Keyword: const
const

class Bank
{
    public:
        Money AccountBalance(int id) const;
        int Withdraw(int id, const Money &money);
    private:
        const unsigned int ACCOUNTS; //Not used
        const static unsigned int MAX_ACCOUNTS = 10;
};

//Not used
const int baz = 5;
int* const foo = &baz;
const int* bar = &baz;

• Constants can take many different forms...
Constant "Variable"

```cpp
class Bank {
    public:
        Money AccountBalance(int id) const;
        int Withdraw(int id, const Money &money);
    private:
        const unsigned int ACCOUNTS; // Not used
        const static unsigned int MAX_ACCOUNTS = 10;
};

// Not used
const int baz = 5;
int* const foo = &baz;
const int* bar = &baz;
```

- Data stored can not be changed
- Useful for naming literal values (e.g. PI=3.14...)
- When applied to objects, the object can only call Constant Member Functions
Constant Pointer

class Bank
{
    public:
        Money AccountBalance(int id) const;
        int Withdraw(int id, const Money &money);
    private:
        const unsigned int ACCOUNTS; //Not used
        const static unsigned int MAX_ACCOUNTS = 10;
};

//Not used
const int baz = 5;
int* const foo = &baz;
const int* bar = &baz;

• Pointer always points to the same location
• But data pointed to can change
• Useful for protecting buffer locations when passing to functions
Pointer to a Constant

class Bank
{
    public:
        Money AccountBalance(int id) const;
        int Withdraw(int id, const Money &money);
    private:
        const unsigned int ACCOUNTS; // Not used
        const static unsigned int MAX_ACCOUNTS = 10;
};

// Not used
const int baz = 5;
int* const foo = &baz;
const int* bar = &baz;

• Pointer can be updated to point to other addresses
• But data pointed to can not be changed
• Useful for marking read-only data when passing to functions
Constant Reference

class Bank
{
    public:
        Money AccountBalance(int id) const;
        int Withdraw(int id, const Money &money);
    private:
        const unsigned int ACCOUNTS; //Not used
        const static unsigned int MAX_ACCOUNTS = 10;
};

//Not used
const int baz = 5;
int* const foo = &baz;
const int* bar = &baz;

• Same effect as Pointer to a Constant
• But the syntax is as if the variable was passed by value
Constant Member Data

```cpp
class Bank
{
    public:
        Money AccountBalance(int id) const;
        int Withdraw(int id, const Money &money);
    private:
        const unsigned int ACCOUNTS; //Not used
        const static unsigned int MAX_ACCOUNTS = 10;
};
```

```cpp
//Not used
const int baz = 5;
int* const foo = &baz;
const int* bar = &baz;
```

• Same effect as Constant "Variable" except is applied to an object's scope

• Can only be initialized within the constructor within the initialization list

  - e.g. Bank::Bank() : ACCOUNTS(10) { //Constructor }
Constant Class Data

class Bank
{
    public:
        Money AccountBalance(int id) const;
        int Withdraw(int id, const Money &money);
    private:
        const unsigned int ACCOUNTS;  // Not used
        const static unsigned int MAX_ACCOUNTS = 10;
};

    // Not used
    const int baz = 5;
    int* const foo = &baz;
    const int* bar = &baz;

    • Same effect as Constant “Variable” except is applied to an classes' scope
    • This means you don't need an object to access it
      - e.g. Math::PI
Constant Member Functions

class Bank
{
  public:
    Money AccountBalance(int id) const;
    int Withdraw(int id, const Money &money);
  private:
    const unsigned int ACCOUNTS;  // Not used
    const static unsigned int MAX_ACCOUNTS = 10;
};

// Not used
const int baz = 5;
int* const foo = &baz;
const int* bar = &baz;

• Signals that the function does not modify any member data

• Useful in preventing accessor functions from accidentally modifying data