What should you teach in an introductory programming course?

Some simple answers...

1. Teach programming language syntax
2. Teach how to design programs

Is there more?
What should you teach in an introductory programming course?

Some simple answers...

1. Teach programming language syntax
2. Teach how to design programs
What should you teach in an introductory programming course?

Some simple answers...

1. Teach programming language syntax
2. Teach how to design programs

Is there more?
Some questions we must ask from time to time..

- Is the nature of programming changing as a subject?
  - Programming is becoming more pervasive. Phones, cars, vacuum cleaners, cameras, all contain processors.
  - Computational ideas are appearing everywhere: biological evolution, social phenomenon, understanding text...

- Is the nature of students changing?
  - Students have many distractions: cellphones/media, finance jobs, IAS aspirations..
  - Attention span of students is small: You have to work hard to get their attention, and work harder to hold it..
Some questions we must ask from time to time..

- Is the nature of programming changing as a subject?

Programming is becoming more pervasive. Phones, cars, vacuum cleaners, cameras, all contain processors. Computational ideas are appearing everywhere: biological evolution, social phenomenon, understanding text...

- Is the nature of students changing?

Students have many distractions: cellphones/media, finance jobs, IAS aspirations.. Attention span of students is small: You have to work hard to get their attention, and work harder to hold it..
Some questions we must ask from time to time..

▶ Is the nature of programming changing as a subject?

Programming is becoming more pervasive. Phones, cars, vacuum cleaners, cameras, all contain processors.
Some questions we must ask from time to time..

► Is the nature of programming changing as a subject?

Programming is becoming more pervasive. Phones, cars, vacuum cleaners, cameras, all contain processors.

Computational ideas are appearing everywhere: biological evolution, social phenomenon, understanding text...
Some questions we must ask from time to time..

► Is the nature of programming changing as a subject?

Programming is becoming more pervasive. Phones, cars, vacuum cleaners, cameras, all contain processors

Computational ideas are appearing everywhere: biological evolution, social phenomenon, understanding text...

► Is the nature of students changing?

Students have many distractions: cellphones/media, finance jobs, IAS aspirations..
Attention span of students is small: You have to work hard to get their attention, and work harder to hold it..
Some questions we must ask from time to time..

► Is the nature of programming changing as a subject?

Programming is becoming more pervasive. Phones, cars, vacuum cleaners, cameras, all contain processors.

Computational ideas are appearing everywhere: biological evolution, social phenomenon, understanding text...

► Is the nature of students changing?

Students have many distractions: cellphones/media, finance jobs, IAS aspirations..
Some questions we must ask from time to time..

- Is the nature of programming changing as a subject?

  Programming is becoming more pervasive. Phones, cars, vacuum cleaners, cameras, all contain processors

  Computational ideas are appearing everywhere: biological evolution, social phenomenon, understanding text...

- Is the nature of students changing?

  Students have many distractions: cellphones/media, finance jobs, IAS aspirations..

  Attention span of students is small:
Some questions we must ask from time to time..

➤ Is the nature of programming changing as a subject?

Programming is becoming more pervasive. Phones, cars, vacuum cleaners, cameras, all contain processors

Computational ideas are appearing everywhere: biological evolution, social phenomenon, understanding text...

➤ Is the nature of students changing?

Students have many distractions: cellphones/media, finance jobs, IAS aspirations..

Attention span of students is small:

You have to work hard to get their attention, and work harder to hold it..
Some strategic measures

- Take programming examples from different domains: science, technology, art, ... to excite a wider range of students.

- Standard approach: programming examples drawn from basic numerical processing and text processing.

- May not interest all students, say those not majoring in CS.

- Use graphical input and output, not just text input/output.

- Natural medium of communication, in the era of video games and touch screen-based cell phones.

- Many interesting programming examples can be attempted, with very little effort.

Empowerment and excitement.
Some strategic measures

- Take programming examples from different domains: science, technology, art, ... to excite wider range of students
Some strategic measures

- Take programming examples from different domains: science, technology, art, ... to excite wider range of students
  - Standard approach: programming examples drawn from basic numerical processing and text processing.
Some strategic measures

- Take programming examples from different domains: science, technology, art, ... to excite wider range of students
  - Standard approach: programming examples drawn from basic numerical processing and text processing.
  - May not interest all students, say those not majoring in CS.

- Use graphical input and output, not just text input output.
  - Natural medium of communication, in the era of video games and touch screen based cell phones.
  - Many interesting programming examples can be attempted, with very little effort.

Empowerment and excitement
Some strategic measures

- Take programming examples from different domains: science, technology, art, ... to excite a wider range of students
  - Standard approach: programming examples drawn from basic numerical processing and text processing.
  - May not interest all students, say those not majoring in CS.
- Use graphical input and output, not just text input output.
Some strategic measures

- Take programming examples from different domains: science, technology, art, ... to excite wider range of students
  - Standard approach: programming examples drawn from basic numerical processing and text processing.
  - May not interest all students, say those not majoring in CS.
- Use graphical input and output, not just text input output.
  - Natural medium of communication, in the era of video games and touch screen based cell phones.
Some strategic measures

- Take programming examples from different domains: science, technology, art, ... to excite wider range of students
  - Standard approach: programming examples drawn from basic numerical processing and text processing.
  - May not interest all students, say those not majoring in CS.
- Use graphical input and output, not just text input output.
  - Natural medium of communication, in the era of video games and touch screen based cell phones.
  - Many interesting programming examples can be attempted, with very little effort.
Some strategic measures

- Take programming examples from different domains: science, technology, art, ... to excite a wider range of students
  - Standard approach: programming examples drawn from basic numerical processing and text processing.
  - May not interest all students, say those not majoring in CS.
- Use graphical input and output, not just text input output.
  - Natural medium of communication, in the era of video games and touch screen based cell phones.
  - Many interesting programming examples can be attempted, with very little effort.

Empowerment and excitement
Grab students’ attention on day 1 and hold it
Grab students’ attention on day 1 and hold it

▶ Grabbing attention:

- Honestly try to SELL computer programming as an exciting and important activity in the first lecture.
- Give demonstrations of good programs, especially the programs that students might be able to write by the end of the course.
- Arrange your material so that interesting exercises are possible from day 1.
- Do not lose the tempo!
Grab students’ attention on day 1 and hold it

- Grabbing attention:
  - Honestly try to SELL computer programming as an exciting and important activity in the first lecture.
Grab students’ attention on day 1 and hold it

- Grabbing attention:
  - Honestly try to SELL computer programming as an exciting and important activity in the first lecture.
  - Give demonstrations of good programs, especially the programs that students might be able to write by the end of the course.

- Holding attention:
  - Arrange your material so that interesting exercises are possible from day 1.
  - Do not lose the tempo!
Grab students’ attention on day 1 and hold it

Grabbing attention:
- Honestly try to SELL computer programming as an exciting and important activity in the first lecture.
- Give demonstrations of good programs, especially the programs that students might be able to write by the end of the course.

Holding attention:
Grab students’ attention on day 1 and hold it

▶ Grabbing attention:
  ▶ Honestly try to SELL computer programming as an exciting and important activity in the first lecture.
  ▶ Give demonstrations of good programs, especially the programs that students might be able to write by the end of the course.

▶ Holding attention:
  ▶ Arrange your material so that interesting exercises are possible from day 1. Do not lose the tempo!
The approach followed in the book

- Take programming examples from different areas: math, engineering, art, operations research, ...
- A teaching tool: Simplecpp
  - Main element of Simplecpp: 2D graphics library
    - Turtle graphics
    - Coordinate based graphics
    - Elementary animation
    - Elementary graphical input
    - Very easy to use; "alternative to <iostream>
- Additional elements
  - "New statement": main program
  - "New statement": repeat
- A general pedagogical principle: find and work out convincing real examples of anything you want to teach.
The approach followed in the book

- Take programming examples from different areas: math, engineering, art, operations research, ...

Additional elements
- "New statement": main program
- "New statement": repeat

A general pedagogical principle: find and work out convincing real examples of anything you want to teach.
The approach followed in the book

- Take programming examples from different areas: math, engineering, art, operations research, ...
- A teaching tool: Simplecpp
The approach followed in the book

- Take programming examples from different areas: math, engineering, art, operations research, ...
- A teaching tool: Simplecpp

Main element of Simplecpp: 2D graphics library

- *Turtle* graphics
- Coordinate based graphics
- Elementary animation
- Elementary graphical input

Very easy to use; “alternative to `<iostream>`”

Additional elements

- “New statement”: main program
- “New statement”: repeat

A general pedagogical principle: find and work out convincing real examples of anything you want to teach.
The approach followed in the book

- Take programming examples from different areas: math, engineering, art, operations research, ...
- A teaching tool: Simplecpp

Main element of Simplecpp: 2D graphics library

- *Turtle* graphics
- Coordinate based graphics
- Elementary animation
- Elementary graphical input
- Very easy to use; “alternative to `<iostream>`"
The approach followed in the book

- Take programming examples from different areas: math, engineering, art, operations research, ...
- A teaching tool: Simplecpp

Main element of Simplecpp: 2D graphics library

- Turtle graphics
- Coordinate based graphics
- Elementary animation
- Elementary graphical input
- Very easy to use; “alternative to <iostream>”

Additional elements

- “New statement”: main_program
- “New statement”: repeat
The approach followed in the book

- Take programming examples from different areas: math, engineering, art, operations research, ...
- A teaching tool: Simplecpp

Main element of Simplecpp: 2D graphics library
- *Turtle* graphics
- Coordinate based graphics
- Elementary animation
- Elementary graphical input
- Very easy to use; “alternative to `<iostream>`”

Additional elements
- “New statement”: `main_program`
- “New statement”: `repeat`

- A general pedagogical principle: find and work out convincing real examples of anything you want to teach.
Outline

- Turtle graphics
  - Introduced in lecture 1 of the course
  - main_program and repeat
- Coordinate based graphics
- Recursion using graphics
- Other issues
  - Program design vis-a-vis algorithm design.
  - Object oriented programming
  - Software engineering
  - Standard library
- Concluding remarks
- Demos
Turtle graphics

Invented in 1960s by Seymour Pappert, as part of the Logo programming language for teaching programming to children.
Turtle graphics

Invented in 1960s by Seymour Pappert, as part of the Logo programming language for teaching programming to children.

**Turtle:** a symbolic animal that lives on the screen.
Turtle graphics

Invented in 1960s by Seymour Pappert, as part of the Logo programming language for teaching programming to children.

**Turtle:** a symbolic animal that lives on the screen.

Moves as per commands issued by the program.
Turtle graphics

Invented in 1960s by Seymour Pappert, as part of the Logo programming language for teaching programming to children.

Turtle: a symbolic animal that lives on the screen.

Moves as per commands issued by the program.

Has a pen, which draws on the screen as the turtle moves.
Turtle graphics

Invented in 1960s by Seymour Pappert, as part of the Logo programming language for teaching programming to children.

Turtle: a symbolic animal that lives on the screen.

Moves as per commands issued by the program.

Has a pen, which draws on the screen as the turtle moves.

Goal of turtle graphics: Draw interesting pictures on the screen.
A simple program fragment

turtleSim();  // Start turtle graphics

forward(100);
right(90);

forward(100);
right(90);

forward(100);
right(90);

forward(100);
A simple program fragment

turtleSim(); // Start turtle graphics

forward(100);
right(90);

forward(100);
right(90);

forward(100);
right(90);

forward(100);
right(90);

forward(100);

What does this draw?
A simple program fragment

turtleSim(); // Start turtle graphics

forward(100);
right(90);

forward(100);
right(90);

forward(100);
right(90);

forward(100);

What does this draw?

This program can be discussed in the first lecture.
The real day 1 program

```cpp
#include <simplecpp>
main_program{
    turtleSim();
    forward(100); wait(0.5); right(90); wait(0.5);
    forward(100); wait(0.5); right(90); wait(0.5);
    forward(100); wait(0.5); right(90); wait(0.5);
    forward(100); wait(0.5); right(90); wait(0.5);
    forward(100);
    wait(5);
}
```
The real day 1 program

```cpp
#include <simplecpp>
main_program{
    turtleSim();
    forward(100); wait(0.5); right(90); wait(0.5);
    forward(100); wait(0.5); right(90); wait(0.5);
    forward(100); wait(0.5); right(90); wait(0.5);
    forward(100); wait(0.5); right(90); wait(0.5);
    forward(100);
    wait(5);
}
```

▶ `<simplecpp>`: includes graphics functionality. Also includes “using namespace std;”, and `<iostream>`. 
The real day 1 program

```cpp
#include <simplecpp>
main_program{
  turtleSim();
  forward(100); wait(0.5); right(90); wait(0.5);
  forward(100); wait(0.5); right(90); wait(0.5);
  forward(100); wait(0.5); right(90); wait(0.5);
  forward(100); wait(0.5); right(90); wait(0.5);
  forward(100);
  wait(5);
}
```

- `<simplecpp>`: includes graphics functionality. Also includes “using namespace std;”, and `<iostream>`.
- `main_program`: No need to explain `int main()`.
The real day 1 program

```cpp
#include <simplecpp>
main_program{
    turtleSim();
    forward(100); wait(0.5); right(90); wait(0.5);
    forward(100); wait(0.5); right(90); wait(0.5);
    forward(100); wait(0.5); right(90); wait(0.5);
    forward(100); wait(0.5); right(90); wait(0.5);
    forward(100);
    wait(5);
}
```

- `<simplecpp>`: includes graphics functionality. Also includes “using namespace std;”, and `<iostream>`.
- `main_program`: No need to explain `int main()`.
- Students must eventually understand namespaces, iostreams, `int main`, but not on day 1.
The real day 1 program

```cpp
#include <simplecpp>

main_program{
    turtleSim();
    forward(100); wait(0.5); right(90); wait(0.5);
    forward(100); wait(0.5); right(90); wait(0.5);
    forward(100); wait(0.5); right(90); wait(0.5);
    forward(100);
    wait(5);
}
```

- `<simplecpp>`: includes graphics functionality. Also includes “using namespace std;”, and `<iostream>`.
- `main_program`: No need to explain `int main()`.
- Students must eventually understand namespaces, iostreams, `int main`, but not on day 1.
- `main_program` is a C++ preprocessor macro, so it automatically gets translated to `int main()`.
The second program for day 1

```c++
#include <simplecpp>

main_program{
  turtleSim();
  repeat(4){
    forward(100); right(90);
  }
}
```
#include <simplecpp>

main_program{
    turtleSim();
    repeat(4){
        forward(100); right(90);
    }
}

"New statement": repeat

repeat (count) { statements to be repeated }
Another day 1 program

```
main_program{
    turtleSim();
    repeat(10){
        repeat(4){
            forward(100); right(90);
        }
        right(10);
    }
    wait(10); 
}
```

"What do you think it does?"

This is what I ask students. Most figure it out!

Why? Because it is an interesting challenge!
Another day 1 program

```cpp
main_program{
    turtleSim();
    repeat(10){
        repeat(4){
            forward(100); right(90);
        }
        right(10);
    }
    wait(10);
}
```

“What do you think it does?”
Another day 1 program

main_program{
    turtleSim();
    repeat(10){
        repeat(4){
            forward(100); right(90);
        }
        right(10);
    }
    wait(10);
}

“What do you think it does?”

This is what I ask students. Most figure it out!
Another day 1 program

main_program{
    turtleSim();
    repeat(10){
        repeat(4){
            forward(100); right(90);
        }
        right(10);
    }
    wait(10);
}

“What do you think it does?”

This is what I ask students. Most figure it out!

Why? Because it is an interesting challenge!
What have students learnt on day 1?

1. Control flow
2. Elementary iteration
3. Basic ideas of syntax, spaces, indentation
4. Importance of observing patterns in what is to be accomplished, and expressing them using `repeat`. You should not write 100 statements to draw a 100-sided program!

Very important activity while designing programs!

Homework on day 1: draw chessboard, draw circles (as limit of n-sided polygon). Draw 5-sided star.

Need some high school geometry. However, most programming needs some domain knowledge.

Students are happy to do this because they can see interesting things happening.
What have students learnt on day 1?

1. Control flow

...
What have students learnt on day 1?

1. Control flow
2. Elementary iteration

Very important activity while designing programs!

Homework on day 1: draw chessboard, draw circles (as limit of n sided polygon). Draw 5 sided star.

Need some high school geometry. However, most programming needs some domain knowledge.

Students are happy to do this because they can see interesting things happening.
What have students learnt on day 1?

1. Control flow
2. Elementary iteration
3. Basic ideas of syntax, spaces, indentation

Importance of observing patterns in what is to be accomplished, and expressing them using repeat. You should not write 100 statements to draw a 100 sided program! Very important activity while designing programs!

Homework on day 1: draw chessboard, draw circles (as limit of n sided polygon). Draw 5 sided star. Need some high school geometry. However, most programming needs some domain knowledge. Students are happy to do this because they can see interesting things happening.
What have students learnt on day 1?

1. Control flow
2. Elementary iteration
3. Basic ideas of syntax, spaces, indentation
4. Importance of observing patterns in what is to be accomplished, and expressing them using `repeat`
What have students learnt on day 1?

1. Control flow
2. Elementary iteration
3. Basic ideas of syntax, spaces, indentation
4. Importance of observing patterns in what is to be accomplished, and expressing them using `repeat`

You should not write 100 statements to draw a 100 sided program!
What have students learnt on day 1?

1. Control flow
2. Elementary iteration
3. Basic ideas of syntax, spaces, indentation
4. Importance of observing patterns in what is to be accomplished, and expressing them using repeat

You should not write 100 statements to draw a 100 sided program!

Very important activity while designing programs!
What have students learnt on day 1?

1. Control flow
2. Elementary iteration
3. Basic ideas of syntax, spaces, indentation
4. Importance of observing patterns in what is to be accomplished, and expressing them using `repeat`

You should not write 100 statements to draw a 100 sided program!

Very important activity while designing programs!

Homework on day 1: draw chessboard, draw circles (as limit of n sided polygon). Draw 5 sided star.
What have students learnt on day 1?

1. Control flow
2. Elementary iteration
3. Basic ideas of syntax, spaces, indentation
4. Importance of observing patterns in what is to be accomplished, and expressing them using repeat

You should not write 100 statements to draw a 100 sided program!

Very important activity while designing programs!

Homework on day 1: draw chessboard, draw circles (as limit of n sided polygon). Draw 5 sided star.

Need some high school geometry. However, most programming needs some domain knowledge.
What have students learnt on day 1?

1. Control flow
2. Elementary iteration
3. Basic ideas of syntax, spaces, indentation
4. Importance of observing patterns in what is to be accomplished, and expressing them using repeat

You should not write 100 statements to draw a 100 sided program!

Very important activity while designing programs!

Homework on day 1: draw chessboard, draw circles (as limit of n sided polygon). Draw 5 sided star.

Need some high school geometry. However, most programming needs some domain knowledge.

Students are happy to do this because they can see interesting things happening.
Not losing the tempo on day 2

Introduction must be followed by discussion of computer hardware. Then data types and variables. Drill assignments are possible, but how to catch the imagination?

```
main_program{
  int i = 1;
  turtleSim();
  repeat(10){
    forward(i*10); right(90);
    forward(i*10); right(90);
    i = i + 1;
  }
}
```

“What does this do?” Reassignment of variables is a difficult idea to understand. Students want to know why this program works, and so take the time to think about it.
Not losing the tempo on day 2

Introduction must be followed by discussion of computer hardware. Then data types and variables.

main_program{
    int i = 1;
    turtleSim();
    repeat(10){
        forward(i*10); right(90);
        forward(i*10); right(90);
        i = i + 1;
    }
}

“What does this do?”

Reassignment of variables is a difficult idea to understand. Students want to know why this program works, and so take the time to think about it.
Not losing the tempo on day 2

Introduction must be followed by discussion of computer hardware. Then data types and variables. Drill assignments are possible, but how to catch the imagination?

```c
main_program{
    int i = 1;
    turtleSim();
    repeat(10){
        forward(i*10); right(90);
        forward(i*10); right(90);
        i = i + 1;
    }
}
```

"What does this do?"
Reassignment of variables is a difficult idea to understand. Students want to know why this program works, and so take the time to think about it.
Not losing the tempo on day 2

Introduction must be followed by discussion of computer hardware. Then data types and variables. Drill assignments are possible, but how to catch the imagination?

```c
main_program{
    int i = 1;
    turtleSim();
    repeat(10){
        forward(i*10); right(90);
        forward(i*10); right(90);
        i = i + 1;
    }
}
```

“What does this do?”

Reassignment of variables is a difficult idea to understand. Students want to know why this program works, and so take the time to think about it.
Not losing the tempo on day 2

Introduction must be followed by discussion of computer hardware. Then data types and variables. Drill assignments are possible, but how to catch the imagination?

```c
main_program{
    int i = 1;
    turtleSim();
    repeat(10){
        forward(i*10); right(90);
        forward(i*10); right(90);
        i = i + 1;
    }
}
```

“What does this do?”

Reassignment of variables is a difficult idea to understand. Students want to know why this program works, and so take the time to think about it.
Coordinate based graphics

Creating graphical objects

Circle \( c_1(200,225,25) \);
// \( c_1 \) = circle at \( (200,225) \), \( r = 25 \)

Rectangle \( r_1(200,200,50,70) \);
// center coordinates, width, height

Manipulating graphical objects

\( r_1 \).right(45); // rotate 45 degrees
\( c_1 \).move(10,20); // move by 10,20 in x,y directions

Elementary graphical input

\[ \text{int } z = \text{getClick}; \]
// \( z \) = \( 65536 \times \text{x coordinate of cursor} + \text{y coordinate} \)

Excitement value:
Elementary animation can be done.

Educational goals achieved:
Implicit introduction to constructor
and member access syntax. Repetition will be needed to do
interesting animations.
Coordinate based graphics

Creating graphical objects

Circle c1(200,225,25);
   // c1 = circle at (200,225), r = 25
Rectangle r1(200,200,50,70);
   // center coordinates, width, height
Coordinate based graphics

Creating graphical objects

Circle $c_1(200,225,25)$;
   // $c_1 = \text{circle at (200,225), r = 25}$
Rectangle $r_1(200,200,50,70)$;
   // center coordinates, width, height

Manipulating graphical objects

$r_1.\text{right}(45)$;  // rotate 45 degrees
$c_1.\text{move}(10,20)$;  // move by 10,20 in x,y directions
Coordinate based graphics

Creating graphical objects

Circle c1(200,225,25);
  // c1 = circle at (200,225), r = 25
Rectangle r1(200,200,50,70);
  // center coordinates, width, height

Manipulating graphical objects

r1.right(45);  // rotate 45 degrees
cl.move(10,20);  // move by 10,20 in x,y directions

Elementary graphical input

int z = getClick();
  // z = 65536 * x coordinate of cursor + y coordinate
Coordinate based graphics

Creating graphical objects

Circle c1(200,225,25);
    // c1 = circle at (200,225), r = 25
Rectangle r1(200,200,50,70);
    // center coordinates, width, height

Manipulating graphical objects

r1.right(45);  // rotate 45 degrees
r1.move(10,20); // move by 10,20 in x,y directions

Elementary graphical input

int z = getClick();
    // z = 65536 * x coordinate of cursor + y coordinate

Excitement value: Elementary animation can be done.
Coordinate based graphics

Creating graphical objects

Circle $c_1(200,225,25)$;
   // $c_1 = $circle at (200,225), $r = 25$
Rectangle $r_1(200,200,50,70)$;
   // center coordinates, width, height

Manipulating graphical objects

$r_1$.right(45);  // rotate 45 degrees
$c_1$.move(10,20);  // move by 10,20 in $x,y$ directions

Elementary graphical input

int $z = \text{getClick}()$;
   // $z = 65536 \times x$ coordinate of cursor + $y$ coordinate

Excitement value: Elementary animation can be done.  
Educational goals acheived: Implicit introduction to constructor and member access syntax. Repetition will be needed to do interesting animations.
Graphics using recursion
Graphics using recursion

Drawing a mathematical tree
Graphics using recursion

Drawing a mathematical tree
- Recursion is “visible”
Graphics using recursion

Drawing a mathematical tree
  ▶ Recursion is “visible”

Drawing of a realistic tree
Graphics using recursion

Drawing a mathematical tree
  - Recursion is “visible”

Drawing of a realistic tree
  - Perhaps recursion is more applicable than we thought?
Graphics as example of inheritance

Sprite:
Class of all graphics objects

Rectangle, Circle, Polygon,...:
Subclasses of Sprite.

Turtle:
Subclass of Polygon.

Sprite holds attributes common to all graphics objects.

Data members: on-screen position coordinates. Orientation. Colour...

Function members: move, right, left

Rectangle class holds

Data members: width and height

Function member: function that actually draws the rectangle on the screen.
Graphics as example of inheritance

**Sprite**: Class of all graphics objects
Graphics as example of inheritance

**Sprite:** Class of all graphics objects

**Rectangle, Circle, Polygon,**…: Subclasses of Sprite.
Graphics as example of inheritance

**Sprite:** Class of all graphics objects

**Rectangle, Circle, Polygon,...:** Subclasses of Sprite.

**Turtle:** Subclass of Polygon.
Graphics as example of inheritance

**Sprite:** Class of all graphics objects

**Rectangle, Circle, Polygon,....:** Subclasses of Sprite.

**Turtle:** Subclass of Polygon.

Sprite holds attributes common to all graphics objects.
Graphics as example of inheritance

**Sprite:** Class of all graphics objects

**Rectangle, Circle, Polygon,...:** Subclasses of Sprite.

**Turtle:** Subclass of Polygon.

Sprite holds attributes common to all graphics objects.

- Data members: on-screen position coordinates. Orientation. Colour...
Graphics as example of inheritance

**Sprite:** Class of all graphics objects

**Rectangle, Circle, Polygon,...:** Subclasses of Sprite.

**Turtle:** Subclass of Polygon.

Sprite holds attributes common to all graphics objects.
- Data members: on-screen position coordinates. Orientation. Colour...
- Function members: move, right, left
Graphics as example of inheritance

**Sprite:** Class of all graphics objects

**Rectangle, Circle, Polygon,**...: Subclasses of Sprite.

**Turtle:** Subclass of Polygon.

Sprite holds attributes common to all graphics objects.

▶ Data members: on-screen position coordinates. Orientation. Colour...

▶ Function members: move, right, left

Rectangle class holds
Graphics as example of inheritance

**Sprite**: Class of all graphics objects

**Rectangle, Circle, Polygon,....**: Subclasses of Sprite.

**Turtle**: Subclass of Polygon.

Sprite holds attributes common to all graphics objects.
- Data members: on-screen position coordinates. Orientation. Colour...
- Function members: move, right, left

Rectangle class holds
- Data members: width and height
Graphics as example of inheritance

**Sprite**: Class of all graphics objects

**Rectangle, Circle, Polygon,**...: Subclasses of Sprite.

**Turtle**: Subclass of Polygon.

Sprite holds attributes common to all graphics objects.
- Data members: on-screen position coordinates. Orientation. Colour...
- Function members: move, right, left

Rectangle class holds
- Data members: width and height
- Function member: function that actually draws the rectangle on the screen.
Other uses of graphics

- Visualizing output of programs. N-body simulation, Gravitational simulation.
- Visualization is important also because it can help discover programming errors more quickly.
Other uses of graphics


Visualizing output of programs. N-body simulation, Gravitational simulation.

Visualization is important also because it can help discover programming errors more quickly.
Other uses of graphics

- Visualizing output of programs. N-body simulation, Gravitational simulation.
Other uses of graphics

- Visualizing output of programs. N-body simulation, Gravitational simulation.
- Visualization is important also because it can help discover programming errors more quickly.
Some traditional issues in introductory programming
Some traditional issues in introductory programming

- Program design vs. algorithm design
- How much software engineering should you teach?
- How much methodology/OOP?
- How much history of computing?
- How much hardware?
- How much of STL?
Program design vs. Algorithm design

Writing a program when you know how to solve the problem using paper and pencil.

Algorithm design: Deciding what to calculate to solve a problem.

Main focus: Program design

However, Recursion is covered well.

Many good algorithms are described:

- Root finding, curve fitting
- Sorting including mergesort
- Solving linear systems of equations
- Gravitational simulation
- Backtracking, e.g. 8 queens.
- A few advanced algorithms to pique the interest for projects, further study.

Coding some of these is left for exercises.
Program design vs. Algorithm design

Program design: Writing a program when you know how to solve the problem using paper and pencil.
Program design vs. Algorithm design

**Program design:** Writing a program when you know how to solve the problem using paper and pencil.

**Algorithm design:** Deciding what to calculate to solve a problem.
Program design vs. Algorithm design

Program design: Writing a program when you know how to solve the problem using paper and pencil.

Algorithm design: Deciding what to calculate to solve a problem.

Main focus: Program design

- However, Recursion is covered well.
- Many good algorithms are described:
  - Root finding, curve fitting
  - Sorting including mergesort
  - Solving linear systems of equations
  - Gravitational simulation
  - Backtracking, e.g. 8 queens.
  - A few advanced algorithms to pique the interest for projects, further study.

Coding some of these is left for exercises.
Program design vs. Algorithm design

Program design: Writing a program when you know how to solve the problem using paper and pencil.
Algorithm design: Deciding what to calculate to solve a problem.

Main focus: Program design

However, Recursion is covered well.
Program design vs. Algorithm design

**Program design:** Writing a program when you know how to solve the problem using paper and pencil.

**Algorithm design:** Deciding what to calculate to solve a problem.

Main focus: Program design

However, **Recursion** is covered well.

Many good algorithms are described

- Root finding, curve fitting
- Sorting including mergesort
- Solving linear systems of equations
- Gravitational simulation
- Backtracking, e.g. 8 queens.
- A few advanced algorithms to pique the interest for projects, further study.
- Coding some of these is left for exercises.
Program design vs. Algorithm design

Program design: Writing a program when you know how to solve the problem using paper and pencil.
Algorithm design: Deciding what to calculate to solve a problem.

Main focus: Program design

However, Recursion is covered well.

Many good algorithms are described
  ▶ Root finding, curve fitting
Program design vs. Algorithm design

Program design: Writing a program when you know how to solve the problem using paper and pencil.
Algorithm design: Deciding what to calculate to solve a problem.

Main focus: Program design

However, Recursion is covered well.

Many good algorithms are described
  ▶ Root finding, curve fitting
  ▶ Sorting including mergesort
Program design vs. Algorithm design

Program design: Writing a program when you know how to solve the problem using paper and pencil.
Algorithm design: Deciding what to calculate to solve a problem.

Main focus: Program design

However, Recursion is covered well.

Many good algorithms are described
  - Root finding, curve fitting
  - Sorting including mergesort
  - Solving linear systems of equations
  - Gravitational simulation
  - Backtracking, e.g. 8 queens.
  - A few advanced algorithms to pique the interest for projects, further study.
  - Coding some of these is left for exercises.
Program design vs. Algorithm design

**Program design:** Writing a program when you know how to solve the problem using paper and pencil.

**Algorithm design:** Deciding what to calculate to solve a problem.

Main focus: Program design

However, **Recursion** is covered well.

Many good algorithms are described

- Root finding, curve fitting
- Sorting including mergesort
- Solving linear systems of equations
- Gravitational simulation
Program design vs. Algorithm design

**Program design:** Writing a program when you know how to solve the problem using paper and pencil.

**Algorithm design:** Deciding what to calculate to solve a problem.

Main focus: Program design

However, **Recursion** is covered well.

Many good algorithms are described

- Root finding, curve fitting
- Sorting including mergesort
- Solving linear systems of equations
- Gravitational simulation
- Backtracking, e.g. 8 queens.

A few advanced algorithms to pique the interest for projects, further study. Coding some of these is left for exercises.
Program design vs. Algorithm design

**Program design:** Writing a program when you know how to solve the problem using paper and pencil.

**Algorithm design:** Deciding what to calculate to solve a problem.

Main focus: Program design

However, **Recursion** is covered well.

Many good algorithms are described

- Root finding, curve fitting
- Sorting including mergesort
- Solving linear systems of equations
- Gravitational simulation
- Backtracking, e.g. 8 queens.
- A few advanced algorithms to pique the interest for projects, further study.
Program design vs. Algorithm design

**Program design**: Writing a program when you know how to solve the problem using paper and pencil.

**Algorithm design**: Deciding what to calculate to solve a problem.

Main focus: Program design

However, **Recursion** is covered well.

Many good algorithms are described

- Root finding, curve fitting
- Sorting including mergesort
- Solving linear systems of equations
- Gravitational simulation
- Backtracking, e.g. 8 queens.
- A few advanced algorithms to pique the interest for projects, further study.

Coding some of these is left for exercises.
Programming methodologies

Some introductory programming courses teach OOP on day 1. My strategy: Pose progressively harder programming problems to the student, and teach whatever is needed to solve those. Need for OOP can be made to evolve. The book has substantial discussion, but not at the beginning.

Chapter on “inheritance based design”.

▶ A substantial application developed earlier without inheritance is rewritten with inheritance. New implementation is easier to extend. A non-trivial extension is carried out.
Programming methodologies

Some introductory programming courses teach OOP on day 1.
Some introductory programming courses teach OOP on day 1.

My strategy: Pose progressively harder programming problems to the student, and teach whatever is needed to solve those.
Programming methodologies

Some introductory programming courses teach OOP on day 1.

My strategy: Pose progressively harder programming problems to the student, and teach whatever is needed to solve those.

Need for OOP can be made to evolve. The book has substantial discussion, but not at the beginning.
Some introductory programming courses teach OOP on day 1.

My strategy: Pose progressively harder programming problems to the student, and teach whatever is needed to solve those.

Need for OOP can be made to evolve. The book has substantial discussion, but not at the beginning.

Chapter on “inheritance based design”.
Programming methodologies

Some introductory programming courses teach OOP on day 1.

My strategy: Pose progressively harder programming problems to the student, and teach whatever is needed to solve those.

Need for OOP can be made to evolve. The book has substantial discussion, but not at the beginning.

Chapter on “inheritance based design”.

- A substantial application developed earlier without inheritance is rewritten with inheritance.
Some introductory programming courses teach OOP on day 1.

My strategy: Pose progressively harder programming problems to the student, and teach whatever is needed to solve those.

Need for OOP can be made to evolve. The book has substantial discussion, but not at the beginning.

Chapter on “inheritance based design”.

- A substantial application developed earlier without inheritance is rewritten with inheritance.
- New implementation is easier to extend. A non-trivial extension is carried out.
Software engineering
Software engineering

Writing specs: Must be stressed. The book has some discussion. Specs for most non-trivial programs have been carefully developed.
Writing specs: Must be stressed. The book has some discussion. Specs for most non-trivial programs have been carefully developed.

Invariants/program proving: Substantial discussion. Necessary because students usually try a few cases and declare their program to be correct.
Software engineering

Writing specs: Must be stressed. The book has some discussion. Specs for most non-trivial programs have been carefully developed.

Invariants/program proving: Substantial discussion. Necessary because students usually try a few cases and declare their program to be correct.

Testing: Some discussion of how to generate test cases. Also some discussion of code coverage.
Libraries, especially STL

String, Vector, Map classes covered in depth. A baby version of String class is developed to illustrate idea of class design + memory management. Priority queue class is used in chapter on simulation.
Libraries, especially STL

String, Vector, Map classes covered in depth.
Libraries, especially STL

String, Vector, Map classes covered in depth.

A baby version of String class is developed to illustrate idea of class design + memory management.
Libraries, especially STL

String, Vector, Map classes covered in depth.

A baby version of String class is developed to illustrate idea of class design + memory management.

Priority queue class is used in chapter on simulation.
Unusual/Advanced topics

- Lambda expressions (C++ 11)
- Memory management, including reference counting
- Simulation. Gravitational simulation, airport simulation. Dijkstra’s algorithm as a simulation. A general framework is developed.
Demos

Could be shown on day 1 of the course.
Demos

Could be shown on day 1 of the course.
- Plate design
Demos

Could be shown on day 1 of the course.

► Plate design

► Line Fitting
Demos

Could be shown on day 1 of the course.

- Plate design
- Line Fitting
- Button based turtle controller
Demos

Could be shown on day 1 of the course.

- Plate design
- Line Fitting
- Button based turtle controller
- Tree drawing
Demos

Could be shown on day 1 of the course.

- Plate design
- Line Fitting
- Button based turtle controller
- Tree drawing
- Formula layout
Demos

Could be shown on day 1 of the course.

- Plate design
- Line Fitting
- Button based turtle controller
- Tree drawing
- Formula layout
- Gravitational Simulation

All require less than 250 lines of code. Many less than 50.
Demos

Could be shown on day 1 of the course.

- Plate design
- Line Fitting
- Button based turtle controller
- Tree drawing
- Formula layout
- Gravitational Simulation
- Airport simulation
Demos

Could be shown on day 1 of the course.

- Plate design
- Line Fitting
- Button based turtle controller
- Tree drawing
- Formula layout
- Gravitational Simulation
- Airport simulation

All require less than 250 lines of code. Many less than 50.
Concluding remarks

"If you want to build a ship, don’t drum up people together to collect wood and don’t assign them tasks and work, but rather teach them to long for the endless immensity of the sea."

– Antoine de Saint-Exupery

"When you want to teach children to think, you begin by treating them seriously when they are little, giving them responsibilities, talking to them candidly, providing privacy and solitude for them, and making them readers and thinkers of significant thoughts from the beginning. That’s if you want to teach them to think."

– Bertrand Russell

Taking students seriously: explaining the course goals in the language and using images that they understand.
Concluding remarks

- Must excite the imagination of students and then challenge them.
Concluding remarks

- Must excite the imagination of students and then challenge them.

“If you want to build a ship, don’t drum up people together to collect wood and don’t assign them tasks and work, but rather teach them to long for the endless immensity of the sea.”

– Antoine de Saint-Exupery
Concluding remarks

- Must excite the imagination of students and then challenge them.

“If you want to build a ship, don’t drum up people together to collect wood and don’t assign them tasks and work, but rather teach them to long for the endless immensity of the sea.”

– Antoine de Saint-Exupery

“When you want to teach children to think, you begin by treating them seriously when they are little, giving them responsibilities, talking to them candidly, providing privacy and solitude for them, and making them readers and thinkers of significant thoughts from the beginning. That’s if you want to teach them to think.”

– Bertrand Russell
Concluding remarks

▶ Must excite the imagination of students and then challenge them.

“If you want to build a ship, don’t drum up people together to collect wood and don’t assign them tasks and work, but rather teach them to long for the endless immensity of the sea.”

– Antoine de Saint-Exupery

“When you want to teach children to think, you begin by treating them seriously when they are little, giving them responsibilities, talking to them candidly, providing privacy and solitude for them, and making them readers and thinkers of significant thoughts from the beginning. That’s if you want to teach them to think.”

– Bertrand Russell

Taking students seriously: explaining the course goals in the language and using images that they understand.
Concluding remarks (contd.)

Simplecpp graphics is very elementary. Even primary school children understand turtle graphics – Logo experience.

Pictorially observing the effect of iteration, recursion drives home the concepts effortlessly.

New learning must be integrated with old learning wherever possible: ask problems from physics and math learnt in junior college. Or even problems related to art. There is great satisfaction in actually using the math and science you have learnt.

Book contains a lot of applications which will help students for doing projects. Preface describes which parts should be covered in 1 or 2 semester courses.

Transparencies for lectures will soon appear on McGraw Hill homepage and my homepage.
Concluding remarks (contd.)

- Simplecpp graphics is very elementary. Even primary school children understand turtle graphics – Logo experience.
Concluding remarks (contd.)

- Simplecpp graphics is very elementary. Even primary school children understand turtle graphics – Logo experience.
- Pictorially observing the effect of iteration, recursion drives home the concepts effortlessly.
Concluding remarks (contd.)

- Simplecpp graphics is very elementary. Even primary school children understand turtle graphics – Logo experience.
- Pictorially observing the effect of iteration, recursion drives home the concepts effortlessly.
- New learning must be integrated with old learning wherever possible: ask problems from physics and math learnt in junior college. Or even problems related to art.

There is great satisfaction in actually using the math and science you have learnt.

Book contains a lot of applications which will help students for doing projects. Preface describes which parts should be covered in 1 or 2 semester courses.

Transparencies for lectures will soon appear on McGraw Hill homepage and my homepage.
Concluding remarks (contd.)

- Simplecpp graphics is very elementary. Even primary school children understand turtle graphics – Logo experience.
- Pictorially observing the effect of iteration, recursion drives home the concepts effortlessly.
- New learning must be integrated with old learning wherever possible: ask problems from physics and math learnt in junior college. Or even problems related to art.

There is great satisfaction in actually using the math and science you have learnt.
Concluding remarks (contd.)

▶ Simplecpp graphics is very elementary. Even primary school children understand turtle graphics – Logo experience.
▶ Pictorially observing the effect of iteration, recursion drives home the concepts effortlessly.
▶ New learning must be integrated with old learning wherever possible: ask problems from physics and math learnt in junior college. Or even problems related to art.

There is great satisfaction in actually using the math and science you have learnt.

▶ Book contains a lot of applications which will help students for doing projects. Preface describes which parts should be covered in 1 or 2 semester courses.
Concluding remarks (contd.)

- Simplecpp graphics is very elementary. Even primary school children understand turtle graphics – Logo experience.
- Pictorially observing the effect of iteration, recursion drives home the concepts effortlessly.
- New learning must be integrated with old learning wherever possible: ask problems from physics and math learnt in junior college. Or even problems related to art.

There is great satisfaction in actually using the math and science you have learnt.

- Book contains a lot of applications which will help students for doing projects. Preface describes which parts should be covered in 1 or 2 semester courses.
- Transparencies for lectures will soon appear on McGraw Hill homepage and my homepage.
Some applications may appear to be difficult. But please read the relevant portions of the book – you may decide they are not hard after all.

It is possible to skip the "difficult" sections if your course is short.

Comments and suggestions on the book will be greatly appreciated.

We can have an electronic discussion group.

Thank you!!!
Some applications may appear to be difficult. But please read the relevant portions of the book – you may decide they are not hard after all.
Concluding remarks (contd.)

- Some applications may appear to be difficult. But please read the relevant portions of the book – you may decide they are not hard after all.
- It is possible to skip the “difficult” sections if your course is short.
Concluding remarks (contd.)

- Some applications may appear to be difficult. But please read the relevant portions of the book – you may decide they are not hard after all.
- It is possible to skip the “difficult” sections if your course is short.
- Comments and suggestions on the book will be greatly appreciated.
Concluding remarks (contd.)

- Some applications may appear to be difficult. But please read the relevant portions of the book – you may decide they are not hard after all.
- It is possible to skip the “difficult” sections if your course is short.
- Comments and suggestions on the book will be greatly appreciated.
- We can have an electronic discussion group.
Concluding remarks (contd.)

- Some applications may appear to be difficult. But please read the relevant portions of the book – you may decide they are not hard after all.
- It is possible to skip the “difficult” sections if your course is short.
- Comments and suggestions on the book will be greatly appreciated.
- We can have an electronic discussion group.

Thank you!!!