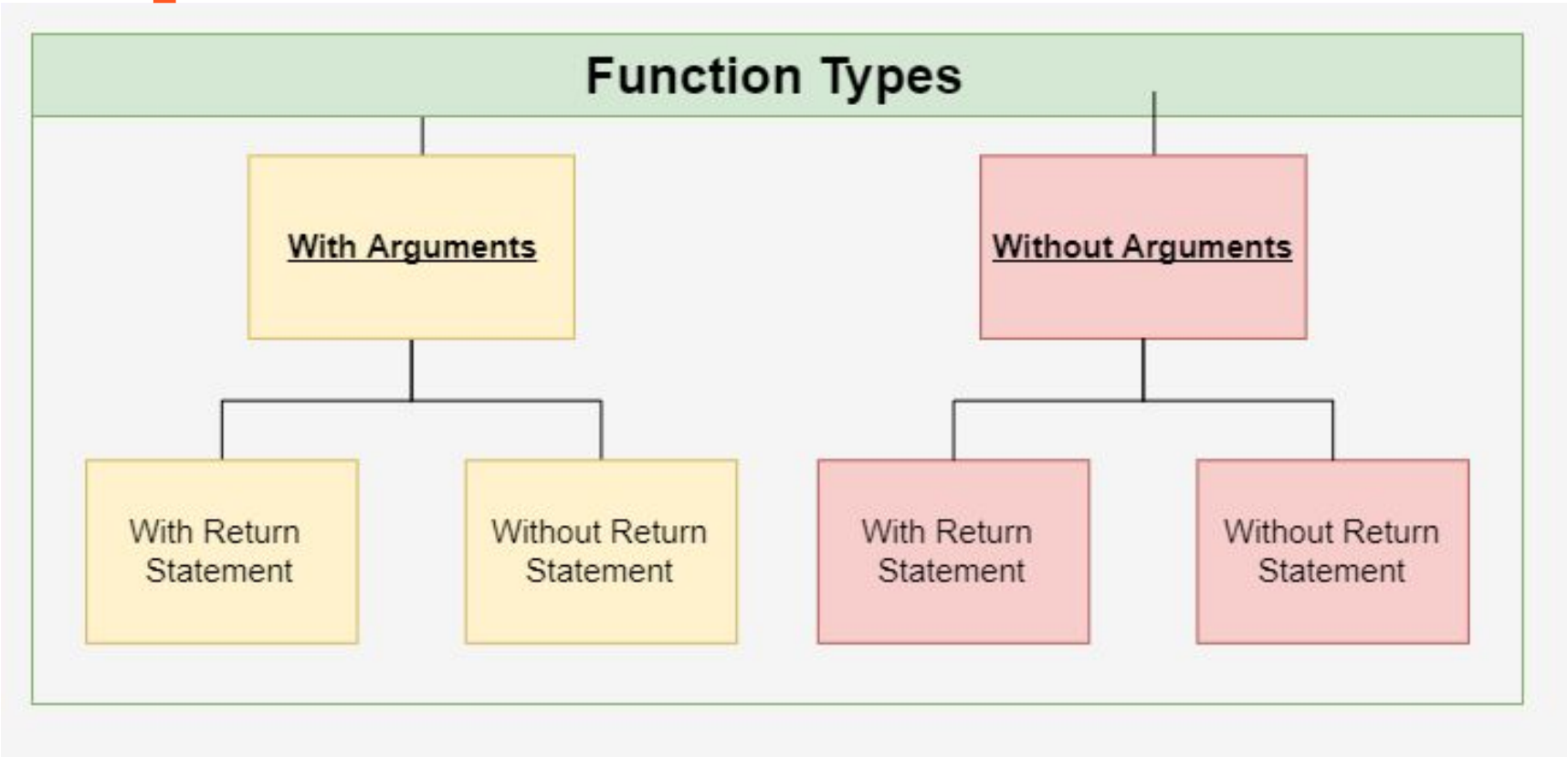
Derived C++

Derived Datatypes with Moatasem Elsayed

Content

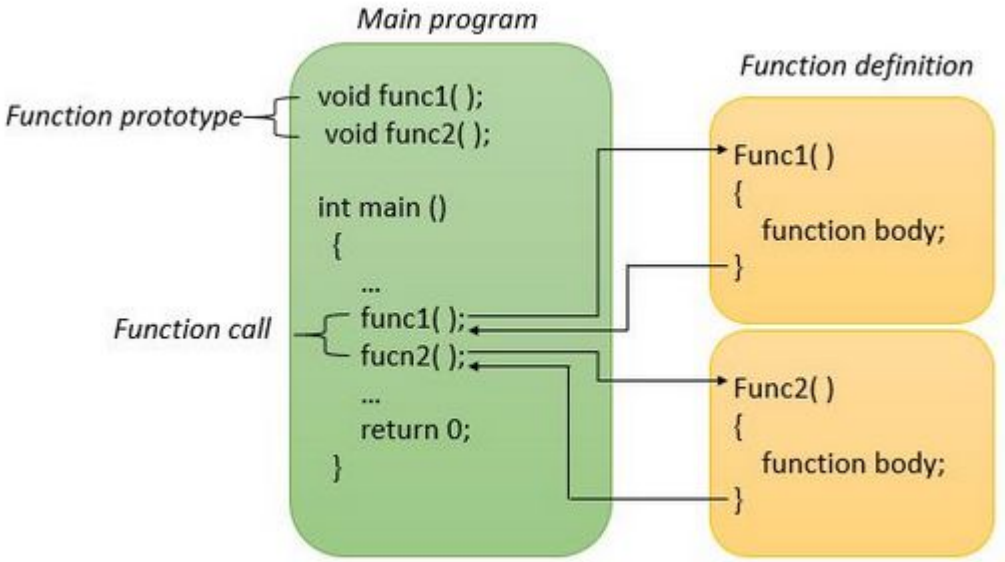
- *Functions*
- *Array*
- *pointers*
- *References*
- *auto*
- *Casting*
- *Lambda*
- *Const vs constexpr*

Recap on C


Function Types
With Arguments
Without Arguments
With Return Statement
Without Return Statement
With Return Statement
Without Return Statement

Basic function

```cpp
#include <iostream>

// Function declaration
int add(int a, int b) {
    return a + b;
}

int main() {
    // Function call
    int result = add(5, 3);
    std::cout << "Result: " << result << std::endl;
    return 0;
}
```




```cpp
// Declaration
int add(int a, int b);

// Definition
int add(int a, int b) {
    return a + b;
```

Default Parameters:

```cpp
#include <iostream>

void printMessage(std::string message = "Hello, World!") {
    std::cout << message << std::endl;
}

int main() {
    printMessage();           // Prints default message
    printMessage("Hi there!"); // Prints custom message
    return 0;
}
```

Tricky

```cpp
void fun(int x = 2, int y);
void fun(int x, int y)
{
    std::cout << x << " " << y << std::endl;
}

int main()
{
    fun(2);
    return 0;
}
```

```
OBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  JUPYTER

10
oatasem@moatasem-Inspiron-3542:~/c++/workspace$ g++ main.cpp
ain.cpp:3:25: error: default argument missing for parameter 2 of 'void fun(int, int)'
    3 | void fun(int x = 2, int y);
      |                     ~~~~^
ain.cpp:3:14: note: ...following parameter 1 which has a default argument
    3 | void fun(int x = 2, int y);
      |          ~~~~^~~~
oatasem@moatasem-Inspiron-3542:~/c++/workspace$
```

Function Overloading:

```cpp
#include <iostream>

int multiply(int a, int b) {
    return a * b;
}

double multiply(double a, double b) {
    return a * b;
}

int main() {
    int result1 = multiply(3, 4);
    double result2 = multiply(2.5, 3.0);

    std::cout << "Result 1: " << result1 << std::endl;
    std::cout << "Result 2: " << result2 << std::endl;

    return 0;
}
```

demangle

```
moatasem@CAI1-L14000:~/vsomeIp$ objdump -t --demangle | grep fun
0000000000001295 l     F .text  0000000000000019              _GLOBAL__sub_I__Z3funi
00000000000011e3 g     F .text  000000000000003d              fun(int, int)
00000000000011a9 g     F .text  000000000000003a              fun(int)
moatasem@CAI1-L14000:~/vsomeIp$ objdump -t | grep fun

0000000000001295 l     F .text  0000000000000019              _GLOBAL__sub_I__Z3funi
00000000000011e3 g     F .text  000000000000003d              _Z3funii
00000000000011a9 g     F .text  000000000000003a              _Z3funi
moatasem@CAI1-L14000:~/vsomeIp$
```

Tricks

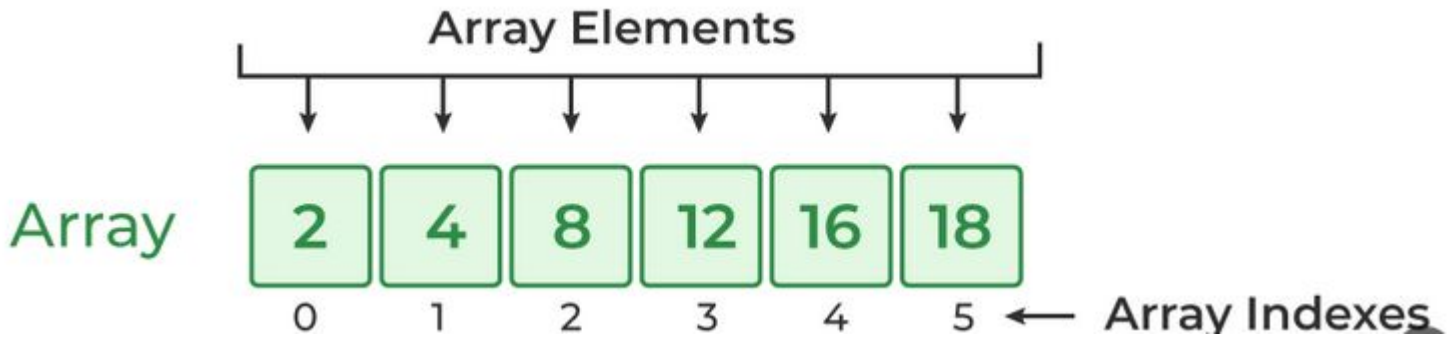
error: call of overloaded ‘fun(int)’

is ambiguous

```
4   void fun(int x, int y = 3)
5   {
6       std::cout << x << " " << y << std::endl;
7   }
8   void fun(int x, float y = 2)
9   {
10      std::cout << "hello float" << std::endl;
11  }
12  int main()
13  {
14      fun(2);
15      fun(2, 3);
16      return 0;
17  }
```

```
PROBLEMS 2   OUTPUT   DEBUG CONSOLE   TERMINAL   JUPYTER

  14 |     fun(2);
     |         ^
main.cpp:4:6: note: candidate: ‘void fun(int, int)’
   4 | void fun(int x, int y = 3)
     |      ^~~
main.cpp:8:6: note: candidate: ‘void fun(int, float)’
```

Arrays



Declaration

To declare an array, specify the type of its elements and the number of elements it can hold:

```
int numbers[5]; // Declare an integer array with 5 elements
```

Initialization

Arrays can be initialized when declared:

```
int numbers[5] = {1, 2, 3, 4, 5}; // Initialize array elements
```

Or you can omit the size, and the compiler will determine it from the number of elements:

```
int numbers[] = {1, 2, 3, 4, 5}; // Compiler determines size as 5
```

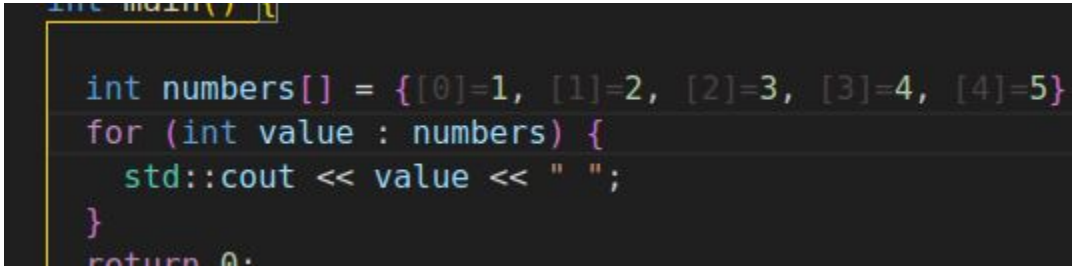
Accessing Elements

```
int thirdNumber = numbers[2]; // Access the third element (3)
```

Iterating through an Array

```
for (int i = 0; i < 5; ++i) {
    cout << numbers[i] << " ";
}
```

Use Ranged loop



Multidimensional Arrays

Arrays can have more than one dimension:

```
int matrix[3][3] = {
    [0]={[0]=1, [1]=2, [2]=3},
    [1]={[0]=4, [1]=5, [2]=6},
    [2]={[0]=7, [1]=8, [2]=9}
};

    for (int i = 0; i < 3; ++i) {
        // Iterate through columns
        for (int j = 0; j < 3; ++j) {
            std::cout << matrix[i][j] << " ";
        }
        std::cout << std::endl; // Move to the next line after each row
    }
```

Array Size

```
int main() {
    int numbers[] = {[0]=1, [1]=2, [2]=3, [3]=4, [4]=5};

    int size = sizeof(numbers) / sizeof(numbers[0]); // Calculate si
```

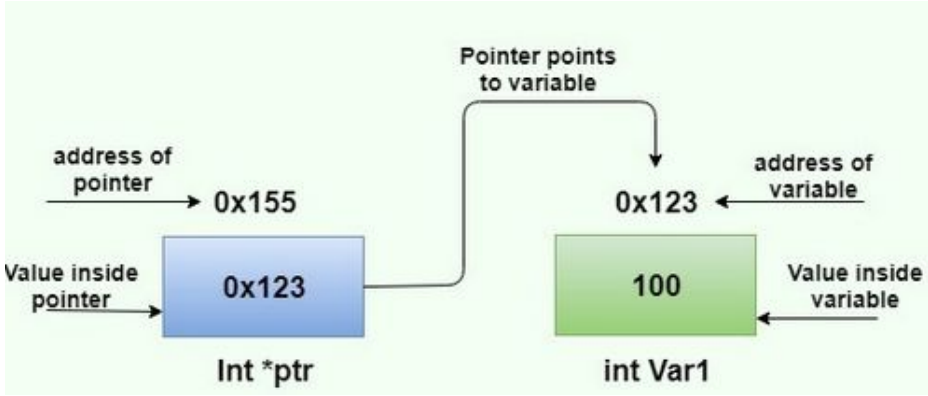
Arrays and Functions

```
void printArray(int arr[], int size) {
    for (int i = 0; i < size; ++i) {
        std::cout << arr[i] << " ";
    }
}

int main() {
    int numbers[] = {[0]=1, [1]=2, [2]=3, [3]=4, [4]=5};
    int size =
        sizeof(numbers) / sizeof(numbers[0]); // Calculate the size of the array

    printArray(arr: numbers, size);
    return 0;
}
```

Pointers

Basics of Pointers

- A pointer is a variable that stores a memory address.
- Declare a pointer with the * symbol




```
int x = 10;
int *ptr = &x; // ptr now holds the address of x
```

Dereferencing Operator (*)

Use the dereferencing operator * to access the value pointed to by the pointer:

```
int value = *ptr; // value now holds the value of x (10)
```

Pass by pointer

```
// Example 4: Pointers and Functions
void modifyValue(int *ptr) {
    *ptr = 100;
}
```

```
int num = 50;
modifyValue(&num);
std::cout << "Modified num: " << num << std::endl;
```

Cont ..

Null Pointers

Pointers can be set to `nullptr` (C++11) to indicate no valid address:

```
int *ptr = nullptr; // Initialize pointer to nullptr
```

Dynamic Memory Allocation

Use `new` to allocate memory on the heap:

```
int *ptr = new int; // Allocates memory for an integer
*ptr = 42; // Assign a value to the memory location
```

Memory Deallocation

Always release dynamically allocated memory with `delete`:

```
delete ptr; // Frees the memory allocated with new
ptr = nullptr; // Point the pointer to nullptr after deletion
```

Arrays and Pointers

- Arrays are closely related to pointers:
  cpp

```
int arr[5] = {1, 2, 3, 4, 5};
int *ptr = arr; // Pointer points to the first element of the array
```

Pointer Arithmetic

- Pointers can be incremented and decremented:
  cpp

```
int *nextPtr = ptr + 1; // Points to the next element in the array 2
```

Pointers to Functions

- Pointers can point to functions:

```
void func() { cout << "Hello!"; }
void (*funcPtr)() = &func; // Pointer to the function
(*funcPtr)(); // Calling the function through the pointer
```

References vs. Pointers

References vs. Pointers

- References are aliases for existing variables, while pointers can be reassigned:
  cpp

```cpp
int x = 10;
int &ref = x; // Reference to x
int *ptr = &x; // Pointer to x
```

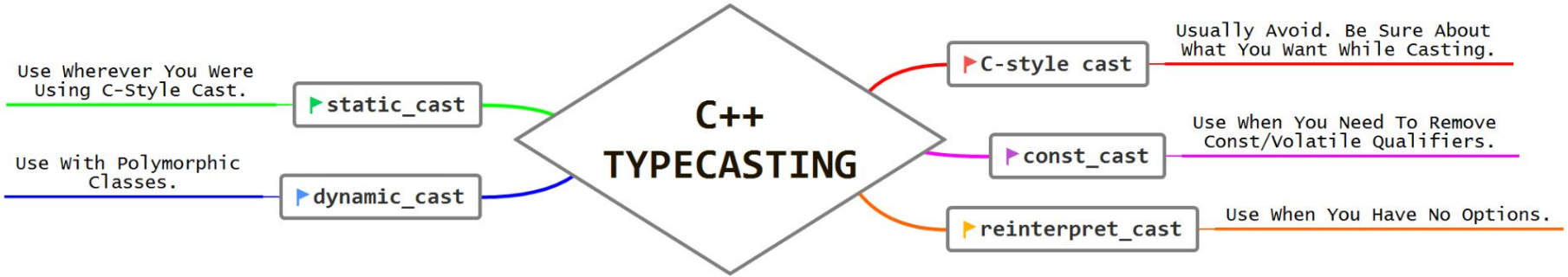
```cpp
int x{10};
int &y = x;

int *ptr = &x; // lvalue {it has a certin name , that you did  }

y = 3;
std::cout << "ref y= " << y << " x= " << x << "ptr " << *ptr << std::endl;

// change referance
int n = 0;
y = n;
y = 12;
std::cout << "x =" << x << "n=" << n << std::endl;
// unitialized referance is error
// int &y2; // Error

return 0;
}
```

auto

```
{

    auto var = 10;
    auto var2 = 10.5;
    auto var3 = 'a';
    auto var4 = "const char*";
    auto var5 = std::string("hello String ");
    // with const
    const int x = 10;
    auto var6 = x; // var is int
    //** to make it const
    const auto var7 = x;
    // with referance const
    auto &var8 = x;
    // with pointer const
    auto ptr = &x;
    // std::initliaizer_list
    auto lst = {1, 2, 3, 4, 5};
    // error
    // auto lst{1, 2, 3, 4};
    return 0;
}
```

```
22
23      // Data d;
24      // printf("welcome to c programming ");
25      // d.HelloWorld();
26      return 0;
27  }*/
28  int main()
29  {
30    int var = 10;
31    double var2 = 10.5;
32    char var3 = 'a';
33    const char * var4 = "const char*";
34    std::basic_string<char> var5 = std::basic_string<char>(std::basic_string<char>("h
35    const int x = 10;
36    int var6 = x;
37    const int var7 = x;
38    const int & var8 = x;
39    const int * ptr = &x;
40    std::initializer_list<int> lst = std::initializer_list<int>{1, 2, 3, 4, 5};
41    return 0;
42  }
43
```

C++ casting


C++
TYPECASTING
static_cast
Use Wherever You Were Using C-Style Cast.
dynamic_cast
Use With Polymorphic Classes.
C-style cast
Usually Avoid. Be Sure About What You Want While Casting.
const_cast
Use When You Need To Remove Const/Volatile Qualifiers.
reinterpret_cast
Use When You Have No Options.

Recap C style cast

Static_cast

Static_cast:

This is the simplest type of cast which can be used. It is a compile time cast.It does things like implicit conversions between types (such as int to float, or pointer to void*), and it can also call explicit conversion functions (or implicit ones).GeeksforGeeks

```
int a = 10;
char c = 'a';

// pass at compile time, may fail at run time
int* q = (int*)&c;
int* p = static_cast<int*>(&c); // error: invalid static_cast from type ‘char*’ to type ‘int*’
```

```
int value=10;
float x=static_cast<float>(value);
```

Rule 5–0–5
Rule 5–0–6

const_cast

const_cast is used to cast away the constness of variables

```
const int val = 10;
const int *ptr = &val;
// int*ptr1=ptr            #ERROR
int *ptr1 = const_cast <int *>(ptr);
// int*ptr1=static_cast<int*>(ptr); // ERROR static_cast cannot cast away const or other type qualifiersC/C++(694)
```

Rule 5–2–5	**(Required)**	**A cast shall not remove any *const* or *volatile* qualification from the type of a pointer or reference.**

```
void f ( const char * p )
{
 *const_cast< char * >( p ) = '\0'; // Non-compliant
    std::cout <<p<<std::endl;//Segmentation fault
}
int main ( )
{
 f ( "Hello World!" );
}
```

reinterpret_cast

It is used to convert a pointer of some data type into a pointer of another data type, even if the data types before and after conversion are different.

```
int* p = new int(65);
char* ch = reinterpret_cast<char*>(p);
```

```
char* x=new char(5);
```

a value of type "char *" cannot be used to initialize an entity of type "int *"

View Problem (Alt+F8) Quick Fix... (Ctrl+.)

```
int*ptr=reinterpret_cast<char*>(x);
```

Use it correctly

1. reinterpret_cast is a very special and dangerous type of casting operator. And is suggested to use it using proper data type i.e., (pointer data type should be same as original data type).
2. It can typecast any pointer to any other data type.

Rule 5–2–7	**(Required)**	**An object with pointer type shall not be converted to an unrelated pointer type, either directly or indirectly.**

[Unspecified 5.2.10(7)]

Rationale

The result of converting from a pointer to an unrelated type is unspecified.

```
class A {
public:
    int x=10;
    void fun_a()
    {
        cout << " In class A\n";
    }
};

class B {
public:
    int x=12;
    void fun_b()
    {
        cout << " In class B\n";
    }
};

int main()
{
    // creating object of class B
    B* x = new B();
    // converting the pointer to object
    A* new_a = reinterpret_cast<A*>(x);
    // accessing the function of class A
    new_a->fun_a();                 // In class A
    std::cout <<new_a->x<<std::endl; // 12
    return 0;
}
```

dynamic_cast

Very important and we will postponed after OOP

In practise

- Avoid C-style casts. Be sure about what you want while casting.
- Use static_cast wherever you were using C-style cast.
- Use dynamic_cast with polymorphic classes.
- Use const_cast when you need to remove const or volatile qualifiers.
- Use reinterpret_cast when you have no options.

Note: const_cast and reinterpret_cast should generally be avoided because they can be harmful if used incorrectly. Don’t use it unless you have a very good reason to use them.

Lambda expression

C++ 11 introduced lambda expressions

```
[ capture clause ] (parameters) -> return-type
{
   definition of method
}
```

```
std::vector<int>v{1,2,3,4,5,6,7,8};
// lambda expression to print vector
std::for_each(v.begin(), v.end(), [](int i)
{
    std::cout << i << " ";
});
```

Cpp insight,check that after Class

```
6
7  // Driver Code
8  int main()
9  {
10   auto func = [] (int first, int second)
11   {
12       return first + second;
13   };
14   func(2,3);
15     return 0;
16 }
17
```

```
7  // Driver Code
8  int main()
9  {
10
11   class __lambda_10_15
12   {
13     public:
14     inline int operator()(int first, int second) const
15     {
16       return first + second;
17     }
```

```
37    func.operator()(2, 3);
```

Capture list

```
6  int main()
7  {
8      int x=10;
9    int y=12;
10   [&x,y](){
11     x=1;
12   return x+y;
13   }();
14
15     return 0;
16 }
```

```
{
  int x = 10;
  int y = 12;

  class __lambda_10_3
  {
    public:
    inline int operator()() const
    {
      x = 1;
      return x + y;
    }

    private:
    int & x;
    int y;

    public:
```

```
[&x,y]()mutable{
  x=1;
return x+y;
}().
```

```
  inline int operator()()
  {
    x = 1;
    return x + y;
  }
```

```
int temp=10;
[](int number){
   return number +temp;// 1- Error need to capture it
};

[temp](int number){    // capture value
   // temp =1*2;       // 2- capture it with const temp
   return number +temp;// 3- read only
};

[&temp](int number){ // capture referance
   temp =1*2;        // 4- change temp of temp itself
   return number +temp;
};

[&](int number){    // capture all by referance
   temp =1*2;       // 4- change temp of temp itself
   return number +temp;
};

[=](int number){       //5- capture all by value
   return number +temp;
};

[=,&temp](int number){ //6- capture all by value except temp by referances
   return number +temp;
};
```

Non-static class data member

```
class Data{
public:
int value;
int x;
void fun(){
    [](){
        // std::cout <<value<<std::endl;    // 7- ERROR need to capture
    };
    [this](){
        std::cout <<value<<std::endl;    // 8- this captured the class member by referance
        value=12;
    };
    [x,this](){                          // 9- ERROR once you use this nothing before or after it
        std::cout <<value <<std::endl;
        value=12;
    };
    auto a=this->x;
    auto b=this->value;
    [&a,b](){                           // 10-Only objects with automatic storage duration can be captured by a
        std::cout <<b <<std::endl;      //lambda in C++11 (i.e. local variables and function parameters). If you want the effect of capturing a
        a=12;                           //non-static class data member, you can either capture the this pointer as i
    };
}
};
```

Memory Usage

```
2  void test(){
3              std::cout<<"Hello "<<std::endl;
4
5  }
6  int main(){
7  //   Test d;
8    test();
9  }
```

```
moatasem@CAI1-L14000:~/vsomeIp$ !g++
g++ test.cpp
moatasem@CAI1-L14000:~/vsomeIp$ !size
size a.out
   text    data     bss     dec     hex filename
   2677     664     280    3621     e25 a.out
moatasem@CAI1-L14000:~/vsomeIp$ objdump -x --demangle | grep -i test
0000000000000000 l    df *ABS*  0000000000000000              test.cpp
000000000000123d l     F .text  0000000000000019              _GLOBAL__sub_I__Z4t
00000000000011a9 g     F .text  0000000000000033              test()
moatasem@CAI1-L14000:~/vsomeIp$
```

```
int main(){
//   Test d;
//   d();
[](){
    std::cout<<"Hello "<<std::endl;
}();
}
```

```
moatasem@CAI1-L14000:~/vsomeIp$ !g++
g++ test.cpp
moatasem@CAI1-L14000:~/vsomeIp$ !size
size a.out
   text    data     bss     dec     hex filename
   2850     672     280    3802     eda a.out
moatasem@CAI1-L14000:~/vsomeIp$ objdump -x --demangle | grep -i lam
00000000000011ca l     F .text  0000000000000037              main::{lambda()#1}::operator()() const
moatasem@CAI1-L14000:~/vsomeIp$
```

Corner cases

auto

```
moatasem@CAI1-L14000:~/vsomeIp$ g++ test.cpp -std=c++11 && ./a.out
test.cpp: In function 'int main()':
test.cpp:39:14: error: use of 'auto' in lambda parameter declaration only available with '-std=c++14'
   39 | auto fn=  [](auto x, auto y){
      |              ^~~~
```

```
//in C++14
auto fn=  [](auto x, auto y){
    return x+y;
};
std::cout << typeid(fn(2,3)).name()<<std::endl;
std::cout << typeid(fn(2,3.5F)).name()<<std::endl;
std::cout << typeid(fn(2,3.5)).name()<<std::endl;
```

mutable

```
int main(){
 int temp=10;

 [temp]()mutable{
    temp=15;
    std::cout <<temp<<std::endl;
 }();
}
```

Initialization

```
//C++14
 [&v=temp,x=temp](){ // v by referance , x by value
    std::cout <<v<<std::endl;
    v=12;
 }();
```

No Misra Rules

When we use it ?

1- passing function as argument

2- traceability

3- return function

C++ / Algorithm library

std::sort

Defined in header <algorithm>

```
template< class RandomIt >
void sort( RandomIt first, RandomIt last );                                   (1)  (until C++20)
template< class RandomIt >
constexpr void sort( RandomIt first, RandomIt last );                              (since C++20)

template< class ExecutionPolicy, class RandomIt >
void sort( ExecutionPolicy&& policy,                                          (2)  (since C++17)
           RandomIt first, RandomIt last );

template< class RandomIt, class Compare >
void sort( RandomIt first, RandomIt last, Compare comp );                     (3)  (until C++20)
```

std::find, std::find_if, std::find_if_not

Defined in header <algorithm>

```
template< class InputIt, class T >
InputIt find( InputIt first, InputIt last, const T& value );                  (1)  (until C++20)
template< class InputIt, class T >
constexpr InputIt find( InputIt first, InputIt last, const T& value );             (since C++20)

template< class ExecutionPolicy, class ForwardIt, class T >
ForwardIt find( ExecutionPolicy&& policy,                                     (2)  (since C++17)
                ForwardIt first, ForwardIt last, const T& value );

template< class InputIt, class UnaryPredicate >
InputIt find_if( InputIt first, InputIt last, UnaryPredicate p );             (3)  (until C++20)
template< class InputIt, class UnaryPredicate >
constexpr InputIt find_if( InputIt first, InputIt last, UnaryPredicate p );        (since C++20)
```

```
auto fn(){
    return [](int x){
        return x+1;
    };
}
int main()
{
    std::cout <<fn()(1);
    return 0;
}
```

const

```
1- Linkage
    c++ : Internal Linkage
    c:    External Linkage
*/
const int ll = 0;

int main()
{
    //**********2- Switch **************
    int var = MOATASEM;
    const int x = 97;
    switch (var)
    {
    case x:
        std::cout << x << std::endl;
        break;
    }
```

```
    //*************3- Array **************
    /*
    #define MAX_FOOS 10
    int foos[MAX_FOOS];
    */
    // const int max_foos = 12;
    // int foos[max_foos];
    // std::cout << std::size(foos) << std::endl;
    //**********4- With pointer **************
    // Error const with int
    const int var2 = 10;
    int *ptr = &var2;
```

Constexpr vs const

```
const int var1 = 5;
constexpr int var2 = 7;
```

✅

```
int getRandomNo()
{
  return rand() % 10;
}

int main()
{
    const int varB = getRandomNo();       // OK
    constexpr int varC = getRandomNo();   // not OK! compilation error
```

❌

Value of **varB** would not anymore compile time. While statement with varC will throw compilation error. ***The reason is constexpr will always accept a strictly compile-time value.***

Constexpr function

```cpp
constexpr int sum(int x, int y)
{
    x=1;                            // 1- Error in c++11 must be one statment
    // std::cout<<"Hi"<<std::endl;  //2- Error only const experssions
    for(int i=0;i<3;i++)            //3- aslong as experssions are not depend on runtime no problem
    {
        x+=i;
    }
    // for(int i=0;i<rand();i++){}          //4- Error because of rand()

    return x + y;                       //5- return must be const expr
}

int main()
{
    const int result = sum(10, 20);     // 6- Here, you can use constexpr
    cout << result;
    return 0;
}
```

```
19      {
(gdb) disassemble /s
Dump of assembler code for function main():
test.cpp:
19      {
=> 0x0000000008001189 <+0>:     endbr64
   0x000000000800118d <+4>:     push   %rbp
   0x000000000800118e <+5>:     mov    %rsp,%rbp
   0x0000000008001191 <+8>:     sub    $0x10,%rsp

20          const int result = sum(10, 20);     // Here, you can use constexpr as well
   0x0000000008001195 <+12>:    movl   $0x1e,-0x4(%rbp)
--Type <RET> for more, q to quit, c to continue without paging--

21          cout << result;
   0x000000000800119c <+19>:    mov    $0x1e,%esi
   0x00000000080011a1 <+24>:    lea    0x2e98(%rip),%rdi        # 0x8004040 <_ZSt4cout@@GLIBCXX_3.4>
   0x00000000080011a8 <+31>:    callq  0x8001090 <_ZNSolsEi@plt>

22          return 0;
   0x00000000080011ad <+36>:    mov    $0x0,%eax
```

If there is no errors in constexpr scope but its argument workin on runtime dependent so it will be a normal function

```
constexpr int result = sum(10, rand());
```

```
{
    const int result = sum(10, rand());
    cout << result;
    return 0;
```

No Misra Rules

```
moatasem@CAI1-L14000:~/vsomeIp$ objdump -t --demangle a.out | grep -i "sum"
0000000000001247  w    F .text  0000000000000038              sum(int, int)
```

```
0x00000000080011b1 <+8>:     sub    $0x10,%rsp

28          const int result = sum(10, rand());     // 5- Here, you can use constexpr as well
0x00000000080011b5 <+12>:    callq  0x8001080 <rand@plt>
0x00000000080011ba <+17>:    mov    %eax,%esi
0x00000000080011bc <+19>:    mov    $0xa,%edi
0x00000000080011c1 <+24>:    callq  0x8001247 <sum(int, int)>
0x00000000080011c6 <+29>:    mov    %eax,-0x4(%rbp)
```

Tasks

1- create a function to find the maximum number of array
2- create a function to search to the number in the array which number is taken from user
3- delete number in array
4- merge two arrays together
5-find the even and odd numbers in the array

Simple Lambda: Write a lambda function to calculate the square of a given number.

Sort with Lambda: Use lambda functions to sort an array of integers in ascending and descending order.

Develop a simple address book program that allows users to add, update, and search for contacts.