CompilationProcess: compile the source file u wrote into an object file and header file into library file and combined them into an executable file.**Namespace:** C++ segments code into structures called namespaces, each of which keeps track of its own set of names to avoid name collisions, using namespace x : an unqualified identifier might come from namespace x.**Data Types:** defined by two properties, **Domain:** the set of values that belong to that type. **ASetOfOperation:** the behavior of that type. **Variable:** is a named address for storing a type of value. **Name conventions:** start with a letter of the underscore, can only contain letters, digits, or the underscore. Scope: where in a program's text the variable may be used. Extent/Lifetime: when in a program's execution a variable has a meaningful value. **Local variables:** defined in the body of a function definition, even variables defined in main() is still local. **Binary/Unary** operator: how many operands it takes. **Associativity**. **L-Value**: has an address, can appear on both sides of assignments. **R-Value:** no address, cannot appear on the left side of assignments.**Short-circuit mode**: it evaluates the right operand only if it is need to. **Switch:** break is used to get out of the switch statement otherwise it will execute all the following cases. **Function** is a named section of code that performs a specific set of statements.**1,Reuse2,abstraction3,Decomposition Parameters** is a placeholder for one of the arguments supplied in the function call. **Predicate functions:** functions that return Boolean results are called predicate functions. **Procedure functions**: functions that return no value. **All functions need to be declared before they are called by specifying a prototype consisting of the header line followed by a semicolon.** Prototypes are not required if you always define functions before u call them. **Signature**: the pattern of arguments (the number and types of the arguments but not the parameter names) required for a particular function is called. **Overloading:** u can define several different functions with the same name as long as the correct version can be determined by looking at the signature. **Default Values**: the specification of the default values appears only in the function prototype and not in the function definition. **Reference variable** is another name for an already existing variable. Ie. Int & x = n1. ==CallByReference. **Decomposition:** functions simplify program maintenance by allowing you to divide a large program into smaller pieces. **Stack frame** is a newly assigned region of memory by C++ when a function is invoked. **Interface** is where the client and the implementation meet. For sharing and reuse/ abstraction and hiding.

String

C String: a sequence of characters, arrays of elements of character type. C++ views string as objects. Character: fits in a single eight-bit byte because the use of ASCII, if u assign a number to char type, this will give u the letter using ASCII. **C String literal:** const char[] type, the characters are stored in an array of bytes, terminated by a **null byte** whose ASCII value is 0. "a" is a string literal containing an 'a' and a null terminator '\0', which is a 2-char array. Wheras 'a' is a 1-char array. **C++ String Objects. C++** allows clients to change the individual characters contained in a string. The **+ operators** cannot be applied to C string literals, can only be applied to C++ String Objects. But <<, the insertion operator can be used for C string literals. #include <string>: C++ library; <cstring>: C string library used in C++; <string.h>: C string library used in C; "string.h": C string library; "cstring.h": error unless defined yourself. string str="nb"+", "+string("cp"); //Flase, quoted strings are C strings, but + can only be applied to C++. When compiler can determine that what you want is a C++ string , convert C string literals into C++ string object automatically. **Str.at(index)** will check the boundary.

Streams

Insertion operator (<<): takes an output stream on the left and an expression of any type on its right. Returns the output stream as its value. **Extraction Operator (>>).** Reads formatted data from the stream on the left into the variables that appear on the right. **Text Files vs Strings.** 1, the information stored in a file is **permanent,** the value of a string variable persists only if the variable does. 2, Data in files are usually accessed **sequentially. read data from a text file:**1)construct ifstream: ifstream stream; stream.open(filename)

2) call open (filename must be a c string iteral): open(filename.c_str())

3)read data: get or **getline: store** the next line of data from the file into the string variable after the EOL.

.fail() –> detect "open" success or not **stream.get()** –> returns the next character value as an int, which is EOF(-1) at the end of the file **stream.get(ch)** –> reads the next char into that variable,retrun bool value **getline(stream, string)** – free function, both two parameters **are taken by reference**, sets fail indicator when read past end. 4)close file –>stream.close(). **<< >>** return the stream argument by **Reference.** 1, enable chaining behavior. 2, avoid copy: the **stream variable cannot be copied**, the **ostream** argument must be **passed by reference. Stream Hierarchy:** ios-(istream, ostream). Istream-(ifstream, istringstream). Ostream-(ofstream, ostringstream).

Collection

ADT: primitive data types: bool, char, int, double and the enumerated types occupy the lowest level. Defined by its **behavior rather than representation** is called ADT. **Collection classes:** contain collection of other objects. Advantage: **Simplicity, Flexibility and Security.** for Stanford collection class: 1, each class (except Lexicon) requires type parameters. 2, Any memory for these is **freed automatically.** 3,usually passed by **reference** to avoid copy. **Vector:** using array and is similar to list in python. Stanford: #include "vector.h" **Vector**<type> vec; **Grid:** 2d vector. **Stack: LIFO. Queue: FIFO. MAP:** the type for the **keys** stored must can be **Compared and Ordered. Iterator over a collection.:** range-based for statement: for (string key : map). **Grid**: row-major order. Set and map will use the defined compare. **Lexicon:** always in alphabetical order. **Set:** 1, unordered. 2, no duplicate.3, can be viewed as keys in a map.4, s1 + s2 will return the union. S1 * s2 will return the intersection. **Lexicon** a set of words with no associated definitions. **Extremely space efficient.**

Designing classes

Structure: C styles, compound values, the individual components are specified by name. struct typename{declarations of fileds}; the definition creates a **type not a variable.** Use the **dot operator** (pt.x) to get **fields or members. Classes**: the creation of new instances is called **instantiation.** Class typename{public: available for clients of the class. While protected section can be accessed by subclasses. Prototypes of public methods; private: declarations of private instance variabls and prototypes of private methods.}; **Getters/Accessors:** methods that retrieve the values of instance variables. **Mutators/Setters:** methods that set the values of specific instance variables. **Constructors:** same name as the class, **no return .** Can have **multiple** constructors as long as they have different parameter sequences------**Default constructor:** takes no argument. **Initializer List**: constant members can only be initialized using the initialize list. **Methods:**Point::point(){}. **Overloading:** to rewrite the existing operator. 1, Defining an operator as a **class method:** the operator is part of the class and has *free access* to the *private* instance variables and methods. 2, Defining an operator as a **free function**: produces code that is easier to read but means that the operator function must be designated as a **friend** to refer to the *private data*.

Recursive:

The Minimax Algorithm: min the max rating available to your opponent.

Sorting complexity	Best	Averag e	Worst
Selection sort	N^2	N^2	N^2
Insertion sort	N	N^2	N^2
Bubble sort	N	N^2	N^2
Merge sort	$N \log N$	$N \log N$	$N \log N$
Quicksort	$N \log N$	$N \log N$	N^2

in practice, insertion sort is the fastest simple sorting algorithm.

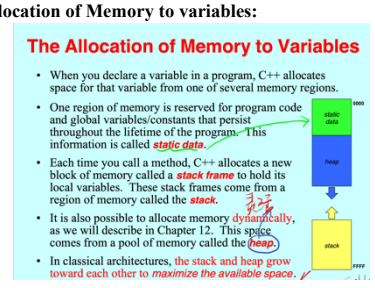
Pointer and Array:

- The memory space required to represent a value depends on the type of value. Although the C++ standard actually allows compilers some flexibility, the following sizes are typical:

1 byte (8 bits)	2 bytes (16 bits)	4 bytes (32 bits)	8 bytes (64 bits)	16 bytes (128 bits)
char bool	short	int float	long double	long double

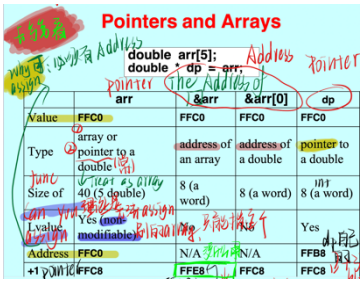
- Enumerated types are typically assigned the space of an **int**.
- Structure types have a size equal to the sum of their fields.
- Arrays take up the element size times the number of elements.
- Pointers take up the space needed to hold an address, which is usually the size of a hardware word, e.g., 4 bytes on a 32-bit machine and 8 bytes on a 64-bit machine.
- **sizeof(t)** returns the actual number of bytes required to store a value of the type **t**; **sizeof x** returns the actual memory size of the variable **x**.

Min addressing unit: 1 **byte** = 8 bits.**Word**: unit that represents the most common integer size on a particular hardware. **The largest addres lenth:** is typically a word, allows one memory address to be efficiently stored in one word. **The allocation of Memory to variables:**

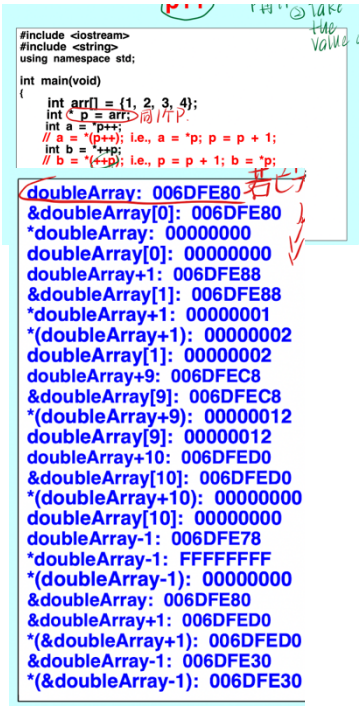


The **address** and **type** of a named variable are **fixed. Lvalue:** any expression that refers to an

internal memory location capable of storing data. If you cannot assign a value to it, it is not a lvalue. The address is a value of a pointer type. **Pointer:** whose value is **an address in memory**. 1, pointers allow u to refer to a large data structure in a compact way, saves the space. 2, reserve new memory during program execution. 3, record relationships among data items. **Pointer declaration:** type* pt. **Pointer Operators: &-the address of :** written before any **lvalue**, will return the address of that variable. ***-the value-pointed-to: Dereferencing**: written before a **pointer** and return the actual value of a variable to which the pointer points. Pt->getX(); use -> for pointer. **Array:** 1) only operation is []. 2) array selection does not check the index is in range. 3) length is fixed and array don't store its actual length. The **name of the array is treated as a pointer.** Int list[100]; compiler treats the name **list** as a pointer to the address &list[0] whenever necessary, and list[i] is just *(list+i). difference: **an array is a non-modifiable lvalue.** Pass an array === call by pointer. ***P++:** is equivalent to



*(p++), means deference p and return as an lvalue the object to which it currently points, and increment the value of p so that the new p points to the next element in the array. You can




```
doublePointer: 006DFE80
doublePointer+1: 006DFE88
&doublePointer: 006DFE7C
&doublePointer+1: 006DFE80
doublePointer[0]: 00000000
doublePointer[1]: 00000002
doublePointer[9]: 00000012
doublePointer[10]: 00000000
&doublePointer[0]: 006DFE80
&doublePointer[1]: 006DFE88
&doublePointer[9]: 006DFEC8
&doublePointer[10]: 006DFED0
*doublePointer: 00000000
*doublePointer+1: 00000001
*(doublePointer+1): 00000002
*doublePointer++: 00000000
*++doublePointer: 00000004
```

still add more graph here since it is the most importanrt thing.

Dynamic Memory Management

new operator to allocate memory on the heap. if no new operator, the variable will only be stored in the stack and will be delete after the function terminated. **delete** operator frees memory previously allocated, but the pointer still stores that address. Eg, delete [] arr; delete pi; the pointher is a **dangling** pointer, and u can **pt = NULL//nullptr** to nullify the pointer. **Destructor:** how to free the storage used to represent an instance of that class. **Copying Objects:** 2 methods to implement this . 1, **opeartor=, which takes care of assignments.** Typically **return a reference to the left-hand operand, i**n order to chain assignments together. **2, copy constructor, which** takes care of by-value parameters. **Const:** 1, **Constant definitins**: tells the compiler to

```
const type & type::operator=(const type & rhs)
type::type(const type & rhs)
```

disallow subsequent assignments to that variable. **Constant call by reference:** the function will not change the value of that parameter. Share the contents without allowing methods to change it. **Constant methods:** the methods will not change the object.

Efficiency:

1, Array model: simplets, allocate the array storage dynamically and to expand the array when full. **Allocated size is capacity and effective size is count.** 2, Linked-list model: **Dummy cell**: an extra cell at the beginning so you can find the right position for cursor. 3, two-stack model: the characters before the cursor are stored in a stack called **before** and the characters after the cursor are stored in a stack called **after.**

		array	list	stack
F	moveCursorForward()	O(1)	O(1)	O(1)
B	moveCursorBackward()	O(1)	O(N)	O(1)
J	moveCursorToStart()	O(1)	O(1)	O(N)
E	moveCursorToEnd()	O(1)	O(N)	O(N)
I	insertCharacter(ch)	O(N)	O(1)	O(1)
D	deleteCharacter()	O(N)	O(1)	O(1)

Overloading: several functions with the same name as long as those functions can be distinguished by their signatures. **Templates:** to automate the overloading process. template<typename *placeholder*> specify before each function. **Hint:** the implementations of the methods need to be included in the header file. **Array-based queue**:1, use front and rear pointer. 2, use ring buffer to reuse the memory. **Return by reference:** means putting the result into an lvalue, another name. never use return by reference **with a value that lives in the stack frame of the current method**, use **new.**

Hash: Collision: have the same hash code. **Bucket hashing:** use hash code for each key to select an index into an array, each element in that array is called bucket, use ll as bucket to avoid collision. **Load Factor:** the ratio of the number of keys to the number of buckets, the map will achieve **O(1) performance** only if the load factor is small. **Rehash:** increase the number of buckets.

Tree:

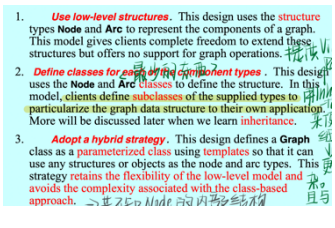
BST: get and put is O(logN). **Heap:** is an array-based data structure that simulates the operation of a **partially ordered tree** (1, complete. 2, root node has higher priority).use it to implement the **Priority Queue.** Parentindex(n): (n-1)/2. Leftchildindex(n):2n+1. Rightchildindex(n): 2n + 2. **Heapsort**: is like selection sort, but the unsorted region is stored in heap. worst-case O(NlogN).

Sets:

Implementing sets: 1, Hash tables: offering average O(1) performance for adding new elements but **do not support ordered iteration.** 2, **Balanced binary trees:** O(logN) performacne and make it possible to write and **ordered** iterator. **Bitwise Operators:** (~)NOT, (&)AND, (|)OR, (x << n) shift the bits in x left n positions. 1, x is unsigned bits, logical shift in which missing digits are always filled with 0. 2, x is signed type, arithmatic shift, the leading bit in the value of x never changes.

Graph:

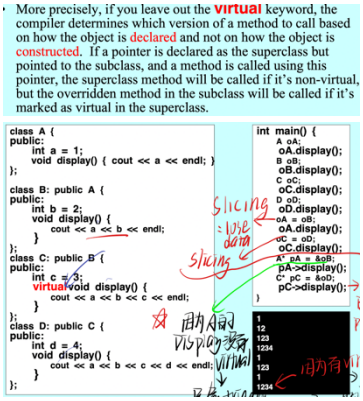
Degree: the number of connections from a node. Out-degree and in-degree for directed



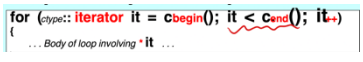
graph. **Nodes and arcs contain** data required by the client along with data required by the implementation of the graph. 2, use inheritance to define your nodes and arcs.

Inheritance:

class subclass : public supercalss{}; **public inheritance:** makes public members of the superclass public in the subclass. **Protected inheritance:** makes the public and protected members of the superclass protected in the subclass. **Private inheritance:** the public and protected members private. **Overriding:** same signature but different implementations defined in different classes.—dynamic polymorphism. **Virtual method:** a method that is overridden differently in each subclass. virtual double func1()=0; is a pure virtual method. **Abstract class:** a class with virtual methods. One cannot decalre an object of an abstract class,but can **declare a pointer of an abstract class. Slicing:** assign an object of a subclass to a superclass variable, c++ will throws away sth. Use private copy to avoid. **Do not get a collection of**



different class, use a collection of pointers!. collection class are independent class while stream classes don't allow hierachy. **Initializer list:** when u called the constructor for an object, the constructor will call the default constrcutor in superclass, but you can use IL to call another.



Mapping func: to each element in turn.

Function pointer

double *g(); Declare g as a function returning a pointer to a double. double (*fn)(double): take/return a double. **Functional, Object-**

Callback Functions

`double x;`	*Declares* `x` *as a* `double`.
`double list[n];`	*Declares* `list` *as an array of* `n` `doubles`.
`double *px;`	*Declares* `px` *as a pointer to a* `double`.
`double **ppx;`	*Declares* `ppx` *as a pointer to a pointer to a* `double`
`double f(double);`	*Declares* `f` *as a function taking and returning a* `double`.
`double *g(double);`	*Declares* `g` *as a function taking a* `double` *and returning a pointer to a* `double`.
`double (*fn)(double);`	*Declares* `fn` *as a pointer to a function taking and returning a* `double`.

oriented(encapsulation, inherientence, polymorphism, Procedural.