Объектно-ориентированное программирование

Введение

Какое программировани е бывает?

Языки программирования

Процедурные

Си
Pascal
Basic
Фортран
Go

Объектно-ориентированные

C++
Java
Perl
Python
Ruby
Objective-C

Декларативные (функциональные, логические)

LISP
Scala

Prolog

Немного истории

Первый объектно-оринентированный язык Simula (Симула) разработан в Осло, Норвения, 1968г.

1974 – Clu
1980 – SmallTalk

Начало 80-х годов - *Бьерн Страуструп*
Язык C++
Пытался избавиться от неудобств языка Си
«Программа на C отражает “способ мышления” процессора, а C++ - способ мышления программиста»

Какими бывают ООЯзыки?

- чистые – в классическом виде реализующие объектно-ориентируемую методологию (Simula, Smalltalk и др.)

- Гибридные – появившиеся в результате внедрения объектно-ориентированных конструкций в популярный язык (C++, Object Pascal и др.)

Достоинства ООподхода

1. Интуитивно понятные жизненные аналогии, проще представлять структуру программы
2. Хорошая структурированность и возможность поддержания модульности
3. Снижение объема избыточного кода за счет использования основных механизмов ООП – наследование, композиция и пр.
4. Проще вносить изменения в программу за счет модульности – меняются лишь необходимые части, остальные остаются без изменений.

Недостатки ООподхода

1. Снижение быстродействия программы при использование некоторых механизмов (виртуальные функции и пр.)
2. На практике разумное применение ООподхода требует опыта и глубоких знаний
3. Возможна неэффективность при распределении памяти (введение дополнительных полей и пр.)

Как создать программу на C++ в linux/Unix Основные принципы

- Текст программы создается в обычном текстовом файле (без специальных знаков форматирования)
- Если текст программы большой, ее принято разбивать на модули и каждый модуль размещать в отдельны файл (до 200 строк). Файлы принято размещать в общую директорию, возможно наличие вложенных директорий.
- Расширение файлов – любое. Принято

.c – для программ на языке Си (для всех типов ОС)

.cc – для программы на языке C++ в Linux/UNIX , .cpp – для MS DOS, Windows

.h – для заголовочных файлов

Что делать с написанной программой? Компиляция

Компиляция – процесс преобразования исходного текста программы на языке высокого уровня в исполняемый бинарный код.

Совокупность программных средств, реализующих этот процесс называется компилятором.

В OS UNIX/Linux наиболее часто используемый вариант компилятора – GCC (GNU C/C++ Compiler) - свободно распространяемый компилятор зачастую устанавливается вместе с ОС.

Какие бывают компиляторы

Существует множество компиляторов с языка C++, которые можно использовать для создания исполняемого кода под разные платформы. Проекты компиляторов можно классифицировать по следующим критериям.

1. Коммерческие и некоммерческие проекты
2. Уровень поддержки современных тенденций и стандартов языка
3. Эффективность результирующего кода

В OS UNIX/Linux наиболее часто используемый вариант компилятора – GCC (GNU C/C++ Compiler) - свободно распространяемый компилятор зачастую устанавливается вместе с ОС.

Состав компилятора GCC

GNU коллекция компиляторов включает в себя несколько языков. Из них, группу языков Си составляет три компилятора.

- **g++** — компилятор с языка C++.
- **gcc** — компилятор с языка C (GNU C Compiler).
- **gcc -lobjc** — Objective-C — это, фактически, язык C с некоторой макро-магией, которая доступна в объектной библиотеке **objc**. Ее следует поставить и указать через ключ компиляции **-l**.

Состав GCC

В состав GCC входят следующие инструментальные программные компоненты:

- препроцессор
- ассемблер
- компилятор
- компановщик (редактор связей)

Препроцессинг

Препроцессинг - обработка текстовых файлов утилитой препроцессора, который производит замены текстов согласно правилам языка препроцессора C/C++. После препроцессора, тексты компилируемых файлов, обычно, значительно вырастают в размерах, но теперь в них содержится все, что потребуется компилятору для создания объектного файла.

Основными элементами языка препроцессора являются директивы и макросимволы. Директивы вводятся с помощью символа "решетка" (#) в начале строки. Все, что следует за символом решетки и до конца строки считается директивой препроцессора. Директива препроцессора **define** вводит специальные макросимволы, которые могут быть использованы в следующих выражениях языка препроцессора. Директива **#include** подставляет вместо имени указанного за ней файла его содержимое.

РЕЗУЛЬТАТ –> файл **.i или .ii**

Ассемблирование

Ассемблирование не является обязательным процессом обработки файлов на языке C++.

По своей сути это процесс трансляции выражений одного языка в другой. Более конкретно, в данном случае, мы имеем на входе утилиты ассемблера файл с текстом на языке C++ (компиляционный лист), а на выходе мы получаем файл с текстом на языке Ассемблера. Язык Ассемблера это низкоуровневый язык который практически напрямую отображается на коды инструкций процессора целевой системы.

Ассемблирование не является обязательным процессом обработки файлов на языке C++. Это делают чтобы максимально объеденить разные языки в одну коллекцию, для каждого из языков реализуется свой транслятор на язык ассемблера и, при необходимости компилятор с языка ассемблера и линковщик делаются общими для всех языков коллекции.

РЕЗУЛЬТАТ –> файл ***.s***

Компиляция

Подразумевается компиляция с языка ассемблер

Результатом его работы является **объектный файл** полученный на основе всего того текста, что был предоставлен в компиляционном листе. Каждый объектный файл проекта соответствует одному компиляционному листу проекта.

Объектный файл — это бинарный файл, фактически состоящий из набора функций – машинных инструкций.

РЕЗУЛЬТАТ –> файл **.o** (по умолчанию имя файла = имени исходного)

Линковка (компановка)

На этапе линковки выполняется объединение всех объектных файлов проекта, откомпилированных по соответствующим компиляционным листам проекта в единую сущность. Это может быть приложение, статическая или динамическая библиотека.

Реализуется утилитой ld.

Основная задача – настройка внешних ссылок объектных модулей.

Результатом является сконструированный из объектных модулей **исполняемый** бинарный файл, содержаоий программу в формате COFE (common Object File Format) или Elf (Exetutable and Lined Format).

По умолчанию исполняемый файл **– a.out**

Расширения **.a** и **.so** используются при создании статических и динамических библиотек.

Результат компиляции

Результатом работы компилятора является перемещаемый объектный файл, а результатом работы компоновщика — файл, готовый к исполнению.

Компоновщик выполняет двойную роль:

- физически объединяет указанные в списке связей файлы в один программный файл.
- решает проблему внешних ссылок и обращений к памяти. Внешняя ссылка делается каждый раз, когда в коде одного файла упоминается код из другого файла.

Запуск процесса компиляции

Чтобы откомпилировать исходный код C++, находящийся в файле **file.cc** необходимо выполнить команду:

gcc file.cc

При этом программа выполнит все вышеуказанные процессы друг за другом без явного участия программиста

Результат – a.out

Ключи g++

Опция	Назначение
-c	Эта опция означает, что необходима только компиляция. Из исходных файлов программы создаются объектные файлы в виде **name.o**. Компоновка не производится.
-o file-name	Использовать **file-name** в качестве имени для создаваемого файла.
-O1 **-O2** **-O3**	Различные уровни оптимизации.
-O0	Не оптимизировать. Если вы используете многочисленные **-O** опции с номерами или без номеров уровня, действительной является последняя такая опция.
-I	Используется для добавления ваших собственных каталогов для поиска заголовочных файлов в процессе сборки
-L	Передается компоновщику. Используется для добавления ваших собственных каталогов для поиска библиотек в процессе сборки.
-l	Передается компоновщику. Используется для добавления ваших собственных библиотек для поиска в процессе сборки.

-E –S – Выполнить только препроцессинг или ассамблирование

Библиотеки и заголовочные файлы

Библиотека — это «сборник» кода, который можно многократно использовать в самых разных программах. Как правило, **библиотека в C++ состоит из двух частей**:

- **Заголовочный файл**, который объявляет функционал библиотеки.
- Предварительно скомпилированный бинарный файл, содержащий реализацию функционала библиотеки.

Некоторые библиотеки могут быть разбиты на несколько файлов и/или иметь несколько заголовочных файлов.

Библиотеки и заголовочные файлы

Библиотека состоит из интерфейса (объявления функций и классов) и реализации (тел функций и методов).

Интерфейс представляет собой заголовочный файл с объявлениями с расширением .H.

Реализация – откомпилированный объектный (OBJ) или библиотечный (LIB) файл.

Заголовочный файл с интерфейсом включается в любой исходный файл, использующий классы, при помощи #include.

Заголовочные файлы

В программировании заголовочный файл (англ. header file) или подключаемый файл — файл, содержимое которого автоматически добавляется препроцессором в исходный текст в том месте, где располагается некоторая директива #include <file.h> в Си).

Заголовочный файл в общем случае может содержать любые конструкции языка программирования, но на практике исполняемый код (за исключением inline-функций в C++) в заголовочные файлы не помещают.

По сложившейся традиции, в заголовочных файлах объявляют функции стандартной библиотеки Си и Си++.

Как устроены библиотеки

Библиотека представляет собой собрание функций. В отличие от объектных файлов в библиотечном файле хранится название каждой функции, объектный код функции и информация, касающаяся «перемещаемости» файла, необходимая для редактирования связей. Когда программа делает ссылку на функцию, содержащуюся в библиотеке, компоновщик отыскивает эту функцию и добавляет ее код к программе. Таким образом, к программе добавляются только те функции, которые действительно будут в ней использоваться.

Поскольку функции хранятся в библиотеке, то в исполняемый код программы войдут лишь действительно используемые в программе функции. (Если бы они входили в состав объектных файлов, программа была бы длиннее на несколько сот килобайт!)

Зачем библиотеки представлены в виде объектного кода?

Библиотеки предварительно компилируют по нескольким причинам.

- их код редко меняется. Было бы напрасной тратой времени повторно компилировать библиотеку каждый раз при её использовании в новой программе.
- поскольку весь код предварительно скомпилирован в машинный язык, то это предотвращает получение доступа к исходному коду (и его изменение) сторонними лицами. Этот пунктик важен для предприятий/людей, которые не хотят, чтобы результат их труда (исходный код) был доступен всем (интеллектуальная собственность, все дела).

Виды библиотек

Есть два типа библиотек: статические и динамические.

- **Статическая библиотека** (или ещё «**архив**») состоит из подпрограмм, которые непосредственно компилируются и линкуются с вашей программой. При компиляции программы, которая использует статическую библиотеку, весь функционал статической библиотеки (тот, что использует ваша программа) становится частью вашего исполняемого файла. В Linux статически библиотеки имеют расширение **.a** (от «**a**rchive»).
- **Динамическая библиотека** (или ещё «**общая библиотека**») состоит из подпрограмм, которые подгружаются в вашу программу во время её выполнения. При компиляции программы, которая использует динамическую библиотеку, эта библиотека не становится частью вашего исполняемого файла — она так и остаётся отдельным модулем. В Linux динамические библиотеки имеют расширение **.so** (от «**s**hared **o**bject» = «общий объект»).

Преимущества и недостатки статических и динамических библиотек

Одним из преимуществ статических библиотек является то, что вам нужно распространять всего лишь 1 файл (исполняемый файл), чтобы пользователи могли запустить и использовать вашу программу. Поскольку статические библиотеки становятся частью вашей программы, то вы можете использовать их подобно функционалу своей собственной программы. С другой стороны, поскольку копия библиотеки становится частью каждого вашего исполняемого файла, то это может привести к увеличению размера файла. Также, если вам нужно будет обновить статическую библиотеку, вам придётся перекомпилировать каждый исполняемый файл, который её использует.

Одним из преимуществ динамических библиотек является то, что разные программы могут совместно использовать одну копию динамической библиотеки, что значительно экономит используемое пространство. Ещё одним преимуществом динамической библиотеки является то, что её можно обновлять до более новой версии без перекомпиляции всех исполняемых файлов, которые её используют.

Где находятся заголовочные файлы и библиотеки

По умолчанию в директориях

/usr/include

/usr/lib

Практическое задние

Запустить программу, состоящую из трех файлов:

1. Основной
2. Функция 1
3. Функция 2, зависящая от функции 1

- Без использования заголовочных файлов и библиотек
- С использованием заголовочных файлов
- С использованием статической библиотеки