

C++

Data Structures & Classes

*originally developed by Weiguang Cui,
refined by Alexander Knebe*



typedef



struct



class



Access



Inherit



Construct



.h / .cpp

Data Types

Alias & Structure

typedef · using · struct · nested structures · pointer to struct

Fundamental Data Types in C++

char (Character)

Represents a single character, e.g. 'A' or '\$'. One byte. Variants: `wchar_t`, `char16_t`, `char32_t` for wider characters.

int (Integer)

Whole numbers, e.g. 7 or 1024. Sizes: `short`, `int`, `long`, `long long`. Each can be signed (negative ok) or unsigned.

float / double (Floating-point)

Real numbers with decimals, e.g. 3.14. `float` ≈ 6 digits; `double` ≈ 15 digits; `long double` ≈ 18+ digits precision.

bool (Boolean)

Stores only true or false. Internally 0 = false, any non-zero = true.

Type Aliases — typedef & using

typedef and using are equivalent — both create a new name (alias) for an existing type.

using (modern C++11+)

```
● ● ●
using C      = char;
using WORD   = unsigned int;
using pChar  = char*;
using field  = char[50];

C      mychar, anotherchar, *ptc1;
WORD   myword;
pChar  ptc2;
field  name;
```

typedef (classic C)

```
● ● ●
typedef char      C;
typedef unsigned int WORD;
typedef char*     pChar;
typedef char      field[50];

C      mychar, anotherchar, *ptc1;
WORD   myword;
pChar  ptc2;
field  name;
```

Try both examples. Are they interchangeable? Which do you prefer for readability? (Modern C++ recommends using.)

Data Structures — struct

A struct groups data elements of different types under one name — a user-defined compound data type.

```
struct product {
    int    weight;
    double price;
};
product apple;
product banana, melon;

// Or declare objects inline:
struct product {
    int    weight;
    double price;
} apple, banana, melon;
```

Nesting structs

```
struct movies_t {
    string title;
    int    year;
};

struct friends_t {
    string  name;
    string  email;
    movies_t favorite_movie;
} charlie, maria;

friends_t* pfriends = &charlie;
```

Where you define a struct matters — it must be visible before it is used. Structs can be nested (a struct member can be another struct).

struct — Accessing Members & Pointers



```
// Dot operator — access via object name
apple.weight
apple.price
charlie.name
maria.favorite_movie.title
charlie.favorite_movie.year

// Arrow operator — access via pointer
pfriends->favorite_movie.title
// pfriends->x is equivalent to (*pfriends).x
```



```
// Pointer to struct:
friends_t* p = &charlie;
p->name = "Charlie";
cout << p->favorite_movie.title;
```

. vs -> operators

obj.member

Use when you have the object directly (by value or reference).

ptr->member

Use when you have a pointer to the object. Equivalent to (*ptr).member.

Is struct name an address?

No — unlike an array name, a struct name is not a pointer. You need & to get its address.

Try it

Can you use . with a pointer? Can you use -> with a plain struct variable?

Exercise I — Return Two Values via a struct

Write a function that computes both x^2 and $x^{1/2}$ (square and square root) for a given x .

Return both results at once using a struct.

Hints

- Define a struct with two double members, e.g. `struct Result { double square; double sqroot; };`
- Your function signature could be: `Result compute(double x);`
- What happens if x is negative when computing the square root?

Class

An expanded concept of data structures

access specifiers · member functions · inheritance · constructors

Class — Concept & Access Specifiers

OOP (C++): breaks a problem into objects with data + behaviour.

Procedural (C): breaks a problem into variables, data structures, and functions.



```
class ClassName {  
    access_specifier_1:  
        member1;  
    access_specifier_2:  
        member2;  
} object_names;
```

private

Default. Only the class itself and friend functions can access. Hidden from outside.

protected

Like private, but also accessible from derived (child) classes.

public

Accessible from anywhere in the program. The public interface of the class.

An object is an instantiation of a class — just as a variable is an instance of a type. You can have many objects of the same class.

Class — private, public & the Scope Operator



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

class Line {
public:
    double length;
    void    setLength(double len);
    double getLength(void);
};

double Line::getLength(void) { return length; }
void    Line::setLength(double len) { length = len; }

int main() {
    Line line;
    line.setLength(6.0);
    cout << line.getLength() << endl;    // via method

    line.length = 10.0;                  // direct (only works
because public)
    cout << line.length << endl;
    return 0;
}
```

Key Points

Line::getLength

The :: scope operator defines a member function outside its class body. Format: ClassName::funcName

public length

Because length is public it can be set directly. Make it private and the direct access line.length = 10.0 becomes a compile error.

Try it

Set length to private. Which lines break? How do you fix them? What if you define a function outside but never declare it inside the class?

Class — Inheritance & protected



```
// Base class (parent)
class Vehicle {
public:
    string brand = "Ford";
    void honk() { cout << "Tuut!\n"; }
};

// Derived class (child)
class Car : public Vehicle {
public:
    string model = "Mustang";
};

int main() {
    Car myCar;
    myCar.honk(); // inherited
    cout << myCar.brand + " " + myCar.model;
    return 0;
}
```

Inheritance rules

class Car : public Vehicle

Car inherits all public and protected members of Vehicle. The child gets the parent's interface.

public / protected / private inheritance

public (most common): public members stay public. Remove it and try — what changes?

protected brand

Set brand to protected. The derived class Car can still access it, but external code cannot.

Try it

Add a function in Car that prints brand. Does it work when brand is protected vs private?

Class — Constructors



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

class Rectangle {
    int width, height;
public:
    Rectangle(int, int);          // constructor declaration
    int area() { return width * height; }
};

// Constructor definition (no return type!)
Rectangle::Rectangle(int a, int b) {
    width = a;
    height = b;
}

int main() {
    Rectangle rect(3, 4);
    Rectangle rectb(5, 6);
    cout << rect.area() << endl;  // 12
    cout << rectb.area() << endl; // 30
    return 0;
}
```

About constructors

Name = class name

The constructor has the same name as the class and no return type — not even void.

Called automatically

Runs whenever a new object is created. You cannot skip it if it requires parameters.

Default values

Can you give constructor parameters default values?
e.g. `Rectangle(int a=1, int b=1);`

Multiple constructors

Yes — overloaded constructors with different parameter lists are allowed.

Try it

Can the constructor be private or protected? What effect does that have?

Exercise I — Student Class

Write a C++ class called Student with:

Data members: name, marks (an array or vector of marks)

Member functions: calc() — computes the mean of the marks

disp() — prints name and mean mark on separate lines

The program should ask the user to enter the name and marks via cin, then call disp() to show the results.

Hints

- Decide whether marks should be private — if so, how does the user supply them?
- Consider using a constructor to set name and marks at creation time.
- For the mean: sum all marks and divide by the count.

Exercise II — Complex Number Class

Write a class for complex numbers. The program should:

1. Ask for the real and imaginary parts of two complex numbers.
2. Add them together.
3. Compute and display the absolute value (modulus): $|z| = \sqrt{a^2 + b^2}$
4. Compute and display the phase (argument): $\arg(z) = \text{atan2}(b, a)$

Hints

- Use `atan2(b, a)` from `<cmath>` for the phase — it handles all four quadrants correctly.
- Overloading the `+` operator inside the class makes addition look natural: `z1 + z2`.
- `abs()` for complex requires `<complex>`, or compute it manually with `sqrt`.

The .h and .cpp File Split

.h — Header file

Contains shared declarations: function prototypes, class definitions, struct definitions, type aliases, constants.

Safe to `#include` from many .cpp files — compiled repeatedly.

.cpp — Source file

Contains definitions and private declarations: function bodies, class method implementations.

Compiled once per build. Main purpose: things that must only appear once.

Questions to explore

Can you put `main()` in a .h file? Can you compile only a .h file?

`#include "matrix.h"` vs `#include <matrix.h>` — what is the difference?

Can you load system libraries with `""`? Can you do `#include "matrix"` without .h?

Can you `#include "my_library.cpp"`? What is the ultimate way to compile multiple .cpp files?

Compiling Multiple Files



```
// mymain.cpp
#include "myheader.h"
int main() {
    cout << sum_up(3, 4) << endl;
    return 0;
}

// myheader.h
#include <iostream>
using namespace std;
int sum_up(int a, int b);    // declaration

// sum.cpp
#include "myheader.h"
int sum_up(int a, int b) {
    cout << "sum of " << a << " and " << b << endl;
    return a + b;
}
```

Build steps

g++ mymain.cpp

Fails — `sum_up` is declared but its definition is in `sum.cpp` which isn't included.

g++ mymain.cpp sum.cpp -o out

Correct — the linker finds `sum_up`'s definition in `sum.cpp`.

Declaration in .h

`sum_up` must be declared in `myheader.h` so `mymain.cpp` knows about it. Without the declaration it won't compile.

#include "myheader.h" in sum.cpp

Simpler than repeating `#include <iostream>` in every file — but changing the header affects all files that include it.

The ultimate approach for larger projects: use a Makefile or CMake to manage which files get compiled and linked together automatically.