KaterynaBondarenko 21 марта 2019 в 01:10

Инкапсуляция в Си++ и Си

Программирование *, C++ *, C *

```
private_Tony.cpp: In function 'int main()':
private_Tony.cpp:7:13: error: 'int Contact::mobile_number' is private
         int mobile_number;
             ^
private_Tony.cpp:25:18: error: within this context
     cout << Tony.mobile_number << endl;
                  ^
```

Определение

Инкапсуляция это набор инструментов для управления доступом к данным или методам которые управляют этими данными. С детальным определением термина “инкапсуляция” можно ознакомиться в моей предыдущей публикации на Хабре по этой ссылке. Эта статья сфокусирована на примерах инкапсуляции в Си++ и Си.

Инкапсуляция в Си++

По умолчанию, в классе (`class`) данные и методы приватные (`private`); они могут быть прочитаны и изменены только классом к которому принадлежат. Уровень доступа может быть изменен при помощи соответствующих ключевых слов которые предоставляет Си++.

В Си++ доступно несколько спецификаторов, и они изменяют доступ к данным следующим образом:

- публичные (`public`) данные – доступны всем;
- защищенные (`protected`) – доступны только классу и дочерним классам;
- приватные (`private`) –доступны только классу которому они принадлежат.

Для краткости, только два уровня (приватный и публичный) будут освещены в примерах.

Пример инкапсуляции

В классе `Contact` , публичные переменные и методы доступны из основной программы

(main). Приватные переменные и методы могут прочитаны, вызваны или изменены только самим классом.

```
#include <iostream>
using namespace std;

class Contact
{
    private:
        int mobile_number;          // private variable
        int home_number;            // private variable
    public:
        Contact()                   // constructor
        {
            mobile_number = 12345678;
            home_number = 87654321;
        }
        void print_numbers()
        {
            cout << "Mobile number: " << mobile_number;
            cout << ", home number: " << home_number << endl;
        }
};

int main()
{
    Contact Tony;
    Tony.print_numbers();
    // cout << Tony.mobile_number << endl;
    // will cause compile time error
    return 0;
}
```

Попытка напечатать или изменить приватную переменную mobile_number из основной программы (main) вызовет ошибку при компиляции потому как доступ к приватным данным в классе ограничен.

Нарушение инкапсуляции с Друзьями (Хорошая практика)

В Си++ присутствует ключевое слово “друг” (friend) которое позволяет добавить исключения в общие правила доступа к данным. Если функция или класс названы другом (friend) класса Contact – они получают свободный доступ к защищенным или

приватным данным.

Существует два основных правила дружбы – дружба не наследуется и не взаимна. Также, наличие “друзей” не изменяет уровень защищенности данных – приватные данные остаются приватными с исключением в виде “друга”.

```
#include <iostream>
using namespace std;

class Contact
{
    private:
        int mobile_number;          // private variable
        int home_number;            // private variable
    public:
        Contact()                   // constructor
        {
            mobile_number = 12345678;
            home_number = 87654321;
        }
        // Declaring a global 'friend' function
        friend void print_numbers( Contact some_contact );
};

void print_numbers( Contact some_contact )
{
    cout << "Mobile number: " << some_contact.mobile_number;
    cout << ", home number: " << some_contact.home_number << endl;
}

int main()
{
    Contact Tony;
    print_numbers(Tony);
    return 0;
}
```

В этом примере, функция `print_numbers()` – обычная функция, не метод класса `Contact`. Объявление функции `print_numbers()` “другом” класса `Contact` – единственная причина по которой функция `print_numbers()` имеет доступ к приватным данным. Если убрать строку с определением друга – код не скомпилируется.

Примечание: друзьями лучше не злоупотреблять. Добавление друга стоит рассматривать как исключение, не как общую практику.

как исключение, не как общую практику.

Нарушение инкапсуляции с Преобразованием типов и Указателями (Плохая практика)

способом – плохая идея. Этот способ не гарантирует получения нужных данных. Он плохо читается и плохо поддерживается. Невзирая на это, он существует.

Си++ получил в наследство от Си множество инструментов, один из которых – преобразование типов (typecasting). По умолчанию, все переменные и методы в классе приватные. В то же время, стандартный уровень доступа к данным в структуре (struct) – публичный. Возможно создать структуру или полностью публичный класс в котором данные будут расположены идентично данным в классе Contact и используя преобразование типов получить доступ к приватным данным.

```
#include <iostream>
using namespace std;

class Contact
{
    private:
        int mobile_number;          // private variable
        int home_number;            // private variable
    public:
        Contact()                   // constructor
        {
            mobile_number = 12345678;
            home_number = 87654321;
        }
        void print_numbers()
        {
            cout << "Mobile number: " << mobile_number;
            cout << ", home number: " << home_number << endl;
        }
};

struct Contact_struct
{
    int mobile_number;
    int home_number;
};

int main()
```

```
{
    Contact Tony;
    Contact_struct * structured_Tony;

    Tony.print_numbers();

    structured_Tony = (Contact_struct *) & Tony;

    structured_Tony->mobile_number = 20;
    structured_Tony->home_number = 30;
    Tony.print_numbers();

    return 0;
}
```

Приватные данные были прочитаны и изменены благодаря преобразованию типов

Инкапсуляция в Си

Традиционно считается что инкапсуляция – один из ключевых ООП принципов. Тем не менее, это не лимитирует использование этого принципа в процедурно-ориентированных языках. В Си, инкапсуляция используется давно, невзирая на отсутствие ключевых слов “приватный” и “публичный”.

Приватные переменные

В контексте инкапсуляции, все данные в Си могут быть рассмотрены как публичные по умолчанию. Уровень доступа к переменным в структурах (`struct`) может быть изменен на приватный если изолировать их определение от основной программы. Нужный эффект может быть достигнут при использовании отдельных заголовочных (header, .h) и исходных (source, .c) файлов.

В данном примере, структура была определена в отдельном исходном файле “private_var.c”. Поскольку инициализация структуры в Си требует выделения и освобождения памяти, несколько вспомогательных функций были добавлены.

```
#include "private_var.h"
#include <stdio.h>
#include <stdlib.h>

struct Contact
{
```

```
    int mobile_number;
    int home_number;
};

struct Contact * create_contact()
{
    struct Contact * some_contact;
    some_contact = malloc(sizeof(struct Contact));
    some_contact->mobile_number = 12345678;
    some_contact->home_number = 87654321;
    return( some_contact );
}

void delete_contact( struct Contact * some_contact )
{
    free(some_contact);
}
```

В соответствующем заголовочном файле "private_var.h", структура Contact была объявлена, но ее содержание осталось скрытым для основной программы.

```
#ifndef PRIVATE_VAR
#define PRIVATE_VAR

struct Contact;

struct Contact * create_contact();
void delete_contact( struct Contact * some_contact );

#endif /* PRIVATE_VAR */
```

Таким образом, для “main.c” содержание структуры неизвестно и попытки прочитать или изменить приватные данные вызовут ошибку при компиляции.

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

int main()
{
    struct Contact * Tony;
    Tony = create_contact();
    // printf( "Mobile number: %d\n", Tony->mobile_number);
    // will cause compile time error
```

```
    // will cause compile time error
    delete_contact( Tony );

    return 0;
}
```

Получение доступа к приватным переменным с Указателями

Преобразование типов может быть использовано для преодоления инкапсуляции в Си также как и в Си++, но данный подход уже был описан. Зная, что в структуре данные расположены в порядке их декларации, указатели и арифметика указателей подойдет для достижения цели.

Доступ к переменным в структуре ограничен. Тем не менее, спрятаны только переменные, не память в которой хранятся данные. Указатели можно рассматривать как ссылку на адрес памяти, и если эта память доступна программе – данные сохраненные в этой памяти можно прочитать и изменить. Если указатель назначен на память в которой структура хранит свои данные – их можно прочитать. Используя то же определение структуры (те же “.c” и “.h” файлы) и модифицированный “main.c” файл, ограничение доступа было преодолено.

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

int main()
{
    struct Contact * Tony;
    Tony = create_contact();

    int * mobile_number_is_here = (int *)Tony;
    printf("Mobile number: %d\n", *mobile_number_is_here);

    int * home_number_is_here = mobile_number_is_here + 1;
    *home_number_is_here = 1;
    printf("Modified home number: %d\n", *home_number_is_here);

    delete_contact( Tony );
    return 0;
}
```

Данные в структуре были прочитаны и модифицированы

Приватные функции

Функции, будучи внешними (extern) по умолчанию, видимы во всей так называемой единице трансляции (translation unit). Другими словами, если несколько файлов скомпилированы вместе в один объектный файл, любой из этих файлов сможет получить доступ к любой функции из любого другого файла. Использование ключевого слова “статический” (static) при создании функции ограничит ее видимость до файла в котором она была определена.Следовательно, для обеспечения приватности функции необходимо выполнить несколько шагов:

- функция должна быть объявлена статической (static) либо в исходном файле (.c), либо в соответствующем заголовочном файле (.h);
- определение функции должно находиться в отдельном исходном файле.

В данном примере, в файле “private_funct.c”, была определена статическая функция print_numbers() . К слову, функция delete_contact() успешно вызывает print_numbers() поскольку они находятся в одном файле.

```
#include "private_funct.h"
#include <stdio.h>
#include <stdlib.h>

struct Contact
{
    int mobile_number;
    int home_number;
};

struct Contact * create_contact()
{
    struct Contact * some_contact;
    some_contact = malloc(sizeof(struct Contact));
    some_contact->mobile_number = 12345678;
    some_contact->home_number = 87654321;
    return( some_contact );
}

static void print_numbers( struct Contact * some_contact )
{
    printf("Mobile number: %d, ", some_contact->mobile_number);
    printf("home number = %d\n", some_contact->home_number);
}
```

```
void delete_contact( struct Contact * some_contact )
{
    print_numbers(some_contact);
    free(some_contact);
}
```

В соответствующем заголовочном файле "private_funct.h", `print_numbers()` была декларирована как статическая функция.

```
#ifndef PRIVATE_FUNCT_H
#define PRIVATE_FUNCT_H

struct Contact;

struct Contact * create_contact();
static void print_numbers( struct Contact * some_contact );
void delete_contact( struct Contact * my_points );

#endif /* PRIVATE_FUNCT_H */
```

Основная программа, “main.c”, успешно вызывает `print_numbers()` опосредовательно через `delete_contact()` , поскольку обе функции находятся в одном документе. Тем не менее, попытка вызвать `print_numbers()` из основной программы вызовет ошибку.

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

int main()
{
    struct Contact * Tony;
    Tony = create_contact();
    // print_numbers( Tony );
    // will cause compile time error
    delete_contact( Tony );
    return 0;
}
```

Получение доступа к приватным функциям

Вызвать функцию `print_numbers()` из основной программы возможно. Для этого можно

использовать ключевое слово `goto` или передавать в `main` указатель на приватную функцию. Оба способа требуют изменений либо в исходном файле “private_funct.c”, либо непосредственно в теле самой функции. Поскольку эти методы не обходят инкапсуляцию а отменяют её, они выходят за рамки этой статьи.

Заключение

Инкапсуляция существует за пределами ООП языков. Современные ООП языки делают использование инкапсуляции удобным и естественным. Существует множество способов обойти инкапсуляцию и избежание сомнительных практик поможет ее сохранить как в Си, так и в Си++.

Теги: инкапсуляция, oop concepts, encapsulation

Хабы: Программирование, C++, C

7 Карма 0 Рейтинг

@KaterynaBondarenko

Пользователь

Комментарии 10

ПОХОЖИЕ ПУБЛИКАЦИИ

7 сентября 2021 в 18:05

Строгая инкапсуляция внутренних API в JDK 17

+3 2.5K 10 0

10 мая 2021 в 02:22

Отладочный вывод на микроконтроллерах: как Concepts и Ranges отправили мой printf на покой

+20 5.5K 56 15 +15

19 марта 2019 в 15:04

Инкапсуляция в Python 3

+11 41K 92 10 +10

54k 80k 106k 132k 158k 184k 210k 236k 262k 288k

Проверить свою зарплату

ЛУЧШИЕ ПУБЛИКАЦИИ ЗА СУТКИ

31 марта в 13:03

В какую крипту не страшно вкладывать деньги: выбираем самый надежный стейблкоин из USDT, USDC, BUSD, DAI, UST

+70 8.8K 99 22 +22

31 марта в 19:01

Мозг тоже болеет: что такое антидепрессанты и нужны ли они вам вообще?

+35 8.3K 62 26 +26

31 марта в 14:02

Максимально подробное FAQ с глупыми вопросами про зубы и пломбы

+32 6.1K 63 19 +19

сегодня в 07:00

Трискета: носитель, который мы ещё помним

+30 3K 3 18 +18

31 марта в 16:00

Безопасность в компании: хоть в лоб, хоть по лбу

+30 3.4K 22 5 +5

Почему программисты пишут статьи?

Мегапост

ЧИТАЮТ СЕЙЧАС

Краткий обзор: как обстоят дела с IT-рынком и релокейтом + несколько советов IT-инженерам

19K 29 +29

Трискета: носитель, который мы ещё помним

3K 18 +18

В работе маркетплейса Wildl

2.4K 2 +2

ЦБ РФ дал отсрочку банкам д

3.5K 10 +10

Мозг тоже болеет: что такое **е?**

8.3K 26 +26

Четыре слагаемых успешной

Интересно

РАБОТА

QT разработчик
12 вакансий

Программист C++
108 вакансий

Программист C
38 вакансий

Все вакансии






Настройка языка

О сайте

Техническая поддержка

Вернуться на старую версию