Lec 1: 13.1: Procedural and Object-Oriented

Sept. 9, 2025

- Object-oriented
  - ↳ based on data
  - ↳ use functions that use data
  - ↳ objects are instances
- Class
  - ↳ map/blueprints
  - ↳ initialize object
- Procedural vs Object
  - ↳ if data structs, functs must ∆
  - ↳ complex func hierarchies
- struct, like a class, but vars and functions in the class can have diff properties than class
  - ↳ object is instance of class
- attributes: members of a class
- methods or behaviours: member functions of class
- data hiding: restrict access to certain members of object
- public interface: members of object that are available outside object

- Format:

```
class ClassName
{

    declaration;
    declaration;

};
```

```
class Rectangle
{
  private:
    double width;
    double length;
    double height;
  public:
    void setWidth;
    void setLength;
    void setHeight;
    double getWidth() const;
    double getLength() const;
    double getArea() const;
    double getVolume() const;
};
```

set private

"gets" private

public members

public vs private

↳ can be accessed outside of class by only functs that are members of class

- can be listed in any order
- can appear multiple times in class
- if not specified, private
- const: specifies func will not Δ

```
int Rectangle::setWidth(double w)
{
    width = w;
}
```

- mutator: member func that stores value in private.
  setter!
- accessor: retrieve values from private.
  getter!

```
Rectangle r; ← object
r.setWidth(5.2);
cout << r.getWidth();
```

- will fail if dot operator tries to access private func

* Try making calculator w/ class

Reference: 13-1 to review

* Do all ch. 13 programs
* typing.com – 60wpm ↳ send rectangle output
  * ↳ write rectangle program w/ volume

- programming for midterm, final, project
* Do program challenges in back of textbook

State Data

↳ best to calc value of data w/in member fun
rather than store in var

Pointer

```
*ptr = nullptr;
```

```
rPtr = &otherRectangle;
rPtr->setLength(12.5);
cout << rPtr->getLength <<
```

- Dynamic Allocation

```
Rectangle *rectPtr = nullptr;
rectPtr = new Rectangle; } dynamically allocate memory
rectPtr->setWidth(10.0); } store values
rectPtr->setLength(15.0); }
delete rectPtr; } delete object from memory
rectPtr = nullptr; }
```

Went from ASCII to Unicode
- ↳ 1 byte
- ↳ 2 bytes
  - ↳ first 8 bits are all 0s for ASCII conversion
  - ↳ 1 byte

Private Members
- ↳ public functs define class's public interface
- ↳ data can only be accessed publicly

Classes = Libraries

- Separating Specification from Implementation
  - ↳ header for midterm
- Inline Member Functs
  - ↳ inline: in class declaration
  - ↳ after the class declaration

```
#ifndef RECTANGLE_H
#define RECTANGLE_H
  ...
#endif
```

- Constructors
  - ↳ constructor funct name is class name
  - ↳ no return type

public:

Rectangle(); constructor

- constructor initializes values

* Input validation code

* Do Program 13-7
  - ↳ include height, volume

In-Place Initialization

Default Constructors

- ↳ C++ autoconstructs one
- ↳ default constructor takes no arguments
- ↳ simple instantiation of class calls default constructor
  - ↳ Rectangle r

↳ Pass args to constructor

- ↳ can have >1 constructor
- ↳ Rectangle r(10, 5);
- ↳ Rectangle(double =0, double 0);
- ↳ when can requires args, no default const.

Destructors

- ↳ 1 that does all destruction
- ↳ ~Rectangle
  - ↳ No return type; no args
  - ↳ one per class; cannot be overloaded
  - ↳ if cons allocates dyn mem, dest releases it

Contents of InventoryItem.h

Cons, Dests, Dyn Allocation

```
Rectangle *r = new Rectangle (10, 20)
```

↳ deletion?

Overloading Constructors

```
Rectangle();
Rectangle(double);
Rectangle(double, double);
```

Programming parameters

- ↳ validation
- ↳ comments

Constructor Delegation

↳ In C++11, can use one constructor to fill another constructor

Square (int p = 0) will not compile

Member Funct Overloading

Private Member Functs

↳ can only be called by another member funct

Arrays of Objects

InventoryItem inventory[40];

- initializer vs constructor

* constructors you can always overload

Accessing Objects in an Array

inventory[2].setUnits(30);

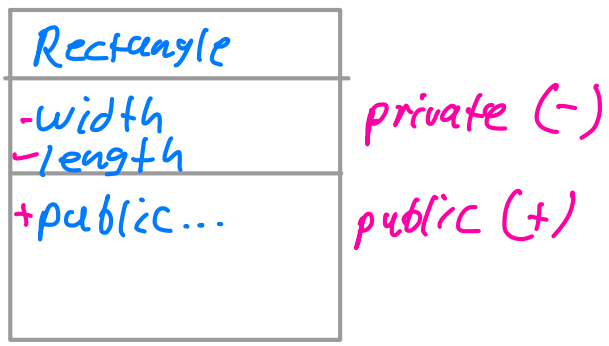
cout<< ...

Hex	Dec
0	0
⋮	⋮
F	9

cout << "Inventory Item" << setw(8)

* Add subtotals and totals

The Unified Modeling Language



UML

+ setWidth(w: double) : void

* Written in Midterm + Final

HW1

↳ RectEagle

↳ Inventory ——→ add 5 objects

↳ ContactInfo

Lec 3: 7: Arrays and Vectors

Sept. 20, 2025

cplusplus.com/reference/vector/vector/

Arrays

↳ values stored in adjacent memory locations

↳ [] operator

Int is data type of array elements

↳ name

↳ size declarator

↳ size of array (func)

↳ number of elements · _____

↳ eg) tests[5] is an array of 20 bytes, each int is 4 bytes

↳ eg) long double is 80 bytes assuming 8 bytes for a long double

↳ 64 bits (8 bits to byte)

Const

↳ const are fixed

↳ used to declare array size

Accessing Array Elems

↳ each element is assigned a unique subscript

↳ cout << array //illegal

* Program 7-1, due

Initializing Array

```
const int ARRAY_SIZE = 5;
...
for (int count = 0; count < ARRAY_SIZE; count++;
    numbers[count] = 99;
```

Global Array = all elements initialize to 0

Local Array = all elements uninitialized

↳ initialize ourselves

```
int tests[SIZE] = {79, 82, 91, 77, 84};
```

↳ can't initialize beyond size

Partial Initialization

↳ remainder will be set to 0

```
int quizzes[] = {12, 17, 23, 24}
```

↳ open, memory decided by computer

No bounds checking in C++

↳ will grab junk in memory

Off-by-One Errors

The Range-base for loop

```
int numbers[] = {1, 2, 3};
for (int val : numbers)
    cout << val << endl;
```

↳ cannot modify array this way

```
for (int &val : numbers)
```

↳ modifies array

```
for (auto &val : numbers)
```

↳ don't know datatype

```
{
  cout <
  cin >
}
```

If you need element, use for loop (general statement)

```
newTests = tests; //won't work
for (i = 0; i < ARRAY_SIZE; i++)
    newTests[i] = tests[i]
```

↳ copy

```
char fName[] = "Henry";
cout << fName << endl;
```

* String is a class

↳ computer converts to string

```
for (int val : numbers)
```

* ranged based

When using accumulator, initialize to 0.

sum += test[num] = sum = sum + test

Parallel arrays

↳ two or more arrays that contain data

* Program 7-15

Arrays as Funct Argument

showScores(test, Array)

* Proggram 7-17

2D Arrays:

```
const int ROWS = 4, COLS = 3;
int exams[ROWS][COLS];
```

exams[2][2] = third row + column

*Program 7-21

```
int exams[ROWS][COLS] = { {84, 78},
                          {92, 97};
```

getExams(exams, 2);

7.9: Arrays w/ three or more Dimensions

When used as param, specify all but first void ... (short [][43][5]);

7.11 Vector - Intro to the STL vector

Vector library ↳ Standard Template Library

library
^
classes
^
functions
^
data

vector<int> scores;

↳ vector is a class

↳ it's open []

vector<int> scores open

vector<int> scores(30,0) defined

* If running program, do C++ 20

scores.size()

class.function

scores.clear();

while(!scores.empty())...

↳ while not empty

*Program 7-25

*Read how vector is programmed

↳ c++ std vector reading

* Mid Oct 18 or 20 or 11

* Final Dec 20

Q1 pg. 812 13-11

Q2 pg. 458 7-15, 7-23

pg. 434 pg. 457

Q3 pg. 460 7.18. Tic Tac Toe

Lec 4: 9: Pointers

Sept. 27, 2025

Variable Addresses

↳ Addresses in hexadecimal: 0-F

```
cout << &num;
```

Pointer Variables

Pointer variable: var that holds address

↳ points to data

```
showValues(numbers, SIZE)
```
↳ array

```
void getOrder(int &donuts)
{
   cout << "...."
   cin >> donuts;
}

int donutsr = 0;
```

Pointers are more low-level than arrays and reference vars.

Arrays are a contiguous block of memory.

```
int *intptr;
```

intptr can hold address of int

```
int * intptr
int* intptr
```
} all the same

```
int *intptr;
intptr = &num;
*intptr = num;
```

int ptr w/o * is the memory address.

```
int *ptr = nullptr;
```

↳ key C++ term

↳ we say empty, really 0.

Program 9-2

stores address of var in pointer

Indirection Operator

Program 9-3

Arrays + Pointers

```
int vals[] = {4, 7, 11};
```

↳ array is memory address

```
cout << *vals
```

↳ 4, first element

```
int *valptr = vals;
cout << valptr[1];
```

Program 9-5

```
int vals[] = {4, 7, 11}, *valptr;
valptr = vals;
valptr + 1
```

↳ means (address in valptr) + (1 * size of an int)

Array Access

ptr and array are memory

Program 9-7

Pointer Arithmetic

```
int vals[]
valptr* = vals
```

↳ can increment, decrement (++, --)

↳ *(valptr + 2)

↳ valptr = vals; valptr += 2;

↳ valptr - val; (pointer from pointer)

Program 9-9

Initializing Pointers

```
int num, *numptr = &num;
int val[3], *valptr = val;
```

illegal

```
double cost;
int *ptr = &cost;
if (!ptr)
```

test w/ auto

Comparing Pointers

```
if (ptr1 == ptr2) // addresses
if (ptr1* == ptr2*) // values
```

When you define array, also pointer

Pointers as Func. Parameters

9-11

Pointers to Constants

```
void displayPayRates...
int* const ptr = &value
```

points to ↑ ↳ cannot point to anything else.

Dynamic Memory Allocation

```
double *dptr = nullptr;
dptr = new double;
```

raad, random access memory

```
delete fptr;
delete [] fptr; for array
```

9-14

Returning Pointers from Functions

```
int* newNum();
```

9-15

Smart Pointers

↳ don't have to worry about delete

unique_ptr
shared_ptr
weak_ptr
} functions?

```
#include <memory>
unique_ptr<int> ptr(new int);
```

9-17

HW

pg. 484 7-2

pg. 553 9-1

pg. 554 9-5

pg. 554 9-9

pg. 555 9-13

Lec 5: Advanced File Operations

Oct 4, 2025

Nov 29 - no classes
Midterm: Oct 18 or 25 will be midterm
Final: Dec. 20

Files

- ↳ set of data stored
- ↳ read from; write to files

fstream

- ↳ ifstream (input file stream)
- ↳ ofstream
- ↳ fstream
- ↳ >> read from
- ↳ << write to
- ↳ eof (end of file)
  - ↳ boolean value

open call

dFile.open ...

File output formatting

setw(x), showprecision(x)

- ↳ requires iomanip
  - ↳ input/output manipulator

12-3

12-5*

- ↳ pass objects by reference

More Detailed Error Testing

12-6*

- getline function: reads input including
- '\n'
  - ↳ newline or return

Get one char
↳ .get(Lettergrade.csv)

Multiple Files

12-12*

Binary Files
↳ contains unformatted, non-ASCII data
↳ inFile.open("nums.dat", ios::in | ios::out)

Record w/ structures

Random-Access

*object.function

.dat = bin file

HW:
Eg
- 9-10, ~
- ~~12-5~~,
- 12-8, ✓
- 12-12
- 12-21
  ↳ check course materials for file: inventory.dat

Midterm
- Chapter 6 menu driven
- Two Programs

How to Think like a Computer Scientist

PC:
12-1
↳ create our own data files

HW: 12-10*
↳ file decrypter filter

Lec 6: Ch. 14: More about Classes

Oct 11, 2025

Instance and Static Members

- static variable: one var shared among all objects of class.
- static member function: used to access...

14-1

static can only access other static vars

Modified version of Tree.h

↳ line 12

14.2 Friends of Classes

Friend: a fxn or class that is not member of class, but has access to private members of the class

↳ friend can emerge from other classes

↳ friend is keyword

```
friend void setAVal(intVal8, int)
friend void SomeClass::setNum(int num)
```

↳ use public, can this be done private?

Membershipwise Assignment

```
Rectangle r2 = r1;
```

14.5*

14.9 Copy Constructors

↳ default copy constructor copies field-to-field

14.5 Operator Overloading

↳ operator+ to overload + operator

↳ operator = to overload = operator

↳ this C++ keyword

↳ predefined pointer available to a class's member fxn

Notes on Overloaded Operators
↳ cannot change operands of operator

pg. 837 in textbook onward

*Version 3 of Student test scores
↳ v1 is not overloaded
↳ STUDENTSTESTSCORE.h

14-6

keyword : using
↳ look up!

Aggregation
↳ supports the modeling of 'has a' relationship btwn classes

14.10 Rvalue References + Temp Values
↳ lvalues persist beyond statement
↳ rvalues are temporary, cannot be accessed beyond statement
↳ use to move data outside class

14-7*

Midterm: Oct. 25 *

6-10*
↳ part of exam
↳ mean driven

In-class HW:
12-3
14-6
14-7
12-11

Out-of class HW:
6-10

Lec 7:

Oct 18, 2025 File questions

Must draw the header file ↳ Comments on every line

* ↳ Draw the UML digram

Questions

↳ 14-11 Notepad++

↳ 14-13 Bring paper and pen

Rectangle
- width : double - length : double
+ setWidth(w : double) : void + setLength(len : double) : void + getWidth() : double + getLength() : double + getArea() : double

FeetInches

- feet : int
- inches : int
- simplify() : void

+ FeetInches(f:int=0, i:int=0)
+ FeetInches(obj : FeetInches)
+ multiply(obj : FeetInches) : FeetInches
+ setFeet(f: int)
+ setInches(i : int, simplify() : void)
+ getFeet(feet : constant int)
+ getInches(inches : constant int)
+