LAB 5 – INTRO TO C++

CLASS VS INSTANTIATE VS OBJECT

- Class
  - Noun
  - The blueprint, or an outline which includes behavior and structure
- Instantiate
  - Verb
  - The process of creating an object of a class
    - Allocate memory (on stack or heap)
- Object
  - Noun
  - The instance of a class within memory (either on stack or heap)

CONSTRUCTORS

- The place where we setup our class's instance
- There can be 0 or more parameters
- Always exactly one constructor

DESTRUCTORS

- The place where we tear down our class's instance
- Always exactly one destructor
- Can be virtual
- This is where we want to free anything that was allocated within the instance

PUBLIC, PRIVATE, PROTECTED, FRIEND

- public:
  - Everybody can access it
- protected:
  - Only classes who inherit from the class can access it
- private:
  - Only itself can access it
- friend:
  - Allow a given class or function private access to a given class
  - `friend class OtherClass; // Friend a class`
  - `friend void ClassB::funcName(double& d); // Friend a member function`
  - `friend void nonMemberFunc(int& i); // Friend a non-member function`

FRIEND EX 1

Avoid using friend when possible

```
class Y
{
    int data; // private member

    // the non-member function operator<< will have access to Y's private members
    friend std::ostream& operator<<(std::ostream& out, const Y& o);
    friend char* X::foo(int); // members of other classes can be friends too
    friend X::X(char), X::~X(); // constructors and destructors can be friends
};

// friend declaration does not declare a member function
// this operator<< still needs to be defined, as a non-member
std::ostream& operator<<(std::ostream& out, const Y& y)
{
    return out << y.data; // can access private member Y::data
}
```

https://en.cppreference.com/w/cpp/language/friend

FRIEND EX 2

Avoid using friend when possible

```
class X
{
    int a;

    friend void friend_set(X& p, int i)
    {
        p.a = i; // this is a non-member function
    }
public:
    void member_set(int i)
    {
        a = i; // this is a member function
    }
};
```

https://en.cppreference.com/w/cpp/language/friend

METHOD VS FUNCTION

- A **related member** is one defined **in** a class
- A **non-related member** is one defined **outside** of a class
- A method is a related member
  - It is defined in the class
- A function is a non-related member
  - It is defined elsewhere of the class

OVERLOADED FUNCTIONS

- An overloaded function has multiple different parameter types with the same name

- int add(int, int)
- double add(double, double)
- string myConcat(string, string)
- char * myConact(char *, char *)

OVERLOADED OPERATOR

- Complex operator+ (const Complex &lhs, const Complex &rhs);
- In this example (1.a.iii), we have an operator which accepts a Complex class for the left and right side of the + operator. It then returns a Complex class.
- Suppose we implement an add function (1.a.ii) in Complex. What does our function look like for the prototype given above?
- Important note: The parameters of the operator are constant and passed by reference!

PASS-BY- VALUE VS REFERENCE

- Pass by value
  - Pass in the value directly, as we have in C
  - Same with pointers, we copy the value of the address to the callee

- Pass by reference
  - Pass in the value indirectly. Not as we use pointers
  - When we modify the value in the function, it acts as if we dereference a pointer
    - (By Reference) We need to: x = 100;
    - (With pointers) We need to: *x = 100;
    - Both of the above modify the source variable

DYNAMIC MEMORY: GOODBYE MALLOC

```
size_t count = 100;


int * myNumber = new int;
int myNumbers[count]; // Note: We can now use non-constant values to set an array size. You can count the number of elements you need before giving it a size. Not resizable, so it is not dynamic memory, but is a feature of C++.
int * myNumbersDyn = new int[count];


delete myNumber; // if you use new TYPE
delete[] myNumbersDyn; // if you use new TYPE[COUNT]
```

USING STATEMENTS

```
#include <iostream>
#include <string>
#include <vector>
using std::cout;
using std::endl;
using std::string;
using std::vector;

// You can now use cout instead of std::cout
// You can now use endl instead of std::endl
// You can now use string instead of std::string

Using namespace std; // <<< You will lose 5 points in PAs
```

HPP (THE PLAN/OUTLINE OF A CLASS)

```
class PrivateProtectedPublic {
private:
    int privInteger;
    int privFunc(void);

protected:
    double protDouble;
    double protFunc(void);

public:
    char pubChar;
    char pubFunc(void);

};
```

CPP (THE IMPLEMENTATION OF PLANS)

```
int PrivateProtectedPublic::privFunc() {
    return this->privInteger;
}
double PrivateProtectedPublic::protFunc() {
    return this->protDouble;
}
char PrivateProtectedPublic::pubFunc() {
    return this->pubChar;
}
```

FUNCTIONS FOR LAB 5 FA 24

- What do the following functions do?
  - memcmp
  - memcpy
  - memset
  - memchr

LAB 5 – FA 24

```
MEMSET(3)                 Library Functions Manual                 MEMSET(3)

NAME
     memset - fill a byte string with a byte value
LIBRARY
     Standard C Library (libc, -lc)
SYNOPSIS
     #include <string.h>
     void *
     memset(void *b, int c, size_t len);
DESCRIPTION
     The memset() function writes len bytes of value c (converted to an
     unsigned char) to the string b.
RETURN VALUES
     The memset() function returns its first argument.
SEE ALSO
     bzero(3), memset_pattern(3), memset_s.3, swab(3), wmemset(3)
STANDARDS
     The memset() function conforms to ISO/IEC 9899:1990 (“ISO C90”).


macOS 15.0                     June 4, 1993                     macOS 15.0
(END)
```

LAB 5 – FA 24

```
MEMCPY(3)                Library Functions Manual                MEMCPY(3)

NAME
     memcpy – copy memory area
LIBRARY
     Standard C Library (libc, -lc)
SYNOPSIS
     #include <string.h>
     void *
     memcpy(void *restrict dst, const void *restrict src, size_t n);
DESCRIPTION
     The memcpy() function copies n bytes from memory area src to memory area
     dst.  If dst and src overlap, behavior is undefined.  Applications in
     which dst and src might overlap should use memmove(3) instead.
RETURN VALUES
     The memcpy() function returns the original value of dst.
SEE ALSO
     bcopy(3), memccpy(3), memmove(3), strcpy(3), wmemcpy(3)
STANDARDS
     The memcpy() function conforms to ISO/IEC 9899:1990 (“ISO C90”).

macOS 15.0                     June 4, 1993                     macOS 15.0
(END)
```

LAB 5 – FA 24

NAME

memcmp – compare byte string

LIBRARY

Standard C Library (libc, -lc)

SYNOPSIS

```
#include <string.h>
int
memcmp(const void *s1, const void *s2, size_t n);
```

DESCRIPTION

The **memcmp**() function compares byte string s1 against byte string s2. Both strings are assumed to be n bytes long.

RETURN VALUES

The **memcmp**() function returns zero if the two strings are identical, otherwise returns the difference between the first two differing bytes (treated as unsigned char values, so that '\200' is greater than '\0', for example). Zero-length strings are always identical. This behavior is not required by C and portable code should only depend on the sign of the returned value.

SEE ALSO

bcmp(3), strcasecmp(3), strcmp(3), strcoll(3), strxfrm(3), wmemcmp(3)

STANDARDS

The **memcmp**() function conforms to ISO/IEC 9899:1990 ("ISO C90").

LAB 5 – FA 24

MEMCHR(3) Library Functions Manual MEMCHR(3)

NAME

memchr – locate byte in byte string

LIBRARY

Standard C Library (libc, -lc)

SYNOPSIS

```
#include <string.h>
void *
memchr(const void *s, int c, size_t n);
```

DESCRIPTION

The **memchr**() function locates the first occurrence of c (converted to an unsigned char) in string s.

RETURN VALUES

The **memchr**() function returns a pointer to the byte located, or NULL if no such byte exists within n bytes.

SEE ALSO

strchr(3), strcspn(3), strpbrk(3), strrchr(3), strsep(3), strspn(3), strstr(3), strtok(3), wmemchr(3)

STANDARDS

The **memchr**() function conforms to ISO/IEC 9899:1990 (“ISO C90”).

macOS 15.0 April 9, 2008 macOS 15.0

(END)

WHAT THIS MEANS:

- Variables are:
  - realPart
  - imaginaryPart
- Operations are:
  - Read
  - Print
  - Add
  - Sub

- Should not take more than 5 min
- All we need to do is design the class

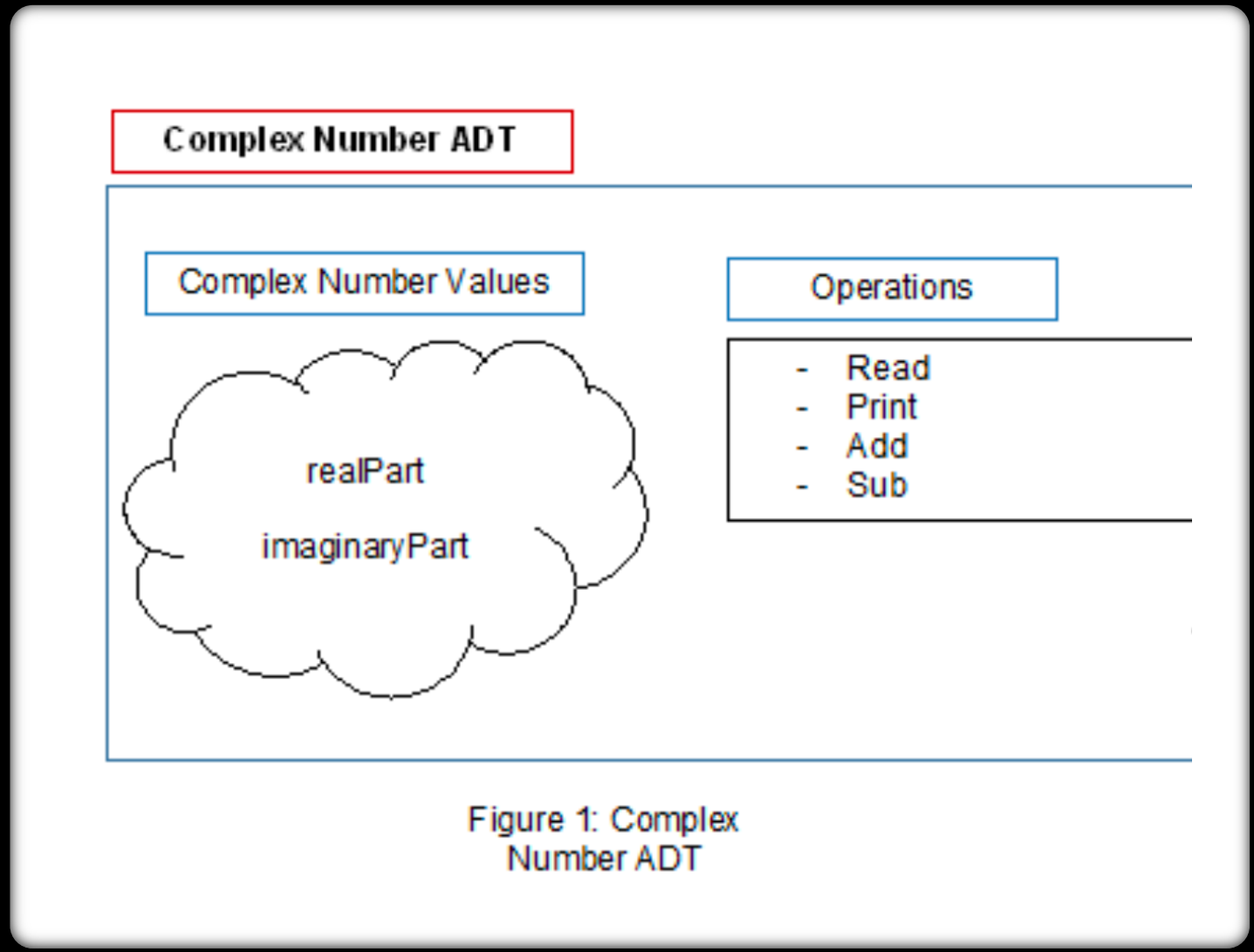

Figure 1: Complex Number ADT

SIMPLE: IMPLEMENT THE ADD FUNCITON

a) Adding two *Complex* numbers: The real parts are added together and the imaginary parts are added together. Implement the addition operation in three ways:
   i. Implement a *member* function of class *Complex* called `add()` that must do the following: accept one *Complex* number *rhs* for an argument, add *rhs* to the data members in the object that invokes the `add()` function, and return the result.

HARDER, BUT NOT TOO DIFFICULT. WE ARE SIMPLY CALLING THE PREVIOUS FUNC

ii. Implement a *non-member* function called `add()`. Make sure it has the same name as the one defined in part (i). Place the prototype/declaration for this function outside of the *Complex* number class's declaration (below the class declaration). Place its definition inside of the Complex.cpp file. The function must do the following:

VERY SIMPLE: USE THE PROTOTYPE BELOW

iii. Implement a *non-member* overloaded addition (+) *operator*. Place the prototype/declaration for this function outside of the *Complex* number class's declaration (below the class declaration). Place its definition inside of the Complex.cpp file. The function must do the following: accept two *Complex* numbers called *lhs* and *rhs* for arguments, add *lhs* and *rhs* together, and return the result. Note: the overloaded + is a binary operation, which requires two arguments! Use the following prototype/declaration:

```
Complex operator+ (const Complex &lhs, const Complex &rhs);
```

RESOURCES

- https://en.cppreference.com/w/cpp/language/friend
- https://learn.microsoft.com/en-us/cpp/cpp/friend-cpp?view=msvc-170
- https://en.cppreference.com/w/cpp/language/class
- https://cplusplus.com/doc/tutorial/classes/
- https://cplusplus.com/doc/tutorial/dynamic/
- https://en.cppreference.com/w/cpp/memory
- https://cplusplus.com/doc/tutorial/classes/
- https://cplusplus.com/doc/tutorial/
- https://en.cppreference.com/w/cpp