- C C ++ Structured
  - Comments
    - Single line
      - //
    - Multi-line
      - /* */
  - Variable initialization
    - C
      - int a = 5
    - C++
      - int a = 5;
      - int a (5);
      - int a {7};
  - Input/Output
    - C
      - Library
        - #include <stdio.h>
      - Output
        - printf();
      - Input
        - scanf("%d", &a);
    - C++
      - Library
        - #include <iostream>
      - Output
        - std::cout
        - <<
          - Insertion operator
        - Example
          - std::cout << "Hello world!";
          - std::cout << "hello" << "world";
          - std::cout << 4;
          - std:cout << "value : " << 4
      - End a line
        - with endl
          - std::endl
          - std::out << "hello!" << std::endl << "world!";
        - with '\n'
          - \n
          - std::out << "hello!" << '\n' << "world!";
      - Input
        - std::cin
        - >>
          - Extraction operator
        - example

        - std::cin >> input_var
        - std::cin >> input_var1 >> input_var2
    - note: The C++ I/O library does not provide a way to accept keyboard input without the user having to press enter. If this is something you desire, you'll have to use a third party library. For console applications, we'd recommend the pdcurses library. Many graphical user libraries have their own functions to do this kind of thing.
- C/C++ Keywords
    -

alignas	const_cast	int	static_assert
alignof	continue	long	static_cast
and	co_await (since C++20)	mutable	struct
and_eq	co_return (since C++20)	namespace	switch
asm	co_yield (since C++20)	new	template
auto	decltype	noexcept	this
bitand	default	not	thread_local
bitor	delete	not_eq	throw
bool	do	nullptr	true
break	double	operator	try
case	dynamic_cast	or	typedef
catch	else	or_eq	typeid
char	enum	private	typename
char8_t (since C++20)	explicit	protected	union
char16_t	export	public	unsigned
char32_t	extern	register	using
class	false	reinterpret_cast	virtual
compl	float	requires (since C++20)	void
concept (since C++20)	for	return	volatile
const	friend	short	wchar_t
consteval (since C++20)	goto	signed	while
constexpr	if	sizeof	xor
constinit (since C++20)	inline	static	xor_eq

- Codes with multiple files C/C++
    -

```cpp
#include <iostream>

int add(int x, int y); // needed so main.cpp knows that add() is a function declared elsewhere

int main()
{
    std::cout << "The sum of 3 and 4 is: " << add(3, 4) << '\n';
    return 0;
}
```

add.cpp (stays the same):

```cpp
int add(int x, int y)
{
    return x + y;
}
```

- Data type sizes C/C++

Category	Type	Minimum Size	Note
boolean	bool	1 byte	
character	char	1 byte	Always exactly 1 byte
	wchar_t	1 byte	
	char16_t	2 bytes	
	char32_t	4 bytes	
integer	short	2 bytes	
	int	2 bytes	
	long	4 bytes	
	long long	8 bytes	
floating point	float	4 bytes	
	double	8 bytes	
	long double	8 bytes	

- C++ adds wchar_t
- to find find size
  - sizeof(data_type);
  - example
    - std::cout >> sizeof(data_type);
- Fixed width integers
  - #include <cstdint>
  -

Name	Type	Range
std::int8_t	1 byte signed	-128 to 127
std::uint8_t	1 byte unsigned	0 to 255
std::int16_t	2 byte signed	-32,768 to 32,767
std::uint16_t	2 byte unsigned	0 to 65,535
std::int32_t	4 byte signed	-2,147,483,648 to 2,147,483,647
std::uint32_t	4 byte unsigned	0 to 4,294,967,295
std::int64_t	8 byte signed	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
std::uint64_t	8 byte unsigned	0 to 18,446,744,073,709,551,615

Access modifiers C/C++

- Constant
  - Keyword
    - **const**
  - Saves variable in to ROM
  - Unchangeable
- Volatile
  - Keyword
    - **volatile**
  - The volatile keyword is intended to prevent the compiler from applying any **optimizations** on objects that can change in ways that cannot be determined by the compiler.
  - Why use
    - Global variables modified by an interrupt service routine outside the scope
    - Global variables within a multi-threaded application

- Storage class specifiers C/C++
  - C/C++
    - extern
      - When using multiple files, a global variable can be redeclared and cause it not to take storage space. The compiler just finds the original declaration of the global variable and uses that.
      - file 1
        - int x;
      - file 2
        - extern int x;
      - The extern file does not need definition as it uses the original variable's definition.
      - Extern can also be used in functions.
        - When you use a global variable in a function, you can use extern.
    - static
      - C
        - Static local
          - Both does the same thing
        - Static global
          - Both does the same thing
      - C++
        - Can be used but deprecated
    - register
      - Keyword
        - register <var declaration>
      - Used to store a variable into the cpu register and not in memory
      - Cannot be declared in global scope
      - Only Char and Int can be stored.
      - C
        - Address of the register variable can not be found, so pointers will not work.
      - C++
        - Addresses can be found but will not optimize the code.
    - auto
  - C++
    - mutable
- Hex and oct constants C/C++
  - Hex and octal variables can be used using int
  - Hex values have to start with 0x
    - int a = 0xFF;
  - Octal values have to start with 0
    - int a = 012;

- Backslash characters C/C++
  - To add “ in a string
    - a = “my\”name\” is”;
  - Escape characters

\a	07	Alert (Beep, Bell) (added in C89)[1]
\b	08	Backspace
\e	1B	Escape character
\f	0C	Form feed Page Break
\n	0A	Newline (Line Feed); see notes below
\r	0D	Carriage Return
\t	09	Horizontal Tab
\v	0B	Vertical Tab
\\	5C	Backslash
\'	27	Apostrophe or single quotation mark
\"	22	Double quotation mark
\?	3F	Question mark (used to avoid trigraphs)

- Type conversion in assignment C/C++
  - int x;
  - char ch;
  - float f;
  - void func(){
    - ch = x; //int is converted to char
    - x = f; //float value is int (x get non faction)
    - f = ch; //char is now float
    - f = x; //int is now float
  - }
- Multiple assignments C/C++
  - x = y = z = 0
- Increment / decrement C/C++
  - x++
    - increases value before next statement
  - ++x
    - increases value right away
  - --
- Logical operator precedence
  - Highest to lowest
    - !
    - > >= < <=
    - == !=
    - &&
    - ||
- Bitwise operators C/C++
  - Operates on a variable in the bit level.
    - & //AND
    - | //OR
    - ^ //XOR 0 0=0, 0 1=1, 1 0=1, 1 1 = 0

        - ~ //NOT
        - >> //Shift right
        - << //Shift left
- The ? operator C/C++
    - Keyword
        - exp1?exp2:exp3
        - if exp1 is true then exp2
        - if exp1 is false then esp3
- The comma operator C/C++
    - x = (y=3, y+1);
        - left side is always void
        - x is now 4
- Switch case C/C++
    - switch(expression):
        - case constant1:
            - //
            - break;
        - case constant 2:
            - //
            - break;
        - default
            - //optional, if no case matches
- Do while loop C/C++
    - Do while loop checks condition at the bottom
    - So the code inside is executed at least once
    - form
        - do{
            - //;
        - }while(condition);
- Go to statement C/C++
    - goto label;
    - label:
    - example
        - x = 1;
        - loop1:
            - x++;
            - if(x<100) goto loop1;
- Break and continue C/C++
    - break;
        - stops the loop and exits
    - continue;
        - skips a loop
- Null terminated string C
    - Known as C string
    - char str[11]
        - 1 element of the char array is used for null termination.

- Pointers C/C++
  - Points to the memory address of the object
  - int *p = &num
    - & refers to “the address of”
    - * returns the value of the variable
  - pointer arithmetic
    - All pointer arithmetic is relative to its base type.
    - uint8_t *p = 1; uint8_t is 1 byte long
    - p++;
      - will be 2
    - uint16 *q = 1;
    - q++;
      - q will be 3, because uint16_t is 2 bytes long.
  - Multiple indirection
    - Pointer to a pointer (newbalance)
    - int **p //is a pointer to a pointer
    - to access **p use **p
  - Returning function pointers
    - if a function returns pointers then the function has to have *.
    - example
      - char *match(char c, char *s){
        - //
        - return s;
      - }
      - char *p = match(ch, s);
  - C++
    - Pointer types have to be the same.
- Reference parameters
  - C
    - void neg (int *i){
      - *i = -*i;
    - }
    - int j = 1;
    - neg(&j);
  - C++
    - void neg(int &i){
      - i = -i;
    - }
    - int j = 1;
    - neg(j);
- Function prototypes C/C++
  - C
    - when function has no parameters void is required
    - void fun(void);

- C++
    - void fun(); and void fun(void); is same

Structures C/C++

- Example
    - struct books{
        - char book_name;
        - int book_number;
    - }book1, book2; //optional
- member access
    - book1.book_name = “abc”;
- Declaration
    - stuck books book1;
- Array of structure
    - structures can be declared as arrays, so that each array element will be a whole structure
    - example
        - struct books book_array[100]
- Passing structures in functions
    - passing a member
        - void test(int book_numb){
            - //
        - }
        - int main(void){
            - struct books book;
            - test(book.book_num);
        - }
    - Passing entire structure
        - void test(struct books book1){
            - //
        - }
        - int main(void){
            - struct books book;
            - test(book);
        - }
        - make sure to pass same type structures
    - Structure pointer
        - struct books book; //struct declaration
        - struct books *book_p; //books type pointer.
        - *book_p = &book;
        - to access members
            - book_p->book_name;
    - Structure bit fields
        - struct books{
            - char book_name:3; //assigned 3 bits
            - int book_number:2; //assigned 2 bits
        - }book1, book2;

- Unions C/C++
  - Are like structures but the objects inside share the same memory.
  - union books{
    - char book_name:3; //assigned 3 bits
    - int book_number:2; //assigned 2 bits
  - }book1, book2;
- Enum C/C++
  - enum switch_states{
    - on,
    - off
  - } switch_1;
- Typedef C/C++
  - rename a type
  - typedef float deci;
  - deci a_float_type_var = 3.14;
- Dynamic memory allocation
  - C/C++
    - malloc
    - calloc
    - free
  - C++
    - Always use the try catch when using new as no/low free memory will give exceptions.
    - new
      - library
        - #include <new>
      - syntax
        - pointer_var = new data_type;
      - Initialization
        - pointer_val = new data_type(value);
      - *Array init*
        - pointer_var = new array_type[size];
      - *Allocating objects*
        - With constructors
          - pointer_var = new class_name (parameters);
        - Without constructors
          - point_var = new class_name;
    - delete
      - library
        - #include <new>
      - syntax
        - delete pointer_var;
      - Delete array var
        - delete [] pointer_var;

- example
  - main
    - int *p
    - try{
      - p = new int //allocate memory
    - }
    - catch (bad_alloc xa){
      - //
      - return 1;
    - }
    - *p = 100;
    - delete p;

- C++ OOP
  - Principles of OOP
    - Inheritance
    - Encapsulation
    - Abstraction
    - Polymorphism
  - Function overloading
    - 2 functions can have the same name if the parameters they take are different.
  - C++ Classes
    - Syntax
      - class class-name{
        - private data and functions
        - public:
          - public data and functions
      - }Object name list //optional
    - Declaration
      - Class-name object-name
    - Accessing data members
      - Public data can be accessed
        - object-name.var;
        - object-name.func();
    - Member functions in classes
      - Inside class definitions
        -
        - #include <bits/stdc++.h>
        - using namespace std;
        - class Geeks
        - {
        - public:
        - string geekname;
        - int id;
        -
        - // printname is not defined inside class definition

```
    void printname();

    // printid is defined inside class definition
    void printid()
    {
       cout << "Geek id is: " << id;
    }
};
```

- Outside class definition

```
// Definition of printname using scope resolution operator ::
void Geeks::printname()
{
   cout << "Geekname is: " << geekname;
}
int main() {

   Geeks obj1;
   obj1.geekname = "xyz";
   obj1.id=15;

   // call printname()
   obj1.printname();
   cout << endl;

   // call printid()
   obj1.printid();
   return 0;
}
```

- Friend functions
  - declaring a function friend in a class, grants access to all the class's public and private members.
  - The prototype of the function has to be declared in the class.
  - Syntax
    - friend func_name();
- Static data members
  - If there's a static member in a class, that member is only declared once and all instances of that member share the same memory, thus the same.
  - Static members need to be defined elsewhere using scope resolution operators.
  - example

```
class c{
    static int r;
}
int c::r;
```

        - Local classes
            - Classes can be declared in functions that are limited to its scope.
    - C++ inheritance
        - The capability of a class to derive properties and characteristics from another class is called **Inheritance.**
        - **Sub class**:
            - The class that inherits properties of another class.
        - **Super class/base class:**
            - The class whose properties are inherited by subclass.
        - Syntax
            - class subclass_name: access_mode base_class_name
            - {
                - //
            - }
        - Subclass does not have access to private data of the superclass but it has the private data members.
        - Access modes
            - Public mode
                - Everyone can r/w
            - Protected
                - A protected member variable or function is very similar to a private member but it provided one additional benefit that they can be accessed in child classes which are called derived classes.mode
            - Private mode
                - Only the super class members can access
            - example
                - class A
                - {
                - public:
                - int x;
                - protected:
                - int y;
                - private:
                - int z;
                - };
                -
                - class B : public A
                - {
                - // x is public
                - // y is protected
                - // z is not accessible from B
                - };
                -
                - class C : protected A
                - {

        - // x is protected
        - // y is protected
        - // z is not accessible from C
        - };
        -
        - class D : private A   // 'private' is default for classes
        - {
        - // x is private
        - // y is private
        - // z is not accessible from D
        - };
- Types of inheritance
    - Single inheritance
        - One superclass one subclass
    - Multiple inheritance
        - Multiple superclasses
        - syntax
            - class subclass-name: access_mode base_class1, base_class 2…..{
                - //
            - }
    - Multilevel inheritance
        - One subclass is derived from another subclass.
    - Hierarchical inheritance
        - More than one subclass is derived from a single base class and has multilevel inheritance. Like a tree structure.
    - Hybrid inheritance
        - When more than one type of inheritance is used.
    - Multipath inheritance
        - A derived class with two base classes and these two base classes have one common base class is called multipath inheritance.
- Pointers to derived types
    - Although pointers must be of the same type to get an address, a subclass can point to a superclass and access the superclass members.
    - Superclass cannot point to a subclass.
        - Although you can use a superclass pointer to point to a subclass, it can only access what the subclass took from the superclass.
- Virtual Functions
    - Virtual functions are members of a superclass that can be modified by the derived subclass.
    - Syntax
        - virtual func_name (){};

        - The subclass only needs to write the function name, virtual is not needed.
    - Pure virtual function
        - a virtual function that has no definition in the superclass.
        - syntax
            - virtual type func_name(param) = 0;
    - Example
        - class test{
            - int i;
            - public:
                - virtual func(){
                    - cout<< “superclass”;
                - }
        - }
        - class sub_test: public test{
            - public:
                - func(){
                    - cout<<”modified to subclass”;
                - }
        - }
        - test *super_t;
        - sub_test sub_t;
        - super_t = &sub_t;
        - super_t->func(); //this will print “modified to subclass.
- C++ Constructor and destructors
    - Constructor
        - A constructor is a special type of member function of a class which initializes objects of a class. In C++, Constructor is automatically called when an object(instance of class) is created. It is a special member function of the class because it does not have any return type.
        - Constructor functions have the same name as the class.
        - Types of constructor
            - Default constructors
                - Default constructor is the constructor which doesn’t take any argument. It has no parameters.
            - Parameterized constructors
                - You can pass parameters like functions to a constructor
                - Has the same name as the class.
                - Initializes the class object.
                - example
                    - class Point {
                    - private:

```
    int x, y;
public:
    // Parameterized Constructor
    Point(int x1, int y1)
    {
        x = x1;
        y = y1;
    }
};

int main(){
    // Constructor called
    Point p1(10, 15);
    return 0;
}
```

- Copy constructor
  - A copy constructor is a member function that initializes an object using another object of the same class.
    - ClassName (const ClassName &old_obj);
    - ClassName (const ClassName &old_obj);

- Destructor
  - Destructor is an instance member function which is invoked automatically whenever an object is going to be destroyed.
  - ~constructor-name();
  - It cannot be declared static or const.
  - The destructor does not have arguments.
  - It has no return type, not even void.
- Copy constructor

Array of objects C++

- Arrays of objects are the same as normal array declaration.
- Example

```
class test{
    int i;
    public:
        if default constructor / no parameter
            test() { i = 0};
        if parameterized constructor
            test (int j, int k) { i = j;}
}
int main(){
    test t1[3];         //default constructor / no parameter
    test t1[3] = { test(1,2), test(3,4), test(4,5)}; //parameter
}
```

- The **this** pointer
  - this pointer points to the object's class members.
    - class test{
      - int i;
      - public:
        - test (int j) { this->i = j};
      - }
      - test t1(3);
      - //here i belongs to t1, therefore this->i
    - This pointer is important when 2 or more object's members are used.
      - class test{
        - int i;
        - public:
          - test(int j) { i = j};
          - test new_test(test t){
            - if(this->i > t.i){
              - //here this->i will have t1's i value and t.i will have t2's i value.
            - }
        - }
        - test t1(1), t2(2);
        - t1.new_test(t2);
- Pointers to class members
  - Pointer to a member is a generic pointer of that class.
    - int test::*d;
    - int *p;
    - test o;
    - p = &o.val //this is a specific val
    - d = test::val // this is offset of generic val
  - Pointing to a member of a class only gives the offset address of the object.
  - To create a pointer to member,
    - type class_name::*data_member_pointer;
    - type (class_name::*func_member_pointer)();
  - Access members
    - .* //when object is not pointer
      - t1.*data
      - t1.*func()
    - ->* //when object is pointer
      - t1_p->*data
      - t1_p->*func()
  - Member to pointer points at the class members, not the object member. When a pointer is created it may point to any object of the class.
  - Example

- #include <iostream>
- using namespace std;
- class test{
    - int i;
    - public:
        - int m;
        - test(int j) { i = j;);
        - void func(int k) { i = k;}
- }
- int main(){
    - //creating pointer to member point
    - int test::*value_p; //data member pointer
    - void (func::*func_p)(); //function member pointer
    - //declare objects
    - test t1(1), t2(2), *t1_p;
    - //defining pointers
    - value_p = &test::m; //assigning offset address
    - func_p = &test:func; //assigning offset address.
    - //non pointer object assess
    - cout << t1.*value_p << t2.*value_p;
    - cout <<(t1.*func_p)() << (t2.*func_p)();
    - //pointer object access
    - cout << t1_p->*value_p;
    - cout <<(t1_p->*func_p)();
    - return 0;
- }

- Passing reference to objects
    - Works the same as pointer reference
    - New object will not be created, it will point to the original object.
    - Example
        - class test{
            - int i;
            - public:
                - test (int j) {i = j;}
                - void func(test &t){
                    - cout << i;
                    - i = t;
                - }
        - }
        - test t1(3);
        - t1.func(t1);

- C++ Makefiles
- C++ File I/O
- C++ Vectors
    - Vectors are the same as dynamic arrays with the ability to resize itself automatically when an element is inserted or deleted, with their storage being handled automatically by the container.

- Vectors always know their sizes, even if referenced.
- Library
  - #include <vector>
- Definition
  - std::vector <data_type> variable_name;
- Declaration
  - std::vector <data_type> variable_name = {element1, element2);
- Functions
  - Iterators
    - begin()
      - Returns the iterator pointer of the first element
    - end()
      - Returns the iterator pointer of the theoretical element that follows the last element.
    - rbegin()
      - Returns the iterator pointer of the first element in reverse
    - end()
      - Returns the iterator pointer of the theoretical element that follows the last element in reverse.
    - cbegin()
    - cend()
    - crbegin()
    - crend()
  - Capacity
    - size()
      - Returns the number of elements in the vector
    - max_size()
      - Returns the maximum number of elements that the vector can hold.
    - capacity()
      - Returns the size of the storage space currently allocated to the vector expressed as number of elements
    - resize(n)
      - Resizes the container so that it contains ‘n’ elements.
    - empty()
      - Returns whether the container is empty.
    - shrink_to_fit()
      - Reduces the capacity of the container to fit its size and destroys all elements beyond the capacity.
    - Reserve()
      - Requests that the vector capacity be at least enough to contain n elements.
  - Element access
    - Reference operator [i]
      - vector_var[0];
    - at(i)

            - Returns a reference to the element at position ‘g’ in the vector
        - Front()
            - Returns a reference to the first element in the vector
        - back()
            - Returns a reference to the last element in the vector
        - data()
            - Returns a direct pointer to the memory array used.
        - push_back(var)
            - adds an element at the end of the vector
- C++ maps
    - Maps are associative containers that keep values paired with a key.
    - The key and the value can be of any data type.
    - Two values cannot have the same key.
    - The backend of maps is Red Black Trees.
    - The pairs in maps are stored in different locations in memory. So it+1 will not work, however, it++ works.
    -
    - Syntax
        - map <key_type> <value_type> map_var;
        - map <string> <int> map_var;
    - Functions
        - begin(), end()
            - returns the first and last value’s position
        - size, max_size()
            - returns the size and maximum size of the map
        - intert({key, value})
            - Adds new element to map
        - map_war[key]
            - Adds a new element to map, takes O(log(n)) but depends on key type. String takes string_size * O(log(n))
        - erase(iterator), erase(key)
            - erase(iterator)
                - eases key and value of the position given
            - ease(key)
                - eases key and value of the matching key
            - example code
                - if(it != maps_var.end())
                - {
                    - m.erase(it);
                - }
            -
        - clear()
            - clears the map.
        - Find()
            - auto it = maps_var.find(key); //find returns the iterator
            - example code

        - auto it = maps_var(3); //find 3 in the keys of the map
        - if(it != maps_var.end())
        - {
            - cout << (*it).first << (*it).second << endl;
        - }
- Iterating through a map
    - First way:
        - map<int, int> map_var;
        - map_var[1] = “a”;
        - map<int, int> :: iterator it;
        - for(it = map_var.begin(); it != map_var.end(); it++)
        - {
            - cout << (*it).first << “ “ << (*it).second;
            - //first is key, second is value
        - }
    - Second way
        - map<int, int> map_var;
        - map_var[1] = “a”;
        - for(auto &pr : mar_var) // : means in. pair in map.
        - {
            - cout << pr.first << “ “ << pr.second;
        - }
- Default value of a key is the default init of the variable type.
    - maps_var[1];
    - //key is inserted but value is not, if the value type is int then maps_var[1]’s value is 0. For string, an empty string and so on.
-