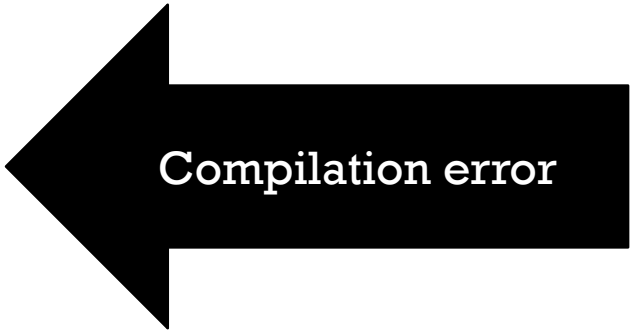
CONSTANTS

- Variables
- Pointers
- Function arguments and return types
- Class Data members
- Class Member functions
- Objects

CONSTANT VARIABLES IN C++

- Cannot change its value of the variable.
- Value of the variable will not change during its lifetime.
- Constant variables must be initialized while they are declared.

```
int main()
{
    const int i = 10;
    const int j = i + 10;
    i++;
}
```



POINTERS WITH CONST KEYWORD

Declaration can be done in TWO ways.

- Pointer to a constant variable
  - The pointer is pointing to a const variable.

```
const int* u;
```

- Constant Pointer
  - Cannot change the pointer, to which it points to
  - Can change the value that it points to, by changing the value of the variable.
  - Useful when one want to change the value but not the memory.
    - The pointer will always point to the same memory location, as it is defined with const keyword, but the value at that memory location can be changed.

```
int x = 1;
int* const w = &x;
```

```
#include <iostream>
using namespace std;

int main() {
    int a = 10;
    int b = 20;
    const int c = 30;

      const int* ptrToConst = &a;
      cout << "ptrToConst points to a = " << *ptrToConst << endl;

      ptrToConst = &c;
      cout << "ptrToConst now points to c = " << *ptrToConst << endl;

      int* const constPtr = &a;
      *constPtr = 25;
       cout << "constPtr points to a with new value = " << *constPtr << endl;

      const int* const constPtrToConst = &c;
      cout << "constPtrToConst points to c = " << *constPtrToConst << endl;
    return 0;
}
```

Compile?

CONSTANT FUNCTION ARGUMENTS AND RETURN TYPES

- The return type or arguments of a function can be defined as constant.
- Member functions should not modify member data.

```
void func(const int i)
{
   i++;   // error
}

const int funcCont()
{
   return 6;
}
```

DEFINING CLASS DATA MEMBERS AS CONSTANT

- Data members are not initialized during declaration.
- The initialization is done in the constructor.
- Once initialized cannot change the value.

Exercise:

Write a function called setValue within the class "TestConst" and try to change the variable 'var'.

Compiles correctly

```
class TestConst
{
   const int var;
   public:
   TestConst(int x):var(x)
   {
      cout << "Const Var: " <<
var<<endl;
   }
};

int main()
{
   TestConst testObj1(3);
   TestConst testObj2(2);
}
```

DEFINING CLASS OBJECT AS CONSTANTS

- During the object lifetime Data members can never be changed.

```
const class_name object;
```

CONSTANT OBJECTS

- Objects are seldom passed by value in C++.
- If you want to pass objects, they are passed by passing a **const** reference
- Passing by **const** reference
  - Avoids copying the object
  - Protects the object from unintentional changes.

```
void displayTime(const Time &t)
```

DEFINING CLASS MEMBER FUNCTION AS CONSTANTS

- **const keyword** must be added to both the function signature and implementation.
- A constant member functions never modifies data members in an object.

```
returnType functionName() const
```

```
int displayList()  const;

  int List:: displayList () const
  {
     .........
  }
```

'THIS' KEYWORD

It points to the **object** that is currently invoking the method

Rectangle.cpp

THE "THIS" POINTER IN C++

- How do objects work with functions and data members of a class?
- When an object is created
  - Each object gets its own copy of the data member.
  - All-access the same function definition as present in the code segment.
- Each object gets its own copy of data members and share a single copy of member functions.
- How the proper data members are accessed and updated?
  - An implicit pointer 'this' (value is the address of the object that generated the call)
  - Passed as a hidden argument to all non-static member function calls and
  - Available as a local variable within the body of all non-static functions.
  - 'this' pointer is not available in static member functions as static member functions can be called without any object (with class name).

IN RECTANGLE EXAMPLE...

- this points to the object that invoked getWL().

```
void Rectangle::getWL()
{
    cout<<"Width is "<<this->width<<endl;
    cout<<"Length is "<<this->length<<endl;
}
```

EXERCISE

```
#include<iostream>
using namespace std;

class Counter {
    int value;
public:
    Counter(int v = 0) : value(v) {}
    Counter& increment() {
        value++;
        return *this;  // returns current object by reference
    }
    void display() {
        cout << "Value: " << value << endl;
    }
};

int main() {
    Counter c;
      c.increment().increment().increment();
      c.display();
}
```

HOMEWORK

Friend Functions(What, Why & How)

Forum post discussion