

# 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

```
1: #include <iostream>
2:
3: int main()
4: {
5:     int value1 = 12500;
6:     int value2 = 1700;
7:     int *pointer2 = nullptr;
8:
9:     // give pointer the address of value2
10:    pointer2 = &value2;
11:    // dereference the pointer and assign to value1
12:    value1 = *pointer2;
13:    pointer2 = 0;
14:
15:    std::cout << "value1 = " << value1 << "\n";
16:
17:    return 0;
18: }
```

## 2. Compile-Time Constant Expressions

### Circle.cpp

```
1: #include <iostream>
2:
3: // get an approximate value of PI
4: constexpr double getPi() {
5:     return (double) 22 / 7;
6: }
7:
8: int main()
9: {
10:     float radius;
11:
12:     std::cout << "Enter the radius of the circle: ";
13:     std::cin >> radius;
14:
15:     // the area equals PI * the radius squared
16:     double area = getPi() * (radius * radius);
17:
18:     std::cout << "\nCircle's area: " << area << "\n";
19:
20:     return 0;
21: }
```

# 3. Auto-Typed Variables

## Combat.cpp

```
1: #include <iostream>
2:
3: int main()
4: {
5:     // define character values
6:     auto strength;
7:     auto accuracy;
8:     auto dexterity;
9:
10:    // define constants
11:    const auto maximum = 50;
12:
13:    // get user input
14:    std::cout << "\nEnter strength (1-100): ";
15:    std::cin >> strength;
16:
17:    std::cout << "\nEnter accuracy (1-50): ";
18:    std::cin >> accuracy;
19:
20:    std::cout << "\nEnter dexterity (1-50): ";
21:    std::cin >> dexterity;
22:
23:    // calculate character combat stats
24:    auto attack = strength * (accuracy / maximum);
25:    auto damage = strength * (dexterity / maximum);
26:
27:    std::cout << "\nAttack rating: " << attack << "\n";
28:    std::cout << "Damage rating: " << damage << "\n";
29: }
```

# 4. New for Loop

## NewForLoop.cpp

```
1: #include <iostream>
2:
3: int main()
4: {
5:     int positions[5] = {4, 3, 10, 25, 8};
6:
7:     for (int &p: positions)
8:     {
9:         p *= 3;
10:        std::cout << p << "\n";
11:    }
12: }
```

# 5. Event Loops

## SimpleEvent.cpp

```
1: #include <iostream>
2:
3: class Condition
4: {
5:     public:
6:         Condition() { }
7:         virtual ~Condition() {}
8:         virtual void log() = 0;
9: };
10:
11: class Normal : public Condition
12: {
13:     public:
14:         Normal() { log(); }
15:         virtual ~Normal() {}
16:         virtual void log()
17:         { std::cout << "Logging normal conditions ...\n"; }
18: };
19:
20: class Error : public Condition
21: {
22:     public:
23:         Error() { log(); }
24:         virtual ~Error() {}
25:         virtual void log() { std::cout << "Logging error!\n"; }
26: };
27:
```

...

```
28: class Alarm : public Condition
29: {
30:     public:
31:         Alarm();
32:         virtual ~Alarm() {}
33:         virtual void warn() { std::cout << "Warning!\n"; }
34:         virtual void log() { std::cout << "General alarm log\n"; }
35:         virtual void call() = 0;
36: };
37:
38: Alarm::Alarm()
39: {
40:     log();
41:     warn();
42: }
43:
44: class FireAlarm : public Alarm
45: {
46:     public:
47:         FireAlarm() { log();};
48:         virtual ~FireAlarm() {}
49:         virtual void call() { std::cout<< "Calling fire department!\n"; }
50:         virtual void log() { std::cout << "Logging fire call\n"; }
51: };
52:
```



...

```
53: int main()
54: {
55:     int input;
56:     int okay = 1;
57:     Condition *pCondition;
58:     while (okay)
59:     {
60:         std::cout << "(0) Quit (1) Normal (2) Fire: ";
61:         std::cin >> input;
62:         okay = input;
63:         switch (input)
64:         {
65:             case 0:
66:                 break;
67:             case 1:
68:                 pCondition = new Normal;
69:                 delete pCondition;
70:                 break;
71:             case 2:
72:                 pCondition = new FireAlarm;
73:                 delete pCondition;
74:                 break;
75:             default:
76:                 pCondition = new Error;
77:                 delete pCondition;
78:                 okay = 0;
79:                 break;
80:         }
81:     }
82:     return 0;
83: }
```

# 6. The Application Programming Interface

## PostMasterMessage.cpp

```
1: class PostMasterMessage : public MailMessage
2: {
3:     public:
4:         PostMasterMessage();
5:         PostMasterMessage(
6:             pAddress sender,
7:             pAddress recipient,
8:             pString subject,
9:             pDate creationDate);
10:
11:         // other constructors here
12:         // remember to include copy constructor
13:         // as well as constructor from storage
14:         // and constructor from wire format
15:         // Also include constructors from other formats
16:         ~PostMasterMessage();
17:         pAddress& getSender() const;
18:         void setSender(pAddress&);
19:         // other member accessors
20:         // operator functions here, including operator equals
21:         // and conversion routines to turn PostMaster messages
22:         // into messages of other formats.
23:
24:     private:
25:         pAddress sender;
26:         pAddress recipient;
27:         pString subject;
28:         pDate creationDate;
29:         pDate lastModDate;
30:         pDate receiptDate;
31:         pDate firstReadDate;
32:         pDate lastReadDate;
33: };
```

# 7. Working with Driver Programs

## Driver.cpp

```
1: #include <iostream>
2: #include <string.h>
3:
4: typedef unsigned long pDate;
5:
6: enum SERVICE { PostMaster, Interchange,
7: Gmail, Hotmail, AOL, Internet };
8:
9: class String
10: {
11:     public:
12:         // constructors
13:         String();
14:         String(const char *const);
15:         String(const String&);
16:         ~String();
17:
18:         // overloaded operators
19:         char& operator[](int offset);
20:         char operator[](int offset) const;
21:         String operator+(const String&);
22:         void operator+=(const String&);
23:         String& operator=(const String&);
24:         friend std::ostream& operator<<
25:         (std::ostream& stream, String& newString);
26:         // General accessors
27:         int getLen() const { return len; }
28:         const char* getString() const { return string; }
29:         // static int constructorCount;
30:
31:     private:
32:         String(int); // private constructor
33:         char* string;
34:         int len;
35: };
36:
```

```
...
37: // default constructor creates string of 0 bytes
38: String::String()
39: {
40:     string = new char[1];
41:     string[0] = '\\0';
42:     len = 0;
43:     // std::cout << "\\tDefault string constructor\\n";
44:     // constructorCount++;
45: }
46:
47: // private (helper) constructor, used only by
48: // class functions for creating a new string of
49: // required size. Null filled.
50: String::String(int newLen)
51: {
52:     string = new char[newLen + 1];
53:     int i;
54:     for (i = 0; i <= newLen; i++)
55:         string[i] = '\\0';
56:     len = newLen;
57:     // std::cout << "\\tString(int) constructor\\n";
58:     // constructorCount++;
59: }
60:
61: // Converts a character array to a String
62: String::String(const char* const cString)
63: {
64:     len = strlen(cString);
65:     string = new char[len + 1];
66:     int i;
67:     for (i = 0; i < len; i++)
68:         string[i] = cString[i];
69:     string[len]='\\0';
70:     // std::cout << "\\tString(char*) constructor\\n";
71:     // constructorCount++;
72: }
73:
```

```
...
74: // copy constructor
75: String::String(const String &rhs)
76: {
77:     len = rhs.getLen();
78:     string = new char[len + 1];
79:     int i;
80:     for (i = 0; i < len; i++)
81:         string[i] = rhs[i];
82:     string[len] = '\\0';
83:     // std::cout << "\\tString(String&) constructor\\n";
84:     // constructorCount++;
85: }
86:
87: // destructor, frees allocated memory
88: String::~String ()
89: {
90:     delete [] string;
91:     len = 0;
92:     // std::cout << "\\tString destructor\\n";
93: }
94:
95: String& String::operator=(const String &rhs)
96: {
97:     if (this == &rhs)
98:         return *this;
99:     delete [] string;
100:    len = rhs.getLen();
101:    string = new char[len + 1];
102:    int i;
103:    for (i = 0; i < len; i++)
104:        string[i] = rhs[i];
105:    string[len] = '\\0';
106:    return *this;
107:    // std::cout << "\\tString operator=\\n";
108: }
109:
```

```
...
110: //non constant offset operator, returns
111: // reference to character so it can be changed
112: char &String::operator[](int offset)
113: {
114:     if (offset > len)
115:         return string[len - 1];
116:     else
117:         return string[offset];
118: }
119:
120: // constant offset operator for use
121: // on const objects (see copy constructor!)
122: char String::operator[](int offset) const
123: {
124:     if (offset > len)
125:         return string[len - 1];
126:     else
127:         return string[offset];
128: }
129:
130: // creates a new string by adding current
131: // string to rhs
132: String String::operator+(const String& rhs)
133: {
134:     int totalLen = len + rhs.getLen();
135:     String temp(totalLen);
136:     int i, j;
137:     for (i = 0; i < len; i++)
138:         temp[i] = string[i];
139:     for (j = 0; j < rhs.getLen(); j++, i++)
140:         temp[i] = rhs[j];
141:     temp[totalLen]='\0';
142:     return temp;
143: }
144:
```

...

```
145: // changes current string, returns nothing
146: void String::operator+=(const String& rhs)
147: {
148:     int rhsLen = rhs.getLen();
149:     int totalLen = len + rhsLen;
150:     String temp(totalLen);
151:     int i, j;
152:     for (i = 0; i < len; i++)
153:         temp[i] = string[i];
154:     for ( j = 0; j < rhs.getLen(); j++, i++)
155:         temp[i] = rhs[i - len];
156:     temp[totalLen]='\0';
157:     *this = temp;
158: }
159:
160: // int String::ConstructorCount = 0;
161:
162: std::ostream& operator<<(std::ostream& stream,
163: String& newString)
164: {
165:     stream << newString.getString();
166:     return stream;
167: }
168:
```

```
...
169: class pAddress
170: {
171:     public:
172:         pAddress(SERVICE newService,
173:                 const String& newAddress,
174:                 const String& newDisplay):
175:             service(newService),
176:             addressString(newAddress),
177:             displayString(newDisplay)
178:         {}
179:         // pAddress(String, String);
180:         // pAddress();
181:         // pAddress(const pAddress&);
182:         ~pAddress(){}
183:         friend std::ostream& operator<<(
184:             std::ostream& stream, pAddress& address);
185:         String& getDisplayString()
186:         { return displayString; }
187:     private:
188:         SERVICE service;
189:         String addressString;
190:         String displayString;
191: };
192:
193: std::ostream& operator<<
194: ( std::ostream& stream, pAddress& address)
195: {
196:     stream << address.getDisplayString();
197:     return stream;
198: }
199:
```



```
...
200: class PostMasterMessage
201: {
202:     public:
203:         // PostMasterMessage();
204:
205:         PostMasterMessage(const pAddress& newSender,
206:             const pAddress& newRecipient,
207:             const String& newSubject,
208:             const pDate& newCreationDate);
209:
210:         ~PostMasterMessage(){}
211:
212:         void Edit(); // invokes editor on this message
213:
214:         pAddress& getSender() { return sender; }
215:         pAddress& getRecipient() { return recipient; }
216:         String& getSubject() { return subject; }
217:         // void setSender(pAddress& );
218:         // other member accessors
219:
220:         // operator functions here, including operator equals
221:         // and conversion routines to turn PostMaster messages
222:         // into messages of other formats.
223:
224:     private:
225:         pAddress sender;
226:         pAddress recipient;
227:         String subject;
228:         pDate creationDate;
229:         pDate lastModDate;
230:         pDate receiptDate;
231:         pDate firstReadDate;
232:         pDate lastReadDate;
233: };
234:
```

```
235: PostMasterMessage::PostMasterMessage(  
236:     const pAddress& newSender,  
237:     const pAddress& newRecipient,  
238:     const String& newSubject,  
239:     const pDate& newCreationDate):  
240:     sender(newSender),  
241:     recipient(newRecipient),  
242:     subject(newSubject),  
243:     creationDate(newCreationDate),  
244:     lastModDate(newCreationDate),  
245:     firstReadDate(0),  
246:     lastReadDate(0)  
247: {  
248:     std::cout << "Postmaster message created. \n";  
249: }  
250:  
251: void PostMasterMessage::Edit()  
252: {  
253:     std::cout << "Postmaster message edit function called\n";  
254: }  
255:  
256:  
257: int main()  
258: {  
259:     pAddress sender(  
260:         PostMaster, "james@ekzemplo.com", "James");  
261:     pAddress recipient(  
262:         PostMaster, "sharon@ekzemplo.com", "Sharon");  
263:     PostMasterMessage postMasterMessage(  
264:         sender, recipient, "Greetings", 0);  
265:     std::cout << "Message review... \n";  
266:     std::cout << "From:\t\t"  
267:     << postMasterMessage.getSender() << "\n";  
268:     std::cout << "To:\t\t"  
269:     << postMasterMessage.getRecipient() << "\n";  
270:     std::cout << "Subject:\t"  
271:     << postMasterMessage.getSubject() << "\n";  
272:     return 0;  
273: }
```

# 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.