl 2;
3          .type 32;
4          .endef
5          .text
6          .globl _main
7          .align 16, 0x90
8       _main:
9          pushl %ebp
10         movl %esp, %ebp
11         subl $12, %esp
12         calll ___main
13         leal L_.str, %eax
```

```
14          movl $0, -4(%ebp)
15          movl %eax, (%esp)
16          calll _printf
17          movl $0, %ecx
18          movl %eax, -8(%ebp)
19          movl %ecx, %eax
20          addl $12, %esp
21          popl %ebp
22          ret
23
24          .data
25      L_.str:
26          .asciz "Hello world!\n"
```

Ассемблерный текст является не просто образом будущего кода, но ещё и образом будущего объектного файла, в котором будут символьные имена меток, которые в свою очередь будут использоваться при линковке.

В нашем случае мы видим, что в коде имеются две метки. Метка `L_.str`, описывающая набор символов от строкового литерала и метка `_main`, описывающая начало функции `main`. У каждой метки есть так называемая область видимости: локальная или глобальная. Локальные метки видны только внутри данного модуля, глобальные метки видны из других модулей. Метка `L_.str` является локальной, т.к. строковой литерал – это внутренние данные нашего модуля. Метка `_main` является глобальной, т.к. эта функция должна быть видна извне (в исходнике у `main'a` нет модификатора `static`), а потому метка имеет определение `.globl`[6].

Помимо вхождения меток мы видим ещё и обращения к меткам: обращение к строковому литералу `L_.str`[13] и обращение к внешней метке `_printf`[16].

Для того, чтобы посмотреть результат, необходимо исполнить команду

```
clang source.c -S -o source.tr.S
```

Результат работы транслятора будет помещен в файл `source.tr.S`.

3. **Ассемблирование.**
Полученный ассемблерный текст далее передаётся программе – ассемблеру, которая преобразует его в объектный файл. Объектный файл представляет собой бинарные коды целевой машины плюс дополнительная информация о метках и их использовании. Информация, содержащаяся в объектном файле принципиально ничем не отличается от информации, содержащейся в ассемблерном тексте. Только весь код вместо мнемоник, понятных человеку, содержит двоичный код, понятный машине. А вместо меток, расставленных по ассемблерному тексту, объектный файл содержит специальную таблицу символов (`symbol table`), описывающую все метки из нашего ассемблерного текста и таблицу перемещений (`relocations table`), описывающую точки, где метки использовались. Эти таблицы спроектированы таким образом, чтобы с ними было удобно работать линкеру и дизассемблеру.

Конкретно в нашем случае эти таблицы будут выглядеть примерно таким образом. В объектном файле определена локальная метка `L_.str`, глобальная метка `_main` и внешняя метка `_printf`, к которой в данном файле есть обращения (но определения метки нет). В объектном файле есть использование метки `L_.str` и использования метки `_printf`

Важным моментом является то, что в объектном файле адреса ещё не настроены. Например, у нас в функции `main` есть обращение к метке `L_.str`. но в том месте кода, где происходит это обращение, адрес метки `L_.str` ещё не проставлен, т.к. он будет известен только на этапе линковки.

Объектный файл можно получить следующей командой:

```
clang -source.c -c -o source.o
```

В итоге создастся объектный файл с именем `source.o`. Просто так в него заглянуть уже не получится, т.к. это бинарный файл, но при помощи воспомогательных утилит можно посмотреть необходимую информацию. Таблица символов (меток) – `symbol table`:

```
objdump --syms source.o

source.o:  file format pe-i386

SYMBOL TABLE:
[ 0](sec -2)(fl 0x00)(ty 0)(scl 103) (nx 1) 0x00000000 fake
File
[ 2](sec 1)(fl 0x00)(ty 20)(scl 2) (nx 1) 0x00000000 _main
AUX tagndx 0ttlsiz 0x0 lnnos 0next 0
[ 4](sec 1)(fl 0x00)(ty 0)(scl 3) (nx 1) 0x00000000 .text
AUX scnlen 0x2f nreloc 3nlnno 0
[ 6](sec 2)(fl 0x00)(ty 0)(scl 3) (nx 1) 0x00000000 .data
AUX scnlen 0xe nreloc 0nlnno 0
[ 8](sec 3)(fl 0x00)(ty 0)(scl 3) (nx 1) 0x00000000 .bss
AUX scnlen 0x0 nreloc 0nlnno 0
[ 10](sec 0)(fl 0x00)(ty 0)(scl 2) (nx 0) 0x00000000 ___main
[ 11](sec 0)(fl 0x00)(ty 0)(scl 2) (nx 0) 0x00000000 _printf
```

Таблица перемещений (использований) – `relocation table`:

```
objdump --reloc source.o

source.o:  file format pe-i386

RELOCATION RECORDS FOR [.text]:
OFFSET TYPE              VALUE
00000007DISP32           ___main
0000000d dir32           .data
0000001c DISP32          _printf
```

4. **Линковка.**
   Полученный объектный файл (а их может быть несколько) отдаётся линкеру. Линкер склеивает между собой все поданные ему файлы и формирует один большой исполняемый файл. Помимо объектных файлов компилятор подаёт в линкер ещё и библиотеки. Какие-то библиотеки компилятор подаёт невидимым для пользователя образом (т.е. пользователь непосредственно в этом процессе не участвует). Какие-то библиотеки пользователь сам просит компилятор передать линкеру. В первую группу, как правило, относятся библиотеки, отвечающие за `runtime` поддержку языка программирования и библиотеки, входящие в состав стандарта языка программирования или входящие в состав стандартной библиотечной поддержки на данной операционной системе. Библиотека, содержащая реализацию функции `printf` относится именно к этой группе. Ко второй группе относятся все пользовательские библиотеки (графические библиотеки, библиотеки для работы с криптографией и прочее).

   Помимо склеивания файлов линкер ещё и занимается настройкой адресов. Поскольку весь набор кодов, требуемых для формирования программы-бинарника, уже имеется на руках у линкера, то линкер после склеивания уже однозначно может сказать, по какому адресу будет располагаться та или иная функция или переменная. Для каждого файла, поступившего на ликновку, линкер заглянет в таблицу перемещений (использований), из которой поймёт, в какое место кода какой адрес нужно прописать.

Состав языка C++. Константы и переменные C++.

Типы данных в C++.

Выражения. Знаки операций.

Сводка операций: скобки, порядок вычислений, инкремент и декремент, преобразование типа.

Сводка операций: скобки, порядок вычислений, инкремент и декремент, преобразование типа.

Основные операторы C++ (присваивание, составные, выбора, циклов, перехода). Синтаксис, семантика, примеры.

Основные операторы C++ (присваивание, составные, выбора, циклов, перехода). Синтаксис, семантика, примеры.

Этапы решения задачи. Виды ошибок. Тестирование.

Массивы (определение, инициализация, способы перебора).

Сортировка массивов (простой обмен, простое включение, простой выбор).

Сортировка массивов (простой обмен, простое включение, простой выбор).

Поиск в одномерных массивах (дихотомический и линейный).

Указатели. Операции с указателями. Примеры.

Динамические переменные. Операции new и delete. Примеры.

Ссылки. Примеры.

Одномерные массивы и указатели. Примеры.

Многомерные массивы и указатели. Примеры.

Динамические массивы. Примеры.

Символьная информация и строки. Функции для работы со строками (библиотечный файл `string.h`).

Символьная информация и строки. Функции для работы со строками (библиотечный файл `string.h`).

Функции в C++. Рекурсия. Примеры.

Прототип функции. Библиотечные файлы. Директива препроцессора `# include`.

Прототип функции. Библиотечные файлы. Директива препроцессора ♯ include.

Передача одномерных массивов в функции. Примеры.

Передача многомерных массивов в функции. Примеры.

Передача строк в функции. Примеры.

Функции с умалчиваемыми параметрами. Примеры.

Подставляемые функции. Примеры.

Функции с переменным числом параметров. Примеры.

Перегрузка функции. Шаблоны функций. Примеры.

Указатели на функции. Примеры.

Ссылки на функции. Примеры.

Типы данных, определяемые пользователем (переименовани типов, перечисление, структуры, объединения). Примеры.

Типы данных, определяемые пользователем (переименование типов, перечисление, структуры, объединения). Примеры.

Структуры. Определение, инициализация, присваивание структур, доступ к элементам структур, указатели на структуры, битовые поля структур.

Структуры. Определение, инициализация, присваивание структур, доступ к элементам структур, указатели на структуры, битовые поля структур.

Динамические структуры данных (однонаправленные и двунаправленные списки).

Динамические структуры данных (однонаправленные и двунаправленные списки).

Создание списка, печать, удаление, добавление элементов (на примере однонаправленных и двунаправлен списков).

Создание списка, печать, удаление, добавление элементов (на примере однонаправленных и двунаправленных списков).

Классы и члены: функции-члены, классы, ссылка на себя, инициализация, удаление, подстановка.

Классы и члены: функции-члены, классы, ссылка на себя, инициализация, удаление, подстановка.

Классы: друзья, уточнение имени члена, вложенные классы, статические члены, указатели на члены, структуры и объединения.

Классы: друзья, уточнение имени члена, вложенные классы, статические члены, указатели на члены, структуры и объединения.

Конструкторы и деструкторы, локальные переменные.

Объекты класса как члены, массивы объектов класса, небольшие объекты.

Потоковый ввод-вывод в C++. Открытие и закрытие потока. Стандартные потоки ввода-вывода.

Потоковый ввод-вывод в C++. Открытие и закрытие потока. Стандартные потоки ввода-вывода.

Прямой доступ к файлам.

Создание бинарных и текстовых файлов, удаление, добавление, корректировка элементов, печать файлов.

Приложение

Список вопросов

1. Структура программы на языке C++. Примеры. Этапы создания исполняемой программы
2. Состав языка C++. Константы и переменные C++
3. Типы данных в C++
4. Выражения. Знаки операций
5. Сводка операций: скобки, порядок вычислений, инкремент и декремент, преобразование типа
6. Основные операторы C++ (присваивание, составные, выбора, циклов, перехода). Синтаксис, семантика, примеры
7. Этапы решения задачи. Виды ошибок. Тестирование
8. Массивы (определение, инициализация, способы перебора)
9. Сортировка массивов (простой обмен, простое включение, простой выбор)
10. Поиск в одномерных массивах (дихотомический и линейный)
11. Указатели. Операции с указателями. Примеры
12. Динамические переменные. Операции new и delete. Примеры
13. Ссылки. Примеры
14. Одномерные массивы и указатели. Примеры
15. Многомерные массивы и указатели. Примеры
16. Динамические массивы. Примеры
17. Символьная информация и строки. Функции для работы со строками (библиотечный файл `string.h`)
18. Функции в C++. Рекурсия. Примеры

19. Прототип функции. Библиотечные файлы. Директива препроцессора `#include`
20. Передача одномерных массивов в функции. Примеры
21. Передача многомерных массивов в функции. Примеры
22. Передача строк в функции. Примеры
23. Функции с умалчиваемыми параметрами. Примеры
24. Подставляемые функции. Примеры
25. Функции с переменным числом параметров. Примеры
26. Перегрузка функции. Шаблоны функций. Примеры
27. Указатели на функции. Примеры
28. Ссылки на функции. Примеры
29. Типы данных, определяемые пользователем (переименование типов, перечисление, структуры, объединения). Примеры
30. Структуры. Определение, инициализация, присваивание структур, доступ к элементам структур, указатели на структуры, битовые поля структур
31. Динамические структуры данных (однонаправленные и двунаправленные списки)
32. Создание списка, печать, удаление, добавление элементов (на примере однонаправленных и двунаправленных списков)
33. Классы и члены: функции-члены, классы, ссылка на себя, инициализация, удаление, подстановка
34. Классы: друзья, уточнение имени члена, вложенные классы, статические члены, указатели на члены, структуры и объединения
35. Конструкторы и деструкторы, локальные переменные
36. Объекты класса как члены, массивы объектов класса, небольшие объекты
37. Потоковый ввод-вывод в C++. Открытие и закрытие потока. Стандартные потоки ввода-вывода
38. Прямой доступ к файлам
39. Создание бинарных и текстовых файлов, удаление, добавление, корректировка элементов, печать файлов

Литература