Lecture 4

Thursday, February 13, 2020 12:13 AM

- Use class to work on the currency conversion problem
  - homework 2
- CurrencyFactory
  - eager initialization
    - create all currency objects in default constructor
    - then use curr_type to return the object
- Automatic objects / lazy initialization
  - destroy objects when no longer needed--> **free store objects**
  - using the keyword **new**
    - create obj, return address of the obj
    - use * or -> to access free store obj
    - Currency* c = **new** Currency("USD", 1.0);
    - (*c).GetSymbol();
    - c->SetExchangeRate(0.95);
    - delete c;
  - be aware of memory leak.

```
class CurrencyFactory
{
public:
        CurrencyFactory();
        Currency* GetCurrency(int currencyType);
private:
        Currency* currencies_[5];
};
CurrencyFactory::CurrencyFactory()
{
        currencies_[USD] = new Currency("USD", 1.0);
        currencies_[EUR] = new Currency( "EUR", 0.9494);
}
Currency* GetFactory::GetCurrency(int currencyType)
{
        return currencies_[currencyType];
}
```

  - efficiency : no longer
    - default construct Currency objects
    - I assign Currency objects
    - I copy construct Currency objects
  - delete free store objects

```
        CurrencyFactory::~CurrencyFactory()
        {
                for (int i=0; i<5; ++i)
                {
                        delete currencies_[i];
                }
        }
```

- templates
  - allow us to write functions and classes with types as parameters.
  - parameterized classes/functions.
  - function templates: to write functions that work with different types
  - class templates: to write classes where the member variables can be different types
    ```
    template <typename T>
    T Add(const T& a, const T& b)
    {
        return a+b;
    }
    int res1 = Add(1, 2);
    double res2 = Add<double>(1.2, 2.3);
    ```
  - typename: we use it to inform the compiler T is a generic type.

    -
      ```
      template<typename T1, typename T2, typename T3>
      const T1 Add(const T2& a, const T3& b)
      {
          return a + b;
      }
      double value = Add<double, int, double>(2, 3.1);
      ```
- immediate-if
  ```
  template <typename T>
  T max(const &T a, const T& b)
  {
      return a > b ? a : b;
  }
  ```