

Багато файлові програми

add.cpp:

```
1 int add(int x, int y)
2 {
3     return x + y;
4 }
```

main.cpp:

```
1 #include <iostream>
2
3 int main()
4 {
5     std::cout << "The sum of 3 and 4 is: " << add(3, 4) << std::endl;
6     return 0;
7 }
```

add: ідентификатор не знайдено

Для надання доступу main.cpp до функцій add.cpp, потрібно використовувати попереднє оголошення :

```
1 #include <iostream>
2
3 int add(int x, int y); // это нужно для того, чтобы main.cpp знал, что функция add()
4
5 int main()
6 {
7     std::cout << "The sum of 3 and 4 is: " << add(3, 4) << std::endl;
8     return 0;
9 }
```

Заголовки

Заголовки мають розширення **.h**. Метою заголовних файлів є зручне зберігання набору оголошень об'єктів для їх подальшого використання в інших програмах. Досить просто написати один заголовок і його можна повторно використовувати в будь-якій кількості програм.

```
1 // Начинаем с директив препроцессора. ADD_H - это произвольное уникальное имя (обычно испо
2 #ifndef ADD_H
3 #define ADD_H
4
5 // А это уже содержимое заголовочного файла
6 int add(int x, int y); // прототип функции add() (не забывайте точку с запятой в конце!)
7
8 // Заканчиваем директивой препроцессора
9 #endif
```

add.h

main.cpp

```
1 #include <iostream>
2 #include "add.h"
3
4 int main()
5 {
6     std::cout << "The sum of 3 and 4 is " << add(3, 4) << std::endl;
7     return 0;
8 }
```

add.cpp

```
1 int add(int x, int y)
2 {
3     return x + y;
4 }
```

Директиви препроцесора

#include

#include <filename>

#include "filename"

#define

#define identifier

#define identifier substitution_text

#ifdef

#ifndef

#endif

#ifdef

```
1 #define PRINT_JOE
2
3 #ifdef PRINT_JOE
4     std::cout << "Joe" << std::endl;
5 #endif
6
7 #ifdef PRINT_BOB
8     std::cout << "Bob" << std::endl;
9 #endif
```

```
1 #define MY_FAVORITE_NUMBER 9
2
3 std::cout << "My favorite number is: " << MY_FAVORITE_NUMBER << std::endl;
```

```
1 std::cout << "My favorite number is: " << 9 << std::endl;
```

My favorite number is: 9

#ifndef

```
1 #ifndef PRINT_BOB
2     std::cout << "Bob" << std::endl;
3 #endif
```

Створюємо новий клас

File-New-Class

Class definition

Class name:

Arguments:

☐ Has destructor ☐ Has copy ctor
☒ Virtual destructor ☐ Has assignment op.

Inheritance

☐ Inherits another class

Ancestor:

Ancestor's include filename:

Scope:

Member variables

Remove

Add new:

☒ Add "Getter" method
☒ Add "Setter" method
☒ Remove prefix:

Add

Documentation

☐ Add documentation where appropriate

File policy

☒ Add paths to project ☐ Use relative path

☒ Header and implementation file shall be in same folder

Folder: ...

☐ Header and implementation file shall always be lower case

Header file

Folder: ...

Filename:

☒ Add guard block in header file

Guard block:

Implementation file

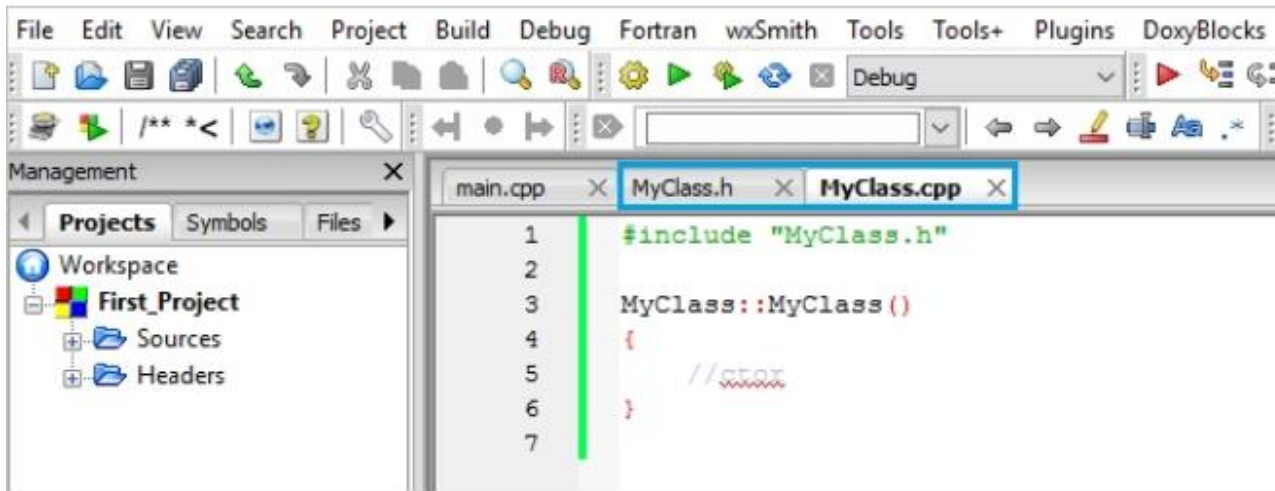
☒ Generate implementation file

Folder: ...

Filename:

Header include:

Create Cancel



MyClass.h

```
#ifndef MYCLASS_H
#define MYCLASS_H

class MyClass
{
public:
    MyClass();
protected:
private:
};

#endif // MYCLASS_H
```

MyClass.cpp

```
#include "MyClass.h"

MyClass::MyClass()
{
    //ctor
}
```

Подвійна двокрапка у початковому файлі (.cpp) називається **оператором дозволу** до області видимості, і він використовується для визначення конструктора.

MyClass::MyClass() посилається на член -функцію *MyClass ()* - або, у цьому випадку, на конструктор класу *MyClass*

```
#include <iostream>
#include "MyClass.h"
using namespace std;
```

```
int main() {
    MyClass obj;
}
```

Заголовок (header) оголошує, що **class** буде робити, а початковий **.cpp** файл визначає, як він буде його виконувати

MyClass.cpp

```
#include "MyClass.h"
#include <iostream>
using namespace std;

MyClass::MyClass()
{
    cout<<"Constructor"<<endl;
}

MyClass::~MyClass()
{
    cout<<"Destructor"<<endl;
}
```

Date.h

```
#ifndef DATE_H
#define DATE_H

class Date
{
private:
    int m_day;
    int m_month;
    int m_year;

public:
    Date(int day, int month, int year);

    void SetDate(int day, int month, int year);

    int getDay() { return m_day; }
    int getMonth() { return m_month; }
    int getYear() { return m_year; }
};

#endif
```

Деструктори

```
class MyClass {
public:
    ~MyClass() {
        // some code
    }
};
```

MyClass.h

```
class MyClass
{
public:
    MyClass ();
    ~ MyClass ();
};
```

Date.cpp

```
#include "Date.h"

// Конструктор класу Date
Date::Date(int day, int month, int year)
{
    SetDate(day, month, year);
}

// Метод класу Date
void Date::SetDate(int day, int month, int year)
{
    m_day = day;
    m_month = month;
    m_year = year;
}
```

Деструктори також є спеціальними функціями. Їх викликають, коли об'єкт знищують або видаляють. Об'єкти руйнуються, коли вони виходять за межі або коли вираз **delete** застосовується до вказівника, спрямованого на об'єкт класу. Ім'я деструктора таке, як і клас, тільки з префіксом тильди (~). Деструктор не може повернути значення або приймати будь-які параметри.

Оскільки деструктори не можуть приймати параметри, вони також не можуть бути перевантажені. У кожному класі може бути лише один деструктор.

Функції-члени

MyClass.h

```
class MyClass  
{  
public:  
    MyClass();  
    void myPrint();  
};
```

MyClass.cpp

```
#include "MyClass.h"  
#include <iostream>  
using namespace std;  
  
MyClass::MyClass() {  
}  
  
void MyClass::myPrint() {  
    cout << "Hello" << endl;  
}
```

```
#include "MyClass.h"  
  
int main() {  
    MyClass obj;  
    obj.myPrint();  
}  
  
// Outputs "Hello"
```

вказівник

```
MyClass obj;  
MyClass *ptr = &obj;
```

Оператор доступу до елементів об'єкту

```
MyClass obj;  
MyClass * ptr = & obj;  
ptr-> myPrint ();
```

Оскільки **myPrint ()** є регулярною функцією-членом, необхідно вказати його тип повернення як при оголошенні, так і при визначенні.

Оператор доступу до елементів об'єкту (->) використовується при зверненні до елементів через вказівник.

Під час роботи з об'єктом використовуйте оператор доступу точка(.). Працюючи з вказівником на об'єкт, використовуйте оператор доступу стрілка (->).

Константи

Константа є виразом з фіксованим значенням. Її не можна змінити під час роботи програми. Використовуйте ключове слово `const` для визначення постійної змінної.

```
const int x = 42;
```

```
const MyClass obj;
```

Як і вбудовані типи даних, ми можемо зробити об'єкти класу постійними за допомогою ключового слова `const`.

Усі змінні `const` повинні бути ініціалізовані при їх створенні. У випадку класів ця ініціалізація проводиться через конструктори. Якщо клас не ініціалізований за допомогою параметризованого конструктора, повинен бути наданий загальнодоступний конструктор за замовчуванням - якщо не буде надано загальнодоступний конструктор за замовчуванням - інакше станеться помилка компілятора. Після ініціалізації об'єкта класу `const` через конструктор, ви не можете змінювати змінні елемента об'єкта.

Тільки не констатні об'єкти, можуть викликати не констатні функції. Константий об'єкт не може викликати звичайні функції. Отже, для роботи констатного об'єкта потрібна констатна функція.

Щоб задати функцію як константну, ключове слово **`const`** має слідувати за прототипом функції, зовні закриваючих дужок параметрів функції. Для констатних функцій, що оголошені за межами визначення класу, ключове слово **`const`** повинно використовуватися як в прототипі функції, так і в визначенні.

MyClass.cpp

MyClass.h

```
клас MyClass  
{  
    public:  
        void myPrint () const ;  
};
```

```
#include "MyClass.h"  
#include <iostream>  
using namespace std;  
  
void MyClass::myPrint() const {  
    cout << "Hello" << endl;  
}
```

```
int main() {  
    const MyClass obj;  
    obj.myPrint();  
}  
// Outputs "Hello"
```

Тепер функція myPrint() є констатною функцією-членом. Вона може бути викликана константним об'єктом.

С ++ надає зручний синтаксис для ініціалізації елементів класу, який називається **списком ініціалізаторів**. Список ініціалізаторів може використовуватися для присвоєння значень змінним-членам

MyClass.h

```
class MyClass {  
    public:  
        MyClass(int a, int b) {  
            regVar = a;  
            constVar = b;  
        }  
    private:  
        int regVar;  
        const int constVar;  
};
```

```
class MyClass {  
    public:  
        MyClass(int a, int b)  
            : regVar(a), constVar(b)  
        {  
        }  
    private:  
        int regVar;  
        const int constVar;  
};
```

```
class MyClass {  
    public:  
        MyClass(int a, int b);  
    private:  
        int regVar;  
        const int constVar;  
};
```

MyClass.cpp

```
MyClass::MyClass(int a, int b)  
    : regVar(a), constVar(b)  
    {  
        cout << regVar << endl;  
        cout << constVar << endl;  
    }
```

```
#include "MyClass.h"  
  
int main() {  
    MyClass obj(42, 33);  
}  
  
/*Outputs  
42  
33  
*/
```

Композиція

У C ++ композиція об'єктів передбачає використання класів як змінних членів в інших класах.

Цей зразок програми демонструє композицію в дії. Він містить класи **Person** та **Birthday**, і кожний клас **Person** має об'єкт **Birthday** як елемент класу.

```
class Birthday {
public:
    Birthday(int m, int d, int y)
    : month(m), day(d), year(y)
    {
    }
private:
    int month;
    int day;
    int year;
};
```

```
class Birthday {
public:
    Birthday(int m, int d, int y)
    : month(m), day(d), year(y)
    {
    }
    void printDate()
    {
        cout<<month<<"/"<<day
        <<"/"<<year<<endl;
    }
private:
    int month;
    int day;
    int year;
};
```

```
#include <string>
#include "Birthday.h"

class Person {
public:
    Person(string n, Birthday b)
    : name(n),
      bd(b)
    {
    }
private:
    string name;
    Birthday bd;
};
```

```
class Person {
public:
    Person(string n, Birthday b)
    : name(n),
      bd(b)
    {
    }
    void printInfo()
    {
        cout << name << endl;
        bd.printDate();
    }
private:
    string name;
    Birthday bd;
};
```

```
int main() {
    Birthday bd(2, 21, 1985);
    Person p("David", bd);
    p.printInfo();
}
```

```
/*Outputs
David
2/21/1985
*/
```

Дружні функції

Оголошення функції, як не член класу, з використанням ключового слова **friend** дозволяє отримувати доступ до прихованих полів класу. Це досягається шляхом включення декларації цієї зовнішньої функції в середині класу з ключовим словом **friend**.

```
class MyClass {  
public:  
    MyClass() {  
        regVar = 0;  
    }  
private:  
    int regVar;  
  
    friend void someFunc(MyClass &obj);  
};
```

Функція **someFunc ()** визначається як звичайна функція поза класом. Вона приймає об'єкт типу **MyClass** як свій параметр і може отримати доступ до приватних членів даних цього об'єкта.

```
class MyClass {  
public:  
    MyClass() {  
        regVar = 0;  
    }  
private:  
    int regVar;  
  
    friend void someFunc(MyClass &obj);  
};  
  
void someFunc(MyClass &obj) {  
    obj.regVar = 42;  
    cout << obj.regVar;  
}
```

Функція **someFunc ()** змінює приватні елементи об'єкта і друкує його значення. Щоб зробити його членами доступними, клас повинен оголосити функцію **friend** у своєму визначенні. Ви не можете "зробити" функцію дружньою до класу без "погодження" класу.

```
int main () {  
    MyClass obj;  
    someFunc (obj);  
}  
  
// Виводи 42
```

Тепер можемо створити об'єкт в функції **main** та викликати функцію **someFunc ()** :

Ключове слово THIS

Кожен об'єкт у C++ має доступ до власної адреси через важливий покажчик, який називається **this**. В середині функції-члена **this** може використовуватися для посилання на об'єкт, що викликається.

```
class MyClass {  
public:  
    MyClass(int a) : var(a)  
    {}  
    void printInfo() {  
        cout << var<<endl;  
        cout << this->var<<endl;  
        cout << (*this).var<<endl;  
    }  
private:  
    int var;  
};
```

```
int main() {  
    MyClass obj(42);  
    obj.printInfo();  
}  
  
/* Outputs  
42  
42  
42  
*/
```

This є покажчиком на об'єкт, так що оператор вибору стрілка використовується для вибору змінної - члена.

Ключове слово this грає важливу роль в перевантаженні операторів.

Перевантаження оператора

```
1 int a = 5;  
2 int b = 6;  
3 std::cout << a + b << '\n';
```

```
1 double m = 4.0;  
2 double p = 5.0;  
3 std::cout << m + p << '\n';
```

```
1 Mystring hello = "Hello, ";  
2 Mystring world = "World!";  
3 std::cout << hello + world << '\n';
```

Більшість вбудованих операторів C++ можна перевизначити або перевантажити. Таким чином, оператори можуть використовуватися і з визначеними користувачем типами

Оператори, які можуть бути перевантажені

+	-	*	/	%	^
&		~	!	,	=
<	>	<=	>=	++	--
<<	>>	==	!=	&&	
+=	-=	/=	%=	^=	&=
=	*=	<<=	>>=	[]	()
->	->*	new	new[]	delete	delete[]

```
class MyClass {
public:
    int var;
    MyClass() {}
    MyClass(int a)
    : var(a) {}
};
```

```
class MyClass {
public:
    int var;
    MyClass() {}
    MyClass(int a)
    : var(a) {}

    MyClass operator+(MyClass &obj) {
    }
};
```

```
class MyClass {
public:
    int var;
    MyClass() {}
    MyClass(int a)
    : var(a) {}

    MyClass operator+(MyClass &obj) {
        MyClass res;
        res.var= this->var+obj.var;
        return res;
    }
};
```

Перевантажені оператори - це функції, що визначені ключовим словом **operator**, за яким слідує символ оператора, що визначається.

Перевантажений оператор схожий на інші функції тим, що має тип повернення та список параметрів

Оголошений новий об'єкт **res**. Потім присвоїли суму змінних членів поточного об'єкта (**this**) та параметр об'єкта (**obj**) до змінної **var** об'єкта **res**. Об'єкт **res** повертається як результат. Це дає нам можливість створювати об'єкти у функції **main** та використовувати оператор **+** для їх додавання.

```
int main() {
    MyClass obj1(12), obj2(55);
    MyClass res = obj1+obj2;

    cout << res.var;
}

//Outputs 67
```