```cpp
//////// Lecture 90: Constant Expressions (constexpr)
int main(){
    constexpr int i = 10; //this indicates that the value of 'i' is computed at compile time, therefore we can use i in those expressions that expect a compile time constant.
    // for example, we can use i as a size of the array.
    int arr[i]; // set the size of the array.
}
// now you might ask, we can do the same thing with the const keyword:
int main(){
    const int i = 10;
    int arr[i];
}
// So what's the difference?
// The initialisation of the constexpr is evaluated at compile time, in the second case, the expression is also evaluated at compile time, but not all constant declarations are evaluated at compile time.
// some constants are initialised at runtime.

int GetNumber(){
    return 42;
}
int main(){
    int j = GetNumber(); // the initialisation of j is deferred until runtime.
    // So its value cannot be computed at compile time.hence it cannot be used as a size of an array.
}
// if we try to initialise the constexpr with GetNumber() it will NOT WORK.
const int j = GetNumber(); // will not work!

// A constant expression variable can only be initialised when there is a constant expression, that means it does not return a constant value. (The compiler doesn't know that the function is returning a constant value.)

// but if we know that a function is returning a const value, then we would like to optimise it.
// We would like the compiler to evaluate the return value of GetNumber() at compile time, and we can do that by making this function as a constexpr so we will have to mark the return type with constexpr.

/////// Do this instead:
constexpr int GetNumber(){
    return 42;
}
// This indicaates that GetNumber() is a constant expression functoion and it's return value is computed at compile time.
int main(){
    constexpr int j = GetNumber(); // now j can be initiaised at compile time as well.
    // And that is why now we can use j to denote the size of the array.
    int array[j];
    // What kind of functions can be constexpr? If a function returns a value that can be computed at compile time, then it can be a constant expression function, but such a function must accept and return ONLY LITERAL TYPES.
    // What are literals?  Literal types are those which are allowed in constant expressions, such types are void types, scalar types - integer / float / double / arrays / CLASSES that has constant expression constructors.
}
// So GetNumber() is a constant expression function, What if we use it to initialise a variable that is not a constant? In this context, Getnumber() would be have like a normal function and not a constant expression function.
// And the initialisation of j will be occur at RUNTIME.
constexpr int GetNumber(){
    return 42;
}
```

```cpp
int main(){
    int j = GetNumber(); // occurs at runtime.
}

/// Let's create a function that accepts 2 numbers and returns their sum.

// Is it possible to make this function a constexpr function?  If we marke the return type as constexpr and we know that sum() accepts only literal types and also returns a literal type, we can use it as a constant expression.
constexpr int Sum (int x , int y ){return x + y;}
int main(){
    constexpr int sum = Sum(2,3); // the execution will be really fast! compared to the case when we invoke Add() as a non constexpr function.
}
// A constexpr function can be used both in context of a non-constexpr and constexpr.

// If we invoke it like this:
int main(){
    int sum = Sum (2,3); // sum will be computed at runtime.
}

// It can even accept variables as arguments.
int main(){
    int x = GetNumber(); // not a compile time constant
    constexpr int sum = Sum(x,3); // will not be a constexpr function in this case.  This does not work, and you will get a COMPILER ERROR.
}
// The one thing about constexpr functions is that they can ONLY HAVE A SINGLE LINE STATEMENT inside, and that statement should be a return statement.
// So this is a limtitation, because most functions would need more than 1 line of code.
// So let's create 1 more function, which returns the maximum of 2 integers:
// --> we'll have to implement it in a single
```

```
  statement.  This works.
constexpr int Max(int x, int y){
    return x > y ? x : y; // this works.
}
constexpr int Max(int x, int y){
    if (x > y) return x;
    return y;
} // this doesn't work! Error: constexpr functions can
  only have 1 return statement. (C++11)
// However C++14 has relaxed these rules, so in C++14,
  you can have conditional statements inside functions.
// There is actually a comprehensive set of rules, and
  you can look them up on cppreference.com.
// when we build this now, it builds fine, and we can
  use this Max() function in context of a constexpr.
// All constexpr are IMPLICITLY INLINE!  this means
  that you have to write a constant expression just
  like an inline function, so constexpr functions will
  always be defined in a header file.

// How should we decide to use the const keyword or
  the constexpr keyword?



```