Analisis dan Desain Object-Oriented dengan C++

Pertemuan 13

Outline

- Using New Features of C++0x
  - Null Pointer Constant
  - Compile-Time Constant Expressions
  - Auto-Typed Variables
  - New **for** Loop
- Employing Object-Oriented Analysis and Design
  - Event Loops
  - The Application Programming Interface
  - Working with Driver Programs

1. Null Pointer Constant

Swapper.cpp

```
#include <iostream>

int main()
{
        int value1 = 12500;
        int value2 = 1700;
        int *pointer2 = nullptr;

        // give pointer the address of value2
        pointer2 = &value2;
        // dereference the pointer and assign to value1
        value1 = *pointer2;
        pointer2 = 0;

        std::cout << "value1 = " << value1 << "\n";

        return 0;
}
```

2. Compile-Time Constant Expressions

Circle.cpp

```cpp
#include <iostream>

// get an approximate value of PI
constexpr double getPi() {
       return (double) 22 / 7;
}

int main()
{
       float radius;

       std::cout << "Enter the radius of the circle: ";
       std::cin >> radius;

       // the area equals PI * the radius squared
       double area = getPi() * (radius * radius);

       std::cout << "\nCircle's area: " << area << "\n";

       return 0;
}
```

3. Auto-Typed Variables

Combat.cpp

```cpp
#include <iostream>

int main()
{
        // define character values
        auto strength;
        auto accuracy;
        auto dexterity;

        // define constants
        const auto maximum = 50;

        // get user input
        std::cout << “\nEnter strength (1-100): “;
        std::cin >> strength;

        std::cout << “\nEnter accuracy (1-50): “;
        std::cin >> accuracy;

        std::cout << “\nEnter dexterity (1-50): “;
        std::cin >> dexterity;

        // calculate character combat stats
        auto attack = strength * (accuracy / maximum);
        auto damage = strength * (dexterity / maximum);

        std::cout << “\nAttack rating: “ << attack << “\n”;
        std::cout << “Damage rating: “ << damage << “\n”;
}
```

4. New **for** Loop

NewForLoop.cpp

```
#include <iostream>

int main()
{
        int positions[5] = {4, 3, 10, 25, 8};

        for (int &p: positions)
        {
                p *= 3;
                std::cout << p << "\n";
        }
}
```

5. Event Loops

SimpleEvent.cpp

```
#include <iostream>

class Condition
{
       public:
                 Condition() { }
                 virtual ~Condition() {}
                 virtual void log() = 0;
};

class Normal : public Condition
{
       public:
                 Normal() { log(); }
                 virtual ~Normal() {}
                 virtual void log()
                 { std::cout << “Logging normal conditions ...\n”; }
};

class Error : public Condition
{
       public:
                 Error() { log(); }
                 virtual ~Error() {}
                 virtual void log() { std::cout << “Logging error!\n”; }
};

```

…

```
class Alarm : public Condition
{
        public:
                Alarm();
                virtual ~Alarm() {}
                virtual void warn() { std::cout << “Warning!\n”; }
                virtual void log() { std::cout << “General alarm log\n”; }
                virtual void call() = 0;
};

Alarm::Alarm()
{
        log();
        warn();
}

class FireAlarm : public Alarm
{
        public:
                FireAlarm() { log();};
                virtual ~FireAlarm() {}
                virtual void call() { std::cout<< “Calling fire department!\n”; }
                virtual void log() { std::cout << “Logging fire call\n”; }
};

```

…

```
int main()
{
    int input;
    int okay = 1;
    Condition *pCondition;
    while (okay)
    {
        std::cout << “(0) Quit (1) Normal (2) Fire: “;
        std::cin >> input;
        okay = input;
        switch (input)
        {
            case 0:
            break;
            case 1:
            pCondition = new Normal;
            delete pCondition;
            break;
            case 2:
            pCondition = new FireAlarm;
            delete pCondition;
            break;
            default:
            pCondition = new Error;
            delete pCondition;
            okay = 0;
            break;
        }
    }
    return 0;
}
```

6. The Application Programming Interface

PostMasterMessage.cpp

```
class PostMasterMessage : public MailMessage
{
        public:
                PostMasterMessage();
                PostMasterMessage(
                pAddress sender,
                pAddress recipient,
                pString subject,
                pDate creationDate);

                // other constructors here
                // remember to include copy constructor
                // as well as constructor from storage
                // and constructor from wire format
                // Also include constructors from other formats
                ~PostMasterMessage();
                pAddress& getSender() const;
                void setSender(pAddress&);
                // other member accessors
                // operator functions here, including operator equals
                // and conversion routines to turn PostMaster messages
                // into messages of other formats.

        private:
                pAddress sender;
                pAddress recipient;
                pString subject;
                pDate creationDate;
                pDate lastModDate;
                pDate receiptDate;
                pDate firstReadDate;
                pDate lastReadDate;
 };
```

7. Working with Driver Programs

Driver.cpp

```
#include <iostream>
#include <string.h>

typedef unsigned long pDate;

enum SERVICE { PostMaster, Interchange,
Gmail, Hotmail, AOL, Internet };

class String
{
        public:
                    // constructors
                    String();
                    String(const char *const);
                    String(const String&);
                    ~String();

                    // overloaded operators
                    char& operator[](int offset);
                    char operator[](int offset) const;
                    String operator+(const String&);
                    void operator+=(const String&);
                    String& operator=(const String&);
                    friend std::ostream& operator<<
                    (std::ostream& stream, String& newString);
                    // General accessors
                    int getLen() const { return len; }
                    const char* getString() const { return string; }
                    // static int constructorCount;

                    private:
                    String(int); // private constructor
                    char* string;
                    int len;
};

```

...

```
// default constructor creates string of 0 bytes
String::String()
{
        string = new char[1];
        string[0] = '\0';
        len = 0;
        // std::cout << "\tDefault string constructor\n";
        // constructorCount++;
}

// private (helper) constructor, used only by
// class functions for creating a new string of
// required size. Null filled.
String::String(int newLen)
{
        string = new char[newLen + 1];
        int i;
        for (i = 0; i <= newLen; i++)
                string[1] = '\0';
        len = newLen;
        // std::cout << "\tString(int) constructor\n";
        // constructorCount++;
}

// Converts a character array to a String
String::String(const char* const cString)
{
        len = strlen(cString);
        string = new char[len + 1];
        int i;
        for (i = 0; i < len; i++)
                string[i] = cString[i];
        string[len]='\0';
        // std::cout << "\tString(char*) constructor\n";
        // constructorCount++;
}

```

...

```
// copy constructor
String::String(const String &rhs)
{
       len = rhs.getLen();
       string = new char[len + 1];
       int i;
       for (i = 0; i < len; i++)
                  string[i] = rhs[i];
       string[len] = ‘\0’;
       // std::cout << “\tString(String&) constructor\n”;
       // constructorCount++;
}

// destructor, frees allocated memory
String::~String ()
{
       delete [] string;
       len = 0;
       // std::cout << “\tString destructor\n”;
}

String& String::operator=(const String &rhs)
{
       if (this == &rhs)
       return *this;
       delete [] string;
       len = rhs.getLen();
       string = new char[len + 1];
       int i;
       for (i = 0; i < len; i++)
                  string[i] = rhs[i];
       string[len] = ‘\0’;
       return *this;
       // std::cout << “\tString operator=\n”;
}

```

...

```
//non constant offset operator, returns
// reference to character so it can be changed
char &String::operator[](int offset)
{
      if (offset > len)
                 return string[len - 1];
      else
                 return string[offset];
}

// constant offset operator for use
// on const objects (see copy constructor!)
char String::operator[](int offset) const
{
      if (offset > len)
                 return string[len - 1];
      else
                 return string[offset];
}

// creates a new string by adding current
// string to rhs
String String::operator+(const String& rhs)
{
      int totalLen = len + rhs.getLen();
      String temp(totalLen);
      int i, j;
      for (i = 0; i < len; i++)
                 temp[i] = string[i];
      for (j = 0; j < rhs.getLen(); j++, i++)
                 temp[i] = rhs[j];
      temp[totalLen]='\0';
      return temp;
}

```

...

```
// changes current string, returns nothing
void String::operator+=(const String& rhs)
{
        int rhsLen = rhs.getLen();
        int totalLen = len + rhsLen;
        String temp(totalLen);
        int i, j;
        for (i = 0; i < len; i++)
                temp[i] = string[i];
        for ( j = 0; j < rhs.getLen(); j++, i++)
                temp[i] = rhs[i - len];
        temp[totalLen]='\0';
        *this = temp;
}

// int String::ConstructorCount = 0;

std::ostream& operator<<(std::ostream& stream,
String& newString)
{
        stream << newString.getString();
        return stream;
}

```

...

```
class pAddress
{
     public:
               pAddress(SERVICE newService,
               const String& newAddress,
               const String& newDisplay):
               service(newService),
               addressString(newAddress),
               displayString(newDisplay)
               {}
               // pAddress(String, String);
               // pAddress();
               // pAddress(const pAddress&);
               ~pAddress(){}
               friend std::ostream& operator<<(
               std::ostream& stream, pAddress& address);
               String& getDisplayString()
               { return displayString; }
     private:
               SERVICE service;
               String addressString;
               String displayString;
};

std::ostream& operator<<
( std::ostream& stream, pAddress& address)
{
     stream << address.getDisplayString();
     return stream;
}

```

...

```
class PostMasterMessage
{
      public:
                  // PostMasterMessage();

                  PostMasterMessage(const pAddress& newSender,
                  const pAddress& newRecipient,
                  const String& newSubject,
                  const pDate& newCreationDate);

                  ~PostMasterMessage(){}

                  void Edit(); // invokes editor on this message

                  pAddress& getSender() { return sender; }
                  pAddress& getRecipient() { return recipient; }
                  String& getSubject() { return subject; }
                  // void setSender(pAddress& );
                  // other member accessors

                  // operator functions here, including operator equals
                  // and conversion routines to turn PostMaster messages
                  // into messages of other formats.

      private:
                  pAddress sender;
                  pAddress recipient;
                  String subject;
                  pDate creationDate;
                  pDate lastModDate;
                  pDate receiptDate;
                  pDate firstReadDate;
                  pDate lastReadDate;
};

```

...

```
PostMasterMessage::PostMasterMessage(
        const pAddress& newSender,
        const pAddress& newRecipient,
        const String& newSubject,
        const pDate& newCreationDate):
        sender(newSender),
        recipient(newRecipient),
        subject(newSubject),
        creationDate(newCreationDate),
        lastModDate(newCreationDate),
        firstReadDate(0),
        lastReadDate(0)
{
        std::cout << “Postmaster message created. \n”;
}

void PostMasterMessage::Edit()
{
        std::cout << “Postmaster message edit function called\n”;
}


int main()
{
        pAddress sender(
        PostMaster, “james@ekzemplo.com”, “James”);
        pAddress recipient(
        PostMaster, “sharon@ekzemplo.com”,”Sharon”);
        PostMasterMessage postMasterMessage(
        sender, recipient, “Greetings”, 0);
        std::cout << “Message review... \n”;
        std::cout << “From:\t\t”
        << postMasterMessage.getSender() << “\n”;
        std::cout << “To:\t\t”
        << postMasterMessage.getRecipient() << “\n”;
        std::cout << “Subject:\t”
        << postMasterMessage.getSubject() << “\n”;
        return 0;
}
```

Tugas

- Tuliskanlah versi baru dari program Circle yang meminta jari-jari (radius) dan tinggi (height) dari silinder. Hitunglah volume nya menggunakan rumusan PI*(radius^2)*height.
- Tuliskanlah sebuah program menggunakan 4 fungsi overloaded square() yang melakukan perkalian bilangan dengan sendirinya dengan int, long, float, dan double sebagai parameter. Simpan hasil dari fungsi tersebut dengan menggunakan variable auto-type dan tampilkan hasilnya.
- Pada program SimpleEvent.cpp program, tambahkan pesan menggunakan "std::cout" pada setiap destructor untuk memastikan bahwa destructor telah dipanggil.
- Buatlah hirarkhi class pada "chess pieces" dengan base class pada setiap tipe move yang mungkin dan derived class pada setiap piece.