Chapter 1

Computer Science and Programming
Figure 1.1:
Figure 1.2:
Figure 1.3:
Efficiency and complexity

Conceptual and formal models

Levels of abstraction

Efficiency and complexity

Figure 1.4:
<table>
<thead>
<tr>
<th>Rational</th>
<th>Integer</th>
<th>Real</th>
<th>Complex</th>
</tr>
</thead>
<tbody>
<tr>
<td>3/4 * 8/9</td>
<td>1.285 * 57</td>
<td>3.14 * 6.023</td>
<td>(3 + 5i) * (2 - 7i)</td>
</tr>
<tr>
<td>2/3</td>
<td>73,245</td>
<td>18.91222</td>
<td>41 - 11i</td>
</tr>
</tbody>
</table>

Figure 1.5:
int operator * (int x, int y)
// pre: post:
{
}

Figure 1.6:
Figure 1.7:
Figure 1.8:
Figure 1.9:
Chapter 2

C++ Programs: Form and Function
```cpp
#include <iostream>
using namespace std;

// traditional first program
// author: Owen Astrachan, 02/22/99
void Hello()
{
    cout << "Hello World" << endl;
}

int main()
{
    Hello();
    return 0;
}
```

Figure 2.1:
void Sing(string person)
{
    cout << "Happy birthday to you" << endl;
    cout << "Happy birthday to you" << endl;
    cout << "Happy birthday dear " << person << endl;
    cout << "Happy birthday to you" << endl;
    cout << endl;
}

int main()
{
    Sing("Grace");
    Sing("Alan");
    ...
}

Figure 2.2:
CHAPTER 2. C++ PROGRAMS: FORM AND FUNCTION

```cpp
void HadA(string animal)
{
    "pig"
    cout << "and on his farm he had a " << animal << ", ";
    EiEiO();
}

void WithA(string noise)
{
    "oink"
    cout << "With a " << noise << " " << noise << " here" << endl;
    ...
}

void Pig()
{
    Refrain();
    HadA("pig");
    WithA("oink");
    Refrain();
}
```

Figure 2.3:
void
Verse(string animal, string noise)
{
  "pig"  "oink"
  Refrain();
  Had(animal);
  WithA(noise);
  Refrain();
}

int main()
{
  Verse("pig","oink");
  ...
}

Figure 2.4:
Chapter 3

Program Design and Implementation
int main()
{
    string animal;  // 1
    string noise;   // 2
    cout << "Enter the name of an animal ";
    cin >> animal;
    cout << "Enter noise that a " << animal << " makes ";
    cin >> noise;
    cout << endl;
    Verse(animal,noise);
    return 0;
}

int main()
{
    string animal;    // 1
    string noise;     // 2
    cout << "Enter the name of an animal ";
    cin >> animal;
    cout << "Enter noise that a " << animal << " makes ";
    cin >> noise;
    cout << endl;
    Verse(animal,noise);
    return 0;
}
Figure 3.2:
Figure 3.3:
Figure 3.4:
void SlicePrice(int radius, double price) {
    ... 
    ... 
    cout << "$" < 3.14159*radius*radius/price
    ... 
}

int main() {
    int radius;  
    double price; 
    ... 
    SlicePrice(radius, price); 
}
Figure 3.6:
Figure 3.7:
```cpp
#include "gballoon.h"

int main()
{
    Balloon b;
    b.Ascend(30);
    return 0;
}
```

```cpp
#include "gballoon.h"

void Balloon::Ascend(int height)
{
    cout << endl;
    cout << "***** (Height = ";
    cout << myAltitude << ");
    ...
Figure 3.9:
Chapter 4

Control, Functions, and Classes
Before execution

```
int amount;
87
```

int quarters;
3

After execution

```
amount = amount - quarters*25;
87 - 3*25
```

```
int amount;
12
```

Figure 4.1:
Figure 4.2:
Chapter 5

Iteration with Programs and Classes
if (test) {
    statement list;
} next statement;

while (test) {
    statement list;
} next statement;

Figure 5.1:
<table>
<thead>
<tr>
<th>Product</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>6</th>
<th>24</th>
<th>120</th>
<th>720</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

Figure 5.2:
Figure 5.3:
```cpp
int limit = sqrt(n) + 1;
int divisor = 3;
while (divisor <= limit)
{
    if (n % divisor == 0)
    {
        return false;
    }
    divisor += 2;
}

k = low;
while (k <= high)
{
    if (IsPrime(k))
    {
        cout << k << endl;
        numPrimes += 1;
    }
    k += 1;
}
```

Figure 5.4:
Chapter 6

Classes, Iterators, and Patterns
Figure 6.1:
Function prototype/header, formal parameters

```
void GiveQuiz(string name, int & correct, int & total)
```

Figure 6.2:

Function call, arguments

```
GiveQuiz(student, correctCount, total);  
```

Owen 6 10
Figure 6.3:
Chapter 7

Class Interfaces, Design, and Implementation
Figure 7.1:
cos(\alpha) = X / \text{step size}
\sin(\alpha) = Y / \text{step size}

Figure 7.2:
Brownian motion: 1,024 steps of unit length

Figure 7.3:
Brownian motion: 1,024 steps of unit length

Figure 7.4:
Figure 7.5:
distance = 3d
Chapter 8

Arrays, Data, and Random Access
Figure 8.1:
Figure 8.2:
CHAPTER 8. ARRAYS, DATA, AND RANDOM ACCESS

tvector<int> diceStats(2*DICE_SIDES+1);

Figure 8.3:
switch (sum) { 
    case 2:
        twos += 1;
        break;
    case 3:
        threes += 1;
        break;
    case 4:
        fours += 1;
        break;
    case 5:
        fives += 1;
        break;
    case 6:
        sixes += 1;
        break;
    case 7:
        sevens += 1;
        break;
    case 8:
        eights += 1;
        break;
    }

    diceStats[sum]++;
Add D to vector maintained in sorted order

Figure 8.5:
Sequential search

one guess
two guesses
three guesses
four guesses
five guesses
six guesses

Binary search

one guess
two guesses
three guesses
four guesses
five guesses
six guesses

(low, high, high, low)

Figure 8.6:
Figure 8.7:
Chapter 9

Strings, Streams, and Operators
CHAPTER 9. STRINGS, STREAMS, AND OPERATORS

Figure 9.1:
Figure 9.2:
Figure 9.3:
Chapter 10

Recursion, Lists, and Matrices
From main()
Print(1478)

PrintDigit(8);

PrintDigit(4);

PrintDigit(7);

PrintDigit(1);

PrintDigit(1);

PrintDigit(14);

PrintDigit(147);

number 1478

number 147

number 14

number 1

Figure 10.1:
Figure 10.2:
Figure 10.3:
Figure 10.4:
Figure 10.5:
Figure 10.6:
CHAPTER 10. RECURSION, LISTS, AND MATRICES

Figure 10.7:
Figure 10.8:
Figure 10.9:
Figure 10.10:
Figure 10.11:
<table>
<thead>
<tr>
<th>3-neighborhood</th>
<th>5-neighborhood</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 12 28</td>
<td>10 12 12 10 10</td>
</tr>
<tr>
<td>10 28 25</td>
<td>10 10 12 32 32</td>
</tr>
<tr>
<td>25 32 32</td>
<td>25 10 28 18 18</td>
</tr>
<tr>
<td></td>
<td>25 25 32 32 18</td>
</tr>
<tr>
<td></td>
<td>32 32 32 25 25</td>
</tr>
</tbody>
</table>

Figure 10.12:
Figure 10.13:
Chapter 11

Sorting, Templates, and
Generic Programming
CHAPTER 11. SORTING, TEMPLATES, AND GENERIC PROGRAMMING

Figure 11.1:
Figure 11.2:
Figure 11.3:
Figure 11.4:
Desired properties of vector, partitioned around pivot

<table>
<thead>
<tr>
<th>Elements &lt;= X</th>
<th>X</th>
<th>Elements &gt;= X</th>
</tr>
</thead>
<tbody>
<tr>
<td>First</td>
<td>Pivot</td>
<td>Last</td>
</tr>
</tbody>
</table>

After several iterations, partially re-arranged, more elements to process

<table>
<thead>
<tr>
<th>X</th>
<th>&lt;= X</th>
<th>&gt; X</th>
<th>???</th>
</tr>
</thead>
<tbody>
<tr>
<td>First</td>
<td>p</td>
<td>k</td>
<td>Last</td>
</tr>
</tbody>
</table>

All elements processed

<table>
<thead>
<tr>
<th>X</th>
<th>&lt;= X</th>
<th>&gt; X</th>
</tr>
</thead>
<tbody>
<tr>
<td>First</td>
<td>p</td>
<td>Last</td>
</tr>
</tbody>
</table>

Final configuration after swapping first element into pivot location

<table>
<thead>
<tr>
<th>&lt;= X</th>
<th>X</th>
<th>&gt; X</th>
</tr>
</thead>
<tbody>
<tr>
<td>First</td>
<td>Pivot</td>
<td>Last</td>
</tr>
</tbody>
</table>

```cpp
if (a[k] <= piv)
{
    p++;
    Swap(a[k],a[p])
}
Swap(a[p],a[first]);
return p;
```

Figure 11.5:
nonzeros | examined | ????

nonZeroIndex  k

Figure 11.6: