Dijkstra’s Control Flows

- Simple Sequence
Dijkstra’s Control Flows

- Simple Sequence
- Decision

Dijkstra’s Control Flows

- Simple Sequence
- Decision
- Loop
```c
int main()
{
    int i;
    i = 10;
    cout << "i = " << i;
    return 0;
}
```
int main()
{
    int i;
    i = 10;
    cout << "i = " << i;
    return 0;
}
# Simple Sequence

```c
int main()
{
    int i;
    i = 10;
    cout << "i = " << i;
    return 0;
}
```

# Decisions

```c
// output a complex number
cout << '(' << real;
if (imaginary < 0) {
    cout << " - j";
    cout << imaginary;
} else {
    cout << " + j";
    cout << imaginary;
}
cout << ')';
```
Decisions

// output a complex number
cout << '(' << real;
if (imaginary < 0) {
    cout << " - j";
    cout << -imaginary;
} else {
    cout << " + j";
    cout << imaginary;
}
cout << ')';

Decisions

// output a complex number
cout << '(' << real;
if (imaginary < 0) {
    cout << " - j";
    cout << -imaginary;
} else {
    cout << " + j";
    cout << imaginary;
}
cout << ')';
Decisions

// output a complex number
cout << '(' << real;
if (imaginary < 0) {
    cout << " - j";
    cout << -imaginary;
}
else {
    cout << " + j";
    cout << imaginary;
}
cout << ')';
Decisions

// output a complex number
cout << '(' << real;
if (imaginary < 0) {
    cout << " - j";
    cout << -imaginary;
} else {
    cout << " + j";
    cout << imaginary;
}
cout << ')';
Decisions

// output a complex number
cout << '(' << real;
if (imaginary < 0) {
    cout << " - j";
    cout << -imaginary;
}
else {
    cout << " + j";
    cout << imaginary;
}
cout << ');};
Decisions

// output a complex number
cout << '(' << real;
if (imaginary < 0) {
    cout << " - j";
    cout << -imaginary;
} else {
    cout << " + j";
    cout << imaginary;
}
cout << ')';

Single Clause Decision

// test for real root
double radical = b*b - 4*a*c;
if (radical > 0) {
    cout << "the roots are real\n";
} else {
    cout << "Proceeding to next test\n";
}
Loops

```cpp
int i = 0;
while (i <= 10){
    cout << i << "\t";
    cout << i*i << "\n";
    i++;
}
cout << "That was 11 squares.\n";
```

Loops

```cpp
int i = 0;
while (i <= 10){
    cout << i << "\t";
    cout << i*i << "\n";
    i++;
}
cout << "That was 11 squares.\n";
```
int i = 0;
while (i <= 10)
{
    cout << i << "\t";
    cout << i*i << "\n";
    i++;
}
cout << "That was 11 squares.";
Loops

```cpp
int i = 0;
while (i <= 10)
{
    cout << i << ":\t";
    cout << i*i << "\n";
    i++;
}
cout << "That was 11 squares."
```

Loops

```cpp
int i = 0;
while (i <= 10)
{
    cout << i << ":\t";
    cout << i*i << "\n";
    i++;
}
cout << "That was 11 squares."
```
Loops

int i = 0;
while (i <= 10){
    cout << i << "\t";
    cout << i*i << '\n';
    i++;
}
cout << "That was 11 squares.";

Loops

int i = 0;
while (i <= 10){
    cout << i << "\t";
    cout << i*i << '\n';
    i++;
}
cout << "That was 11 squares.";
Compound Control
Non-Hierarchial Control

Non-Hierarchial Control
Program Control

© Copyright 2000, 2004 by
michael bruce-lockhart
All rights reserved. No copies may be made
without the express permission of the author.

the TeachingMachine video series