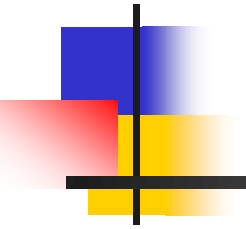
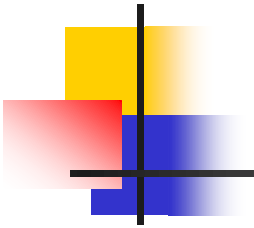


Introduction to

Object Oriented Programming (Continued)

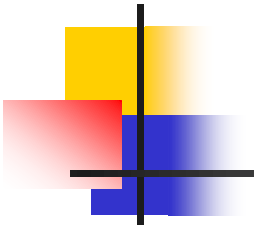
Cats



Objectives

You will be able to:

- Write and use classes with multiple member variables.
- Use the "this" pointer in methods.
- Use "const" correctly in various places within a method declaration.
- Understand #pragma once
- Create objects dynamically.
- Use dynamically created objects in a C++ program.
- Get keyboard input from a user in a C++ program.



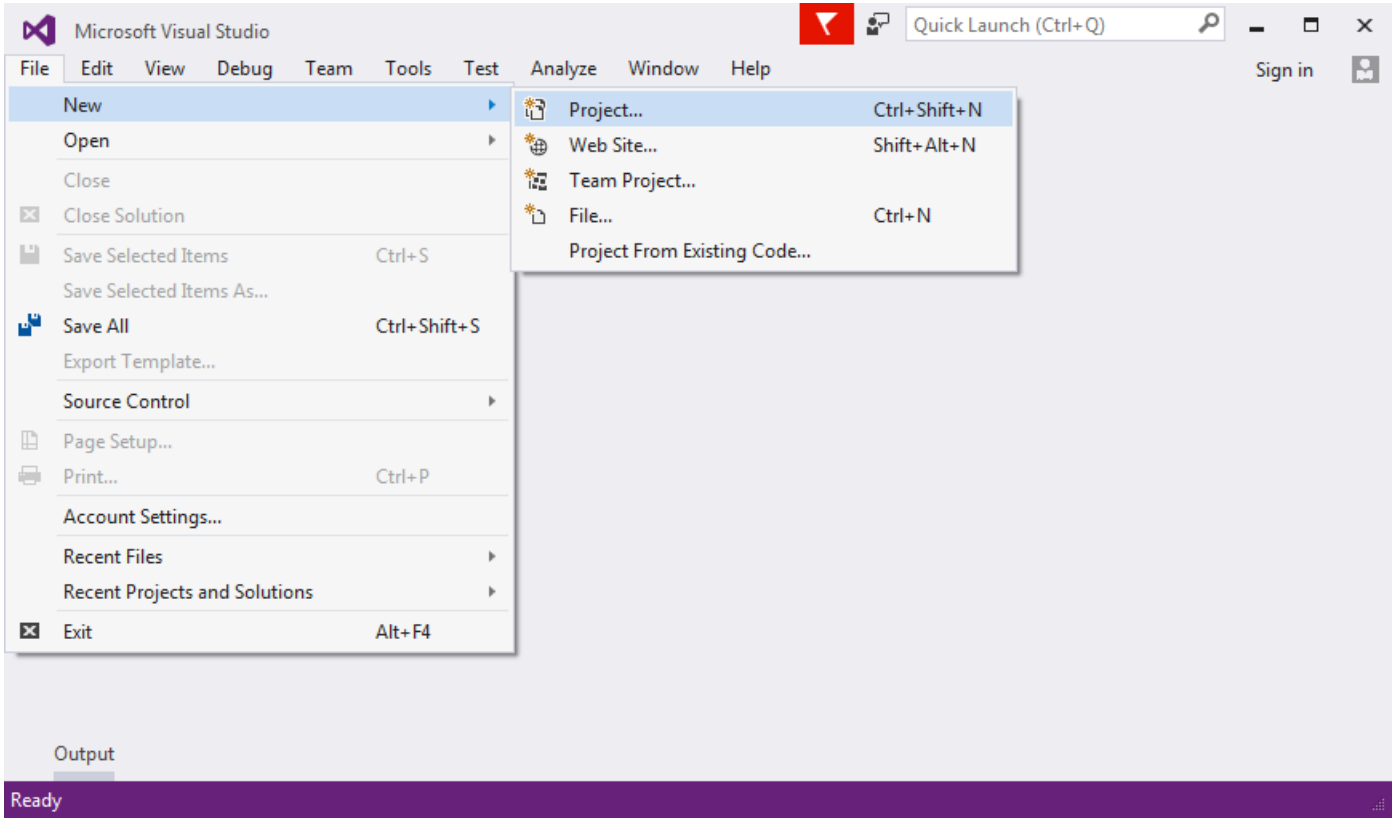
Class Cat

- Let's create a class to represent cats.
  - Perhaps for use in a veterinarian's office.

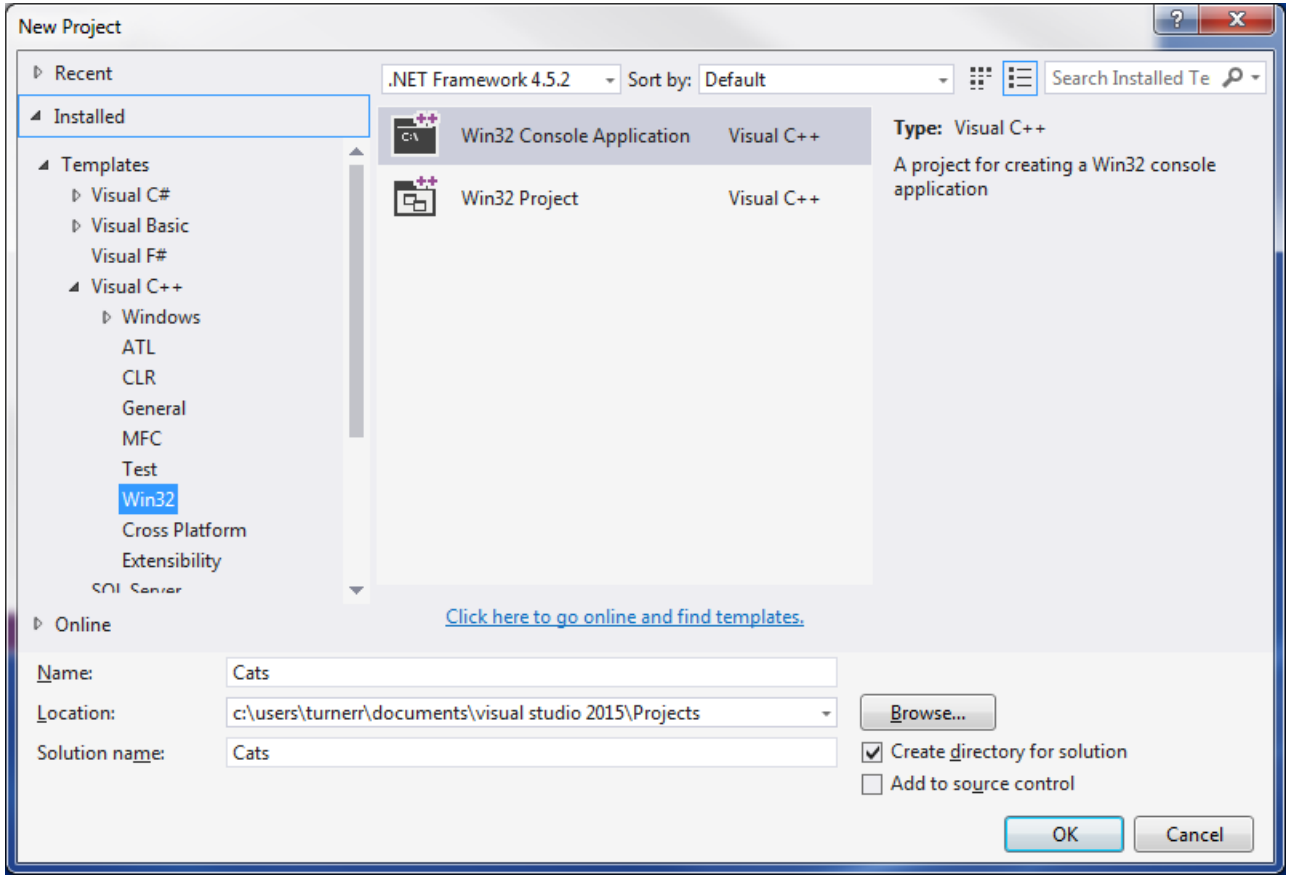
- Each cat has:
  - Name
  - Weight
  - Date of birth

New Project

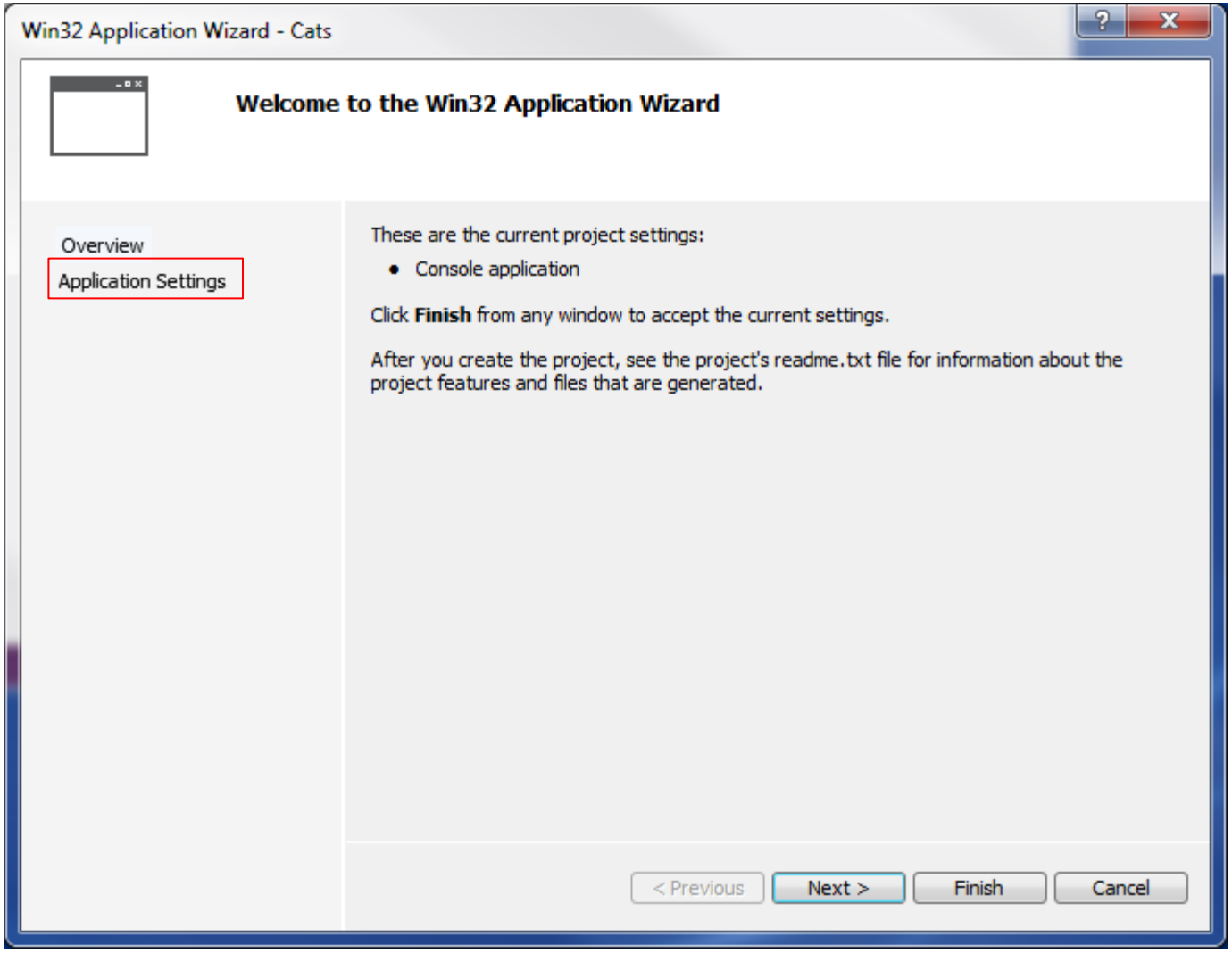
Create a new C++ console project in Visual Studio.



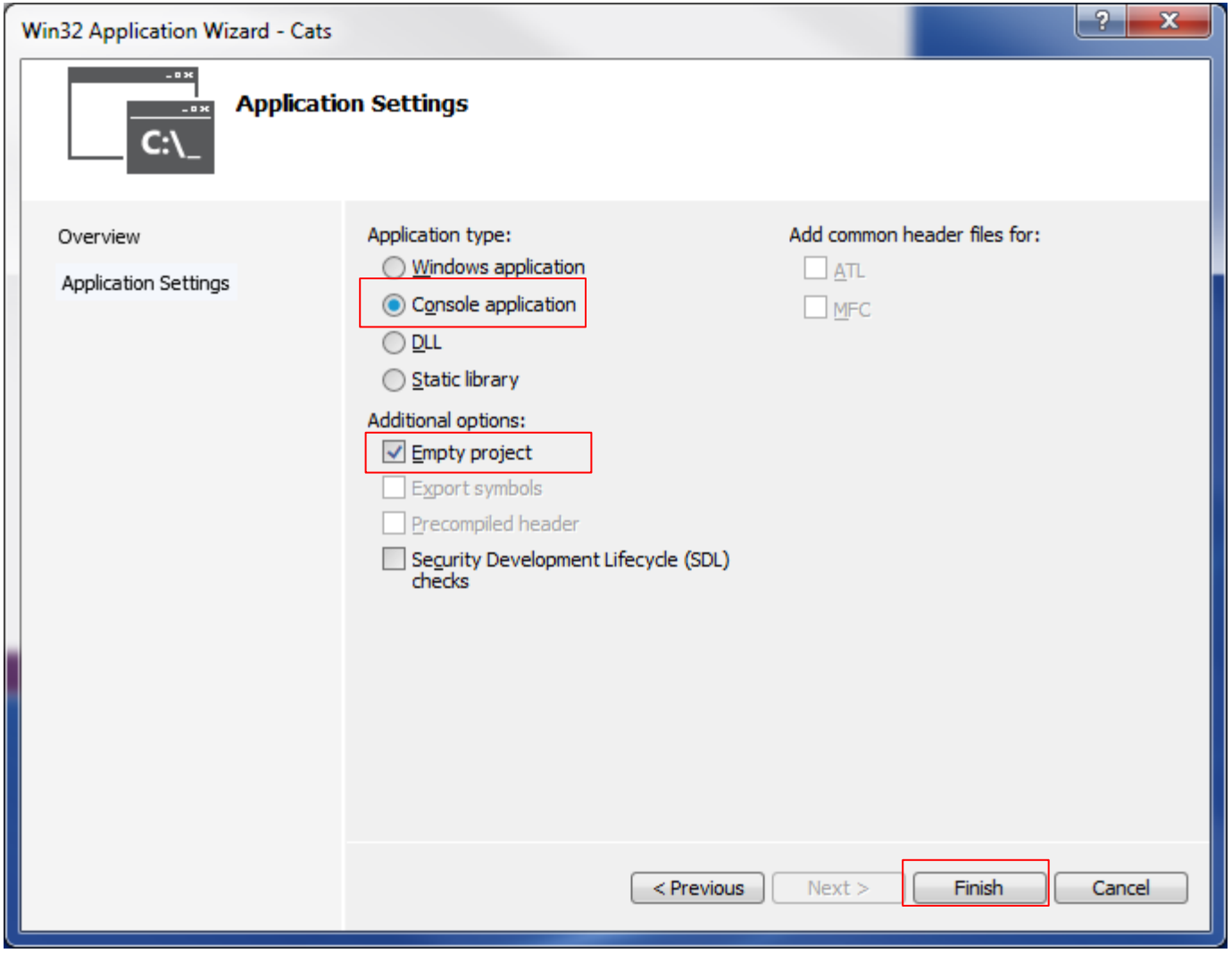
New Project


New Project
Recent
Installed
Templates
Visual C#
Visual Basic
Visual F#
Visual C++
Windows
ATL
CLR
General
MFC
Test
Win32
Cross Platform
Extensibility
Online
.NET Framework 4.5.2
Sort by:
Default
Search Installed Te
Win32 Console Application
Visual C++
Win32 Project
Visual C++
Type: Visual C++
A project for creating a Win32 console application
Click here to go online and find templates.
Name:
Cats
Location:
c:\users\turnerr\documents\visual studio 2015\Projects
Solution name:
Cats
Browse...
Create directory for solution
Add to source control
OK
Cancel

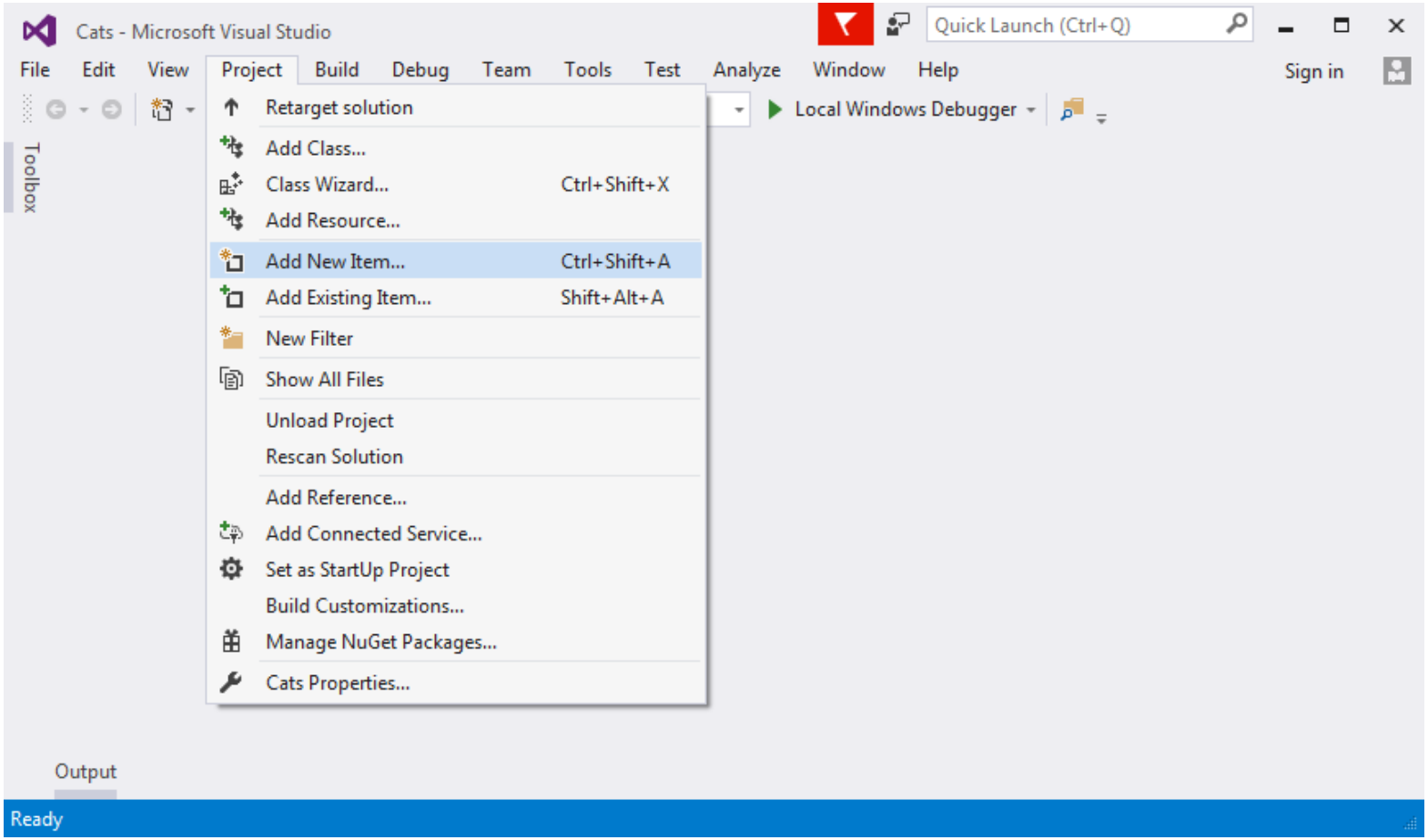
New Project


Win32 Application Wizard - Cats
Welcome to the Win32 Application Wizard
Overview
Application Settings
These are the current project settings:
Console application
Click Finish from any window to accept the current settings.
After you create the project, see the project's readme.txt file for information about the project features and files that are generated.
< Previous
Next >
Finish
Cancel

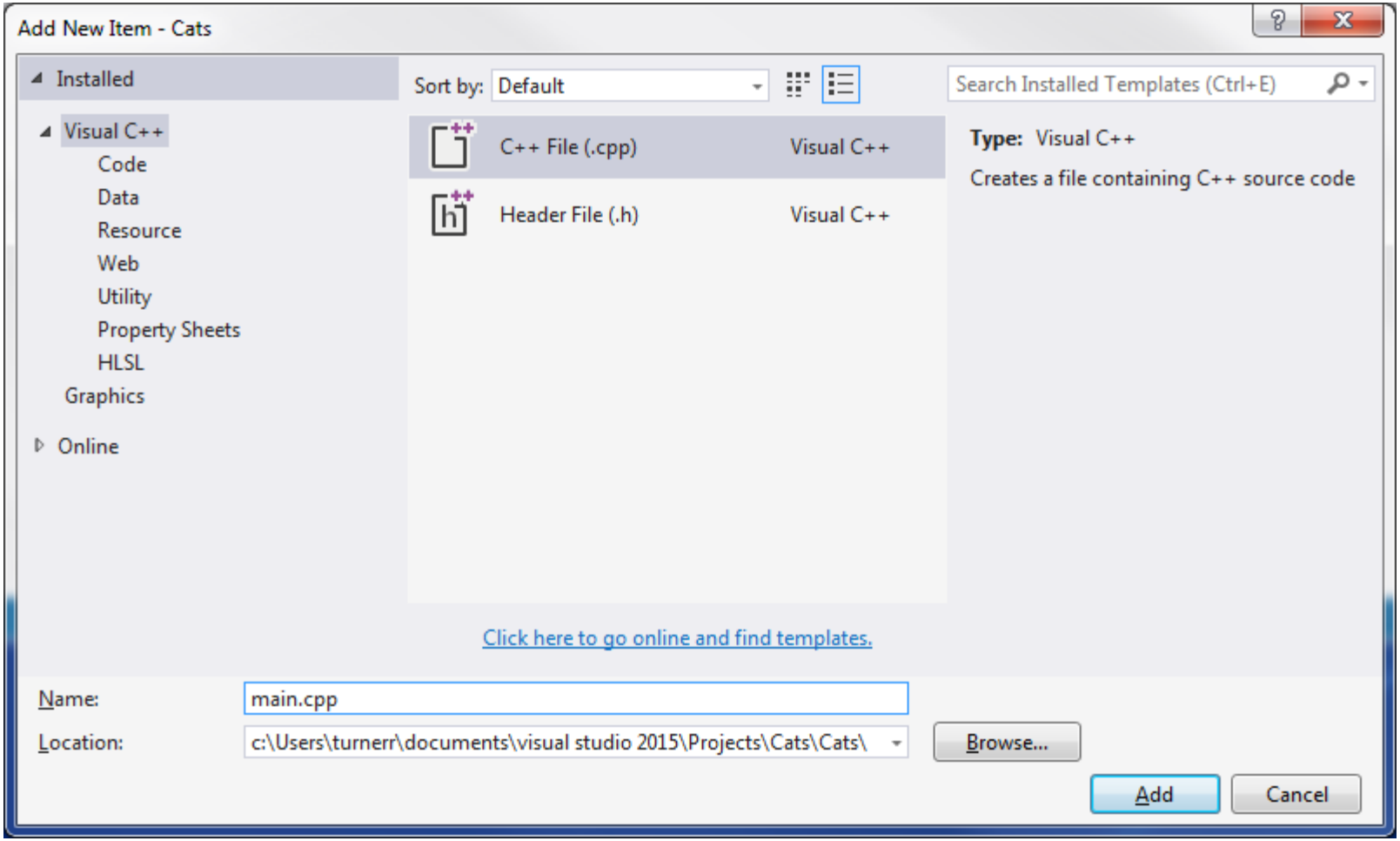
New Project


Win32 Application Wizard - Cats
Application Settings
Overview
Application Settings
Application type:
Windows application
Console application
DLL
Static library
Additional options:
Empty project
Export symbols
Precompiled header
Security Development Lifecycle (SDL) checks
Add common header files for:
ATL
MFC
< Previous
Next >
Finish
Cancel

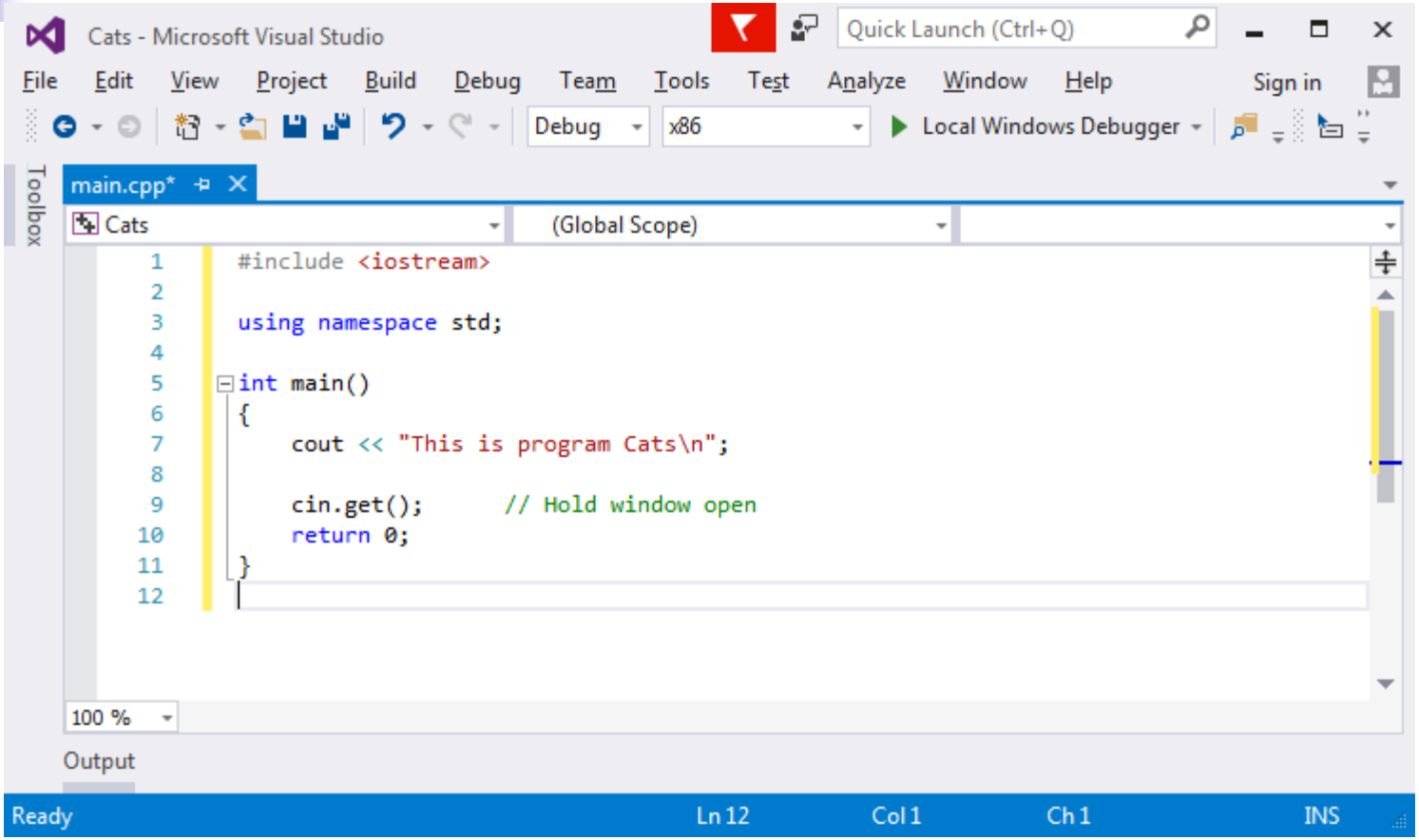
Add main


Cats - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
File
Edit
View
Project
Build
Debug
Team
Tools
Test
Analyze
Window
Help
Sign in
Local Windows Debugger
Toolbox
Retarget solution
Add Class...
Class Wizard... Ctrl+Shift+X
Add Resource...
Add New Item... Ctrl+Shift+A
Add Existing Item... Shift+Alt+A
New Filter
Show All Files
Unload Project
Rescan Solution
Add Reference...
Add Connected Service...
Set as StartUp Project
Build Customizations...
Manage NuGet Packages...
Cats Properties...
Output
Ready

Add main

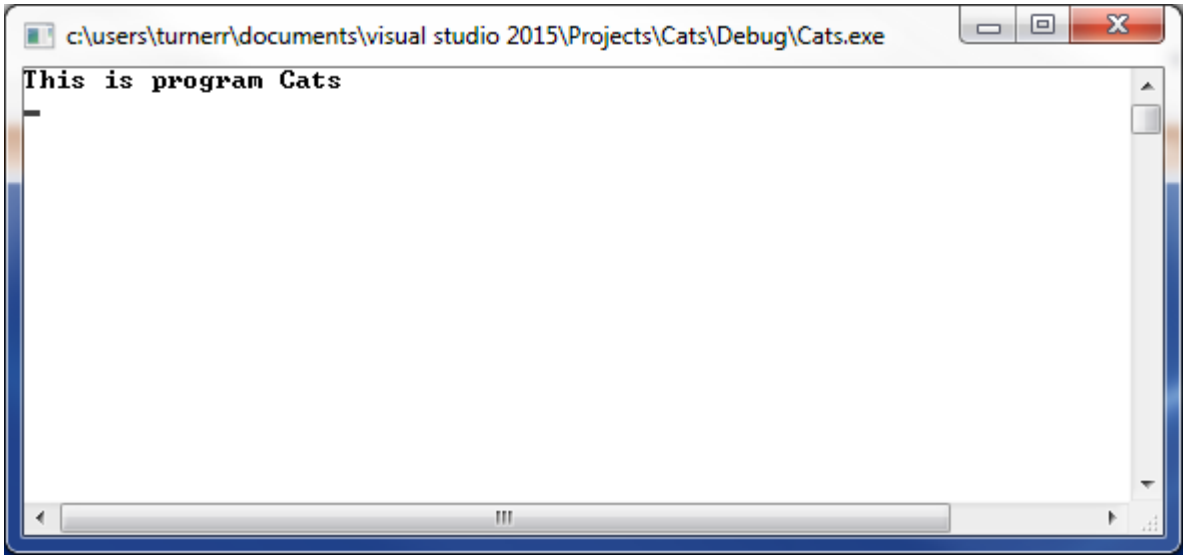

Add New Item - Cats
Installed
Visual C++
Code
Data
Resource
Web
Utility
Property Sheets
HLSL
Graphics
Online
Sort by: Default
Search Installed Templates (Ctrl+E)
C++ File (.cpp) Visual C++
Header File (.h) Visual C++
Type: Visual C++
Creates a file containing C++ source code
Click here to go online and find templates.
Name: main.cpp
Location: c:\Users\turnerr\documents\visual studio 2015\Projects\Cats\Cats\
Browse...
Add
Cancel

Start with a Stub



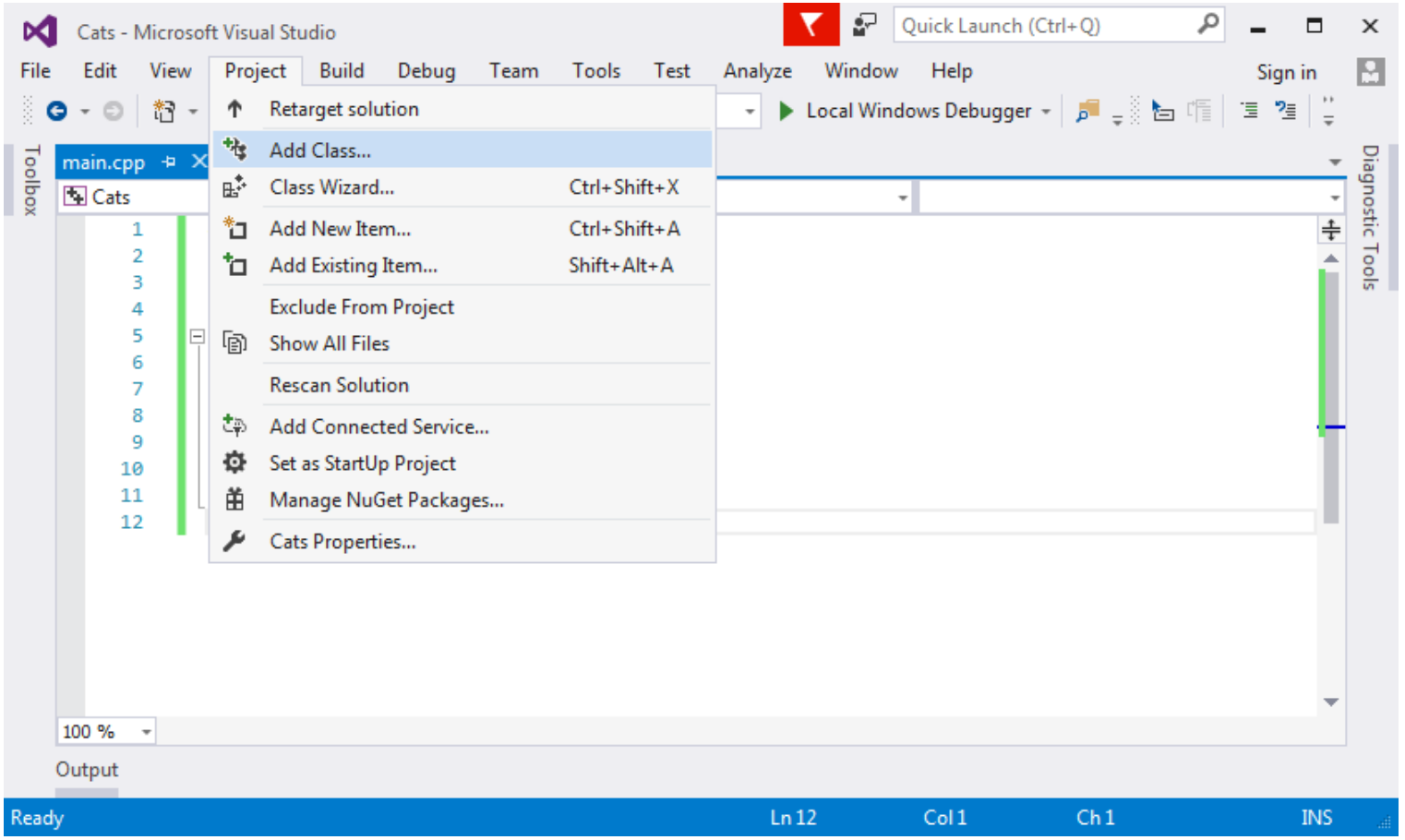
Build and run

Program Running

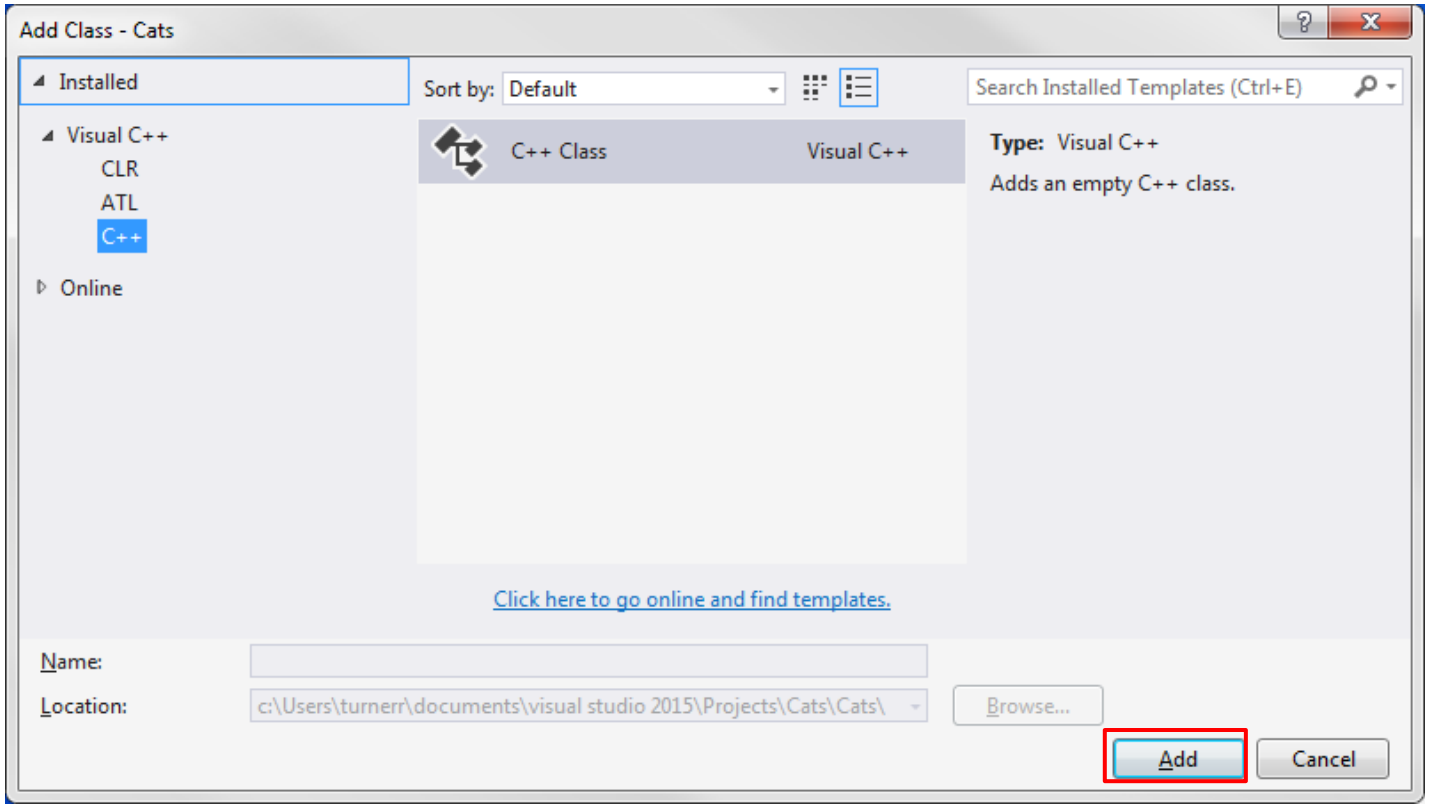


We have a working program!

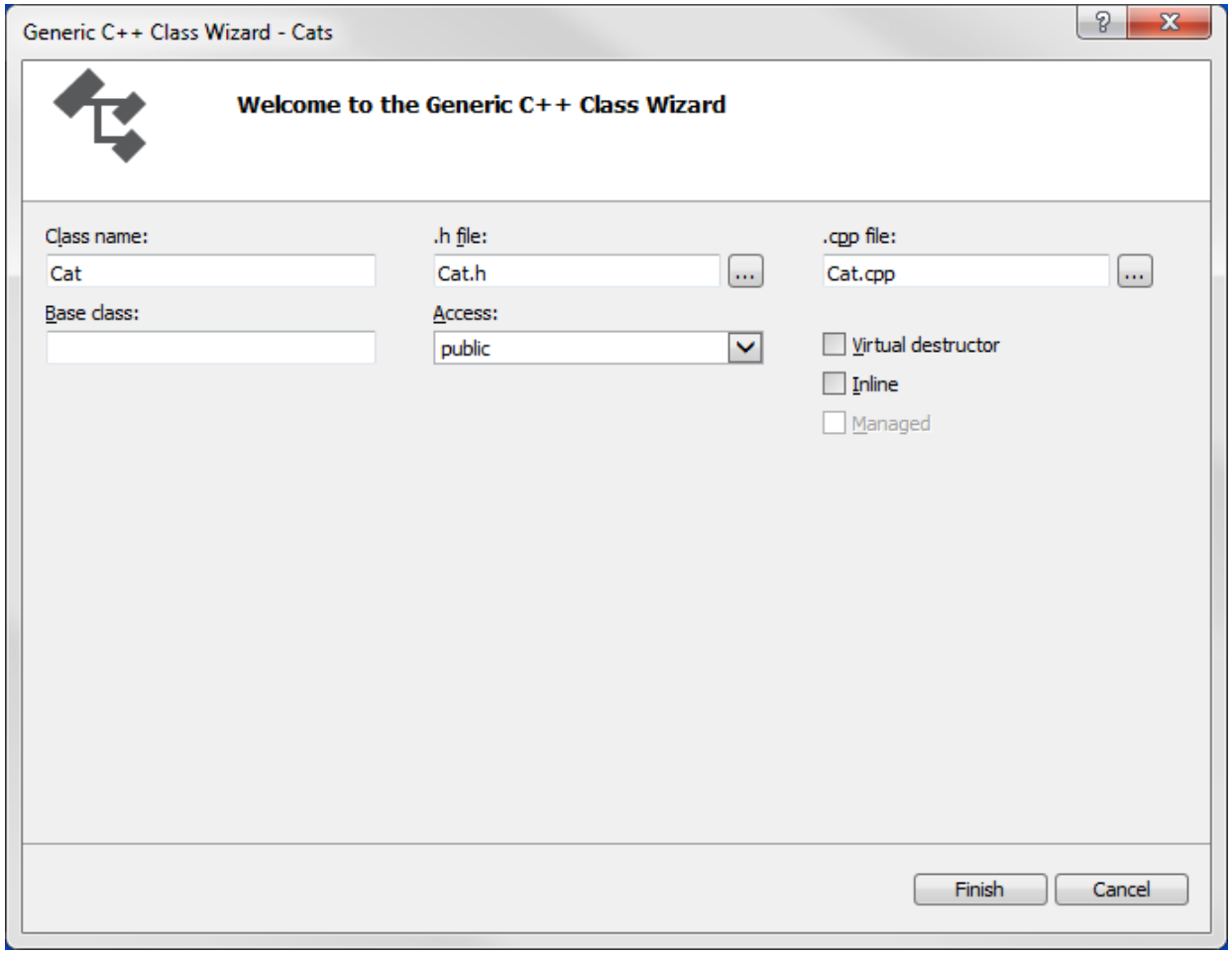
Add Class Cat



Add Class Cat

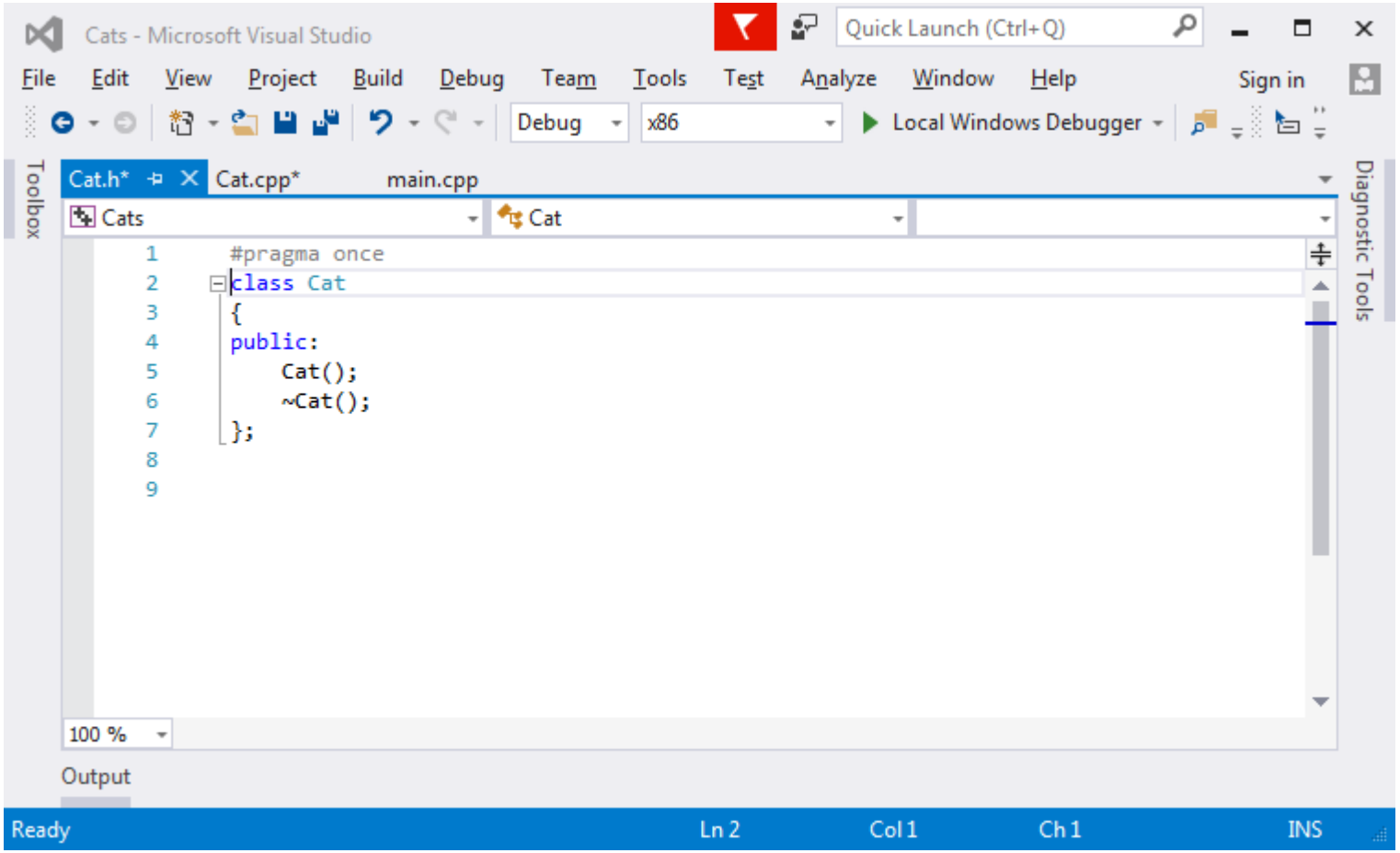


Add Class Cat



Note that class name is a singular noun.

Initial Class Cat



Note #pragma once.

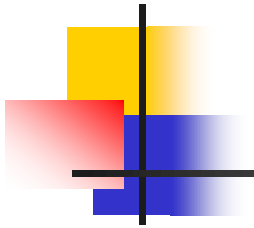
#pragma once

- Same as the traditional guard:

```
#ifndef CAT_H
#define CAT_H
...
// Definition of class Cat
...
#endif
```

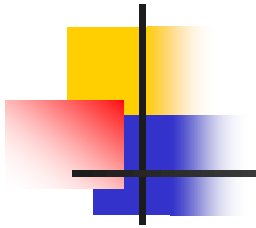
- Not ISO Standard C++, but widely supported by current compilers.

Add Member Variables



- Each cat has:
  - Name
  - Weight
  - Date of birth

- For Name, use a C++ String
- For Weight, use a double.
- What about Date of Birth?

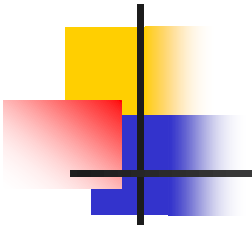


Date of Birth

- The C standard library has a way to represent date/time values and functions to manipulate them.
  - But it's complicated
- Let's define a simple, easy to use, struct to represent dates in a user-friendly form:
  - Day
  - Month
  - Year

Cat.h

```
#pragma once
#include <string>
using namespace std;

struct Date
{
    int Day;
    int Month;
    int Year;
};

class Cat
{

    private:
        string name;
        Date date_of_birth;
        double weight;

    public:
        Cat(string name_, Date dob_, double weight_);

        string Name() const { return name; };
        Date Date_of_Birth() const { return date_of_birth; };
        double Weight() const { return weight; };

        ~Cat(void);
};
```



The C++ string class

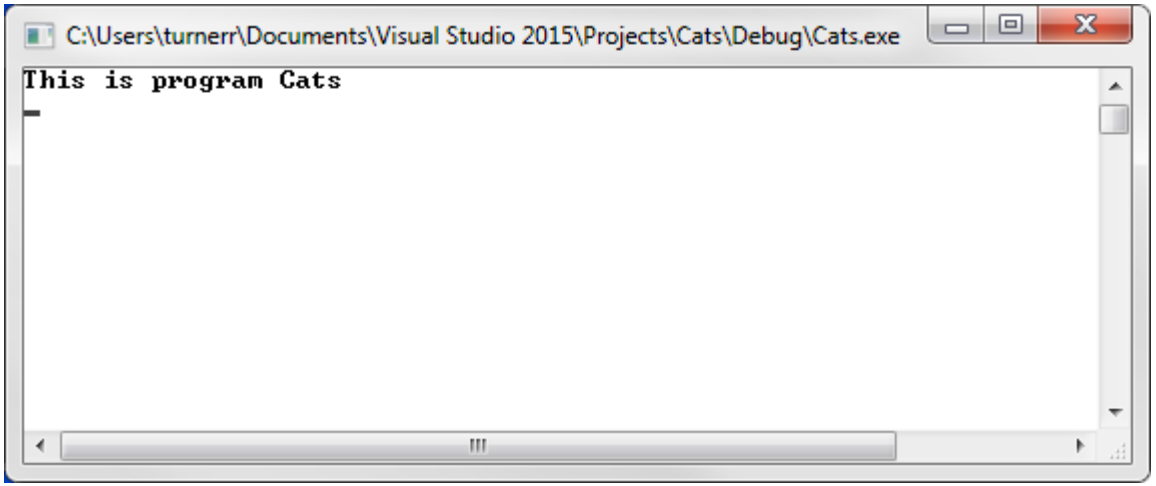
- C++ supports C style strings
  - Arrays of char
  - Used in Chapter 10 of the textbook
- Also has a string class
  - More user friendly
  - Less subject to problems and errors
  - See page 131 and 353 of the textbook

Cat.cpp

```
#include "Cat.h"

Cat::Cat (string name_, Date dob_, double weight_)
{
   name = name_;
   date_of_birth = dob_;
   weight = weight_;
}

Cat::~Cat(void)
{
}
```

Build and run the project

Program Running



Accessor Functions

- We can create Cat objects now, but we can't do anything with them.

- To make the attributes of a Cat visible outside the class, create public we need to provide *accessor functions*.
  - Functions that just return the value of a member variable.

Accessor Functions

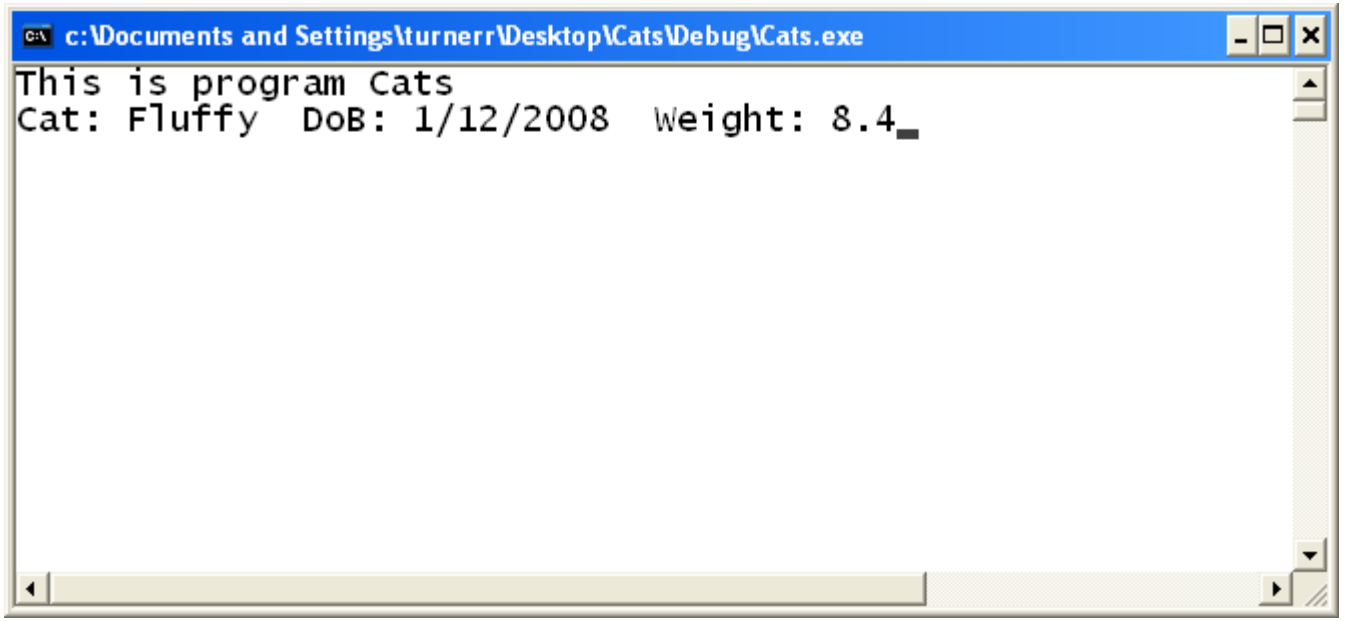
```
public:
    Cat(string name_, Date dob_, double weight_);

    string Name() const { return name; };
    Date Date_of_Birth() const { return date_of_birth;};
    double Weight() const {return weight;};

    ~Cat(void);
};
```

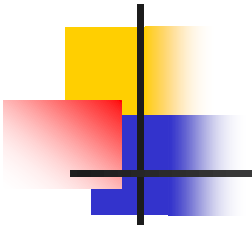
- What are all those *const's* ?
- Note that we are putting the implementation of the methods in the class definition.
  - Methods will be compiled *in line*.

main.cpp

```
#include <iostream>
#include "Cat.h"
using namespace std;

int main()
{
    cout << "This is program Cats\n";
    Date dob = {12,1,2008};
    Cat Fluffy("Fluffy", dob, 8.4);

    cout << "Cat: " << Fluffy.Name() << "  ";
    cout << "DoB: "
         << Fluffy.Date_of_Birth().Month << "/"
         << Fluffy.Date_of_Birth().Day << "/"
         << Fluffy.Date_of_Birth().Year << "  ";
    cout << "Weight: " << Fluffy.Weight();


    cin.get();  // Hold the window open.
    return 0;
}
```

Program Running





Display Method

- We often want a method in a class that outputs the member variables to the screen.

- Add a new public method to Cat.h:

```
void Display() const;
```

In Cat.cpp

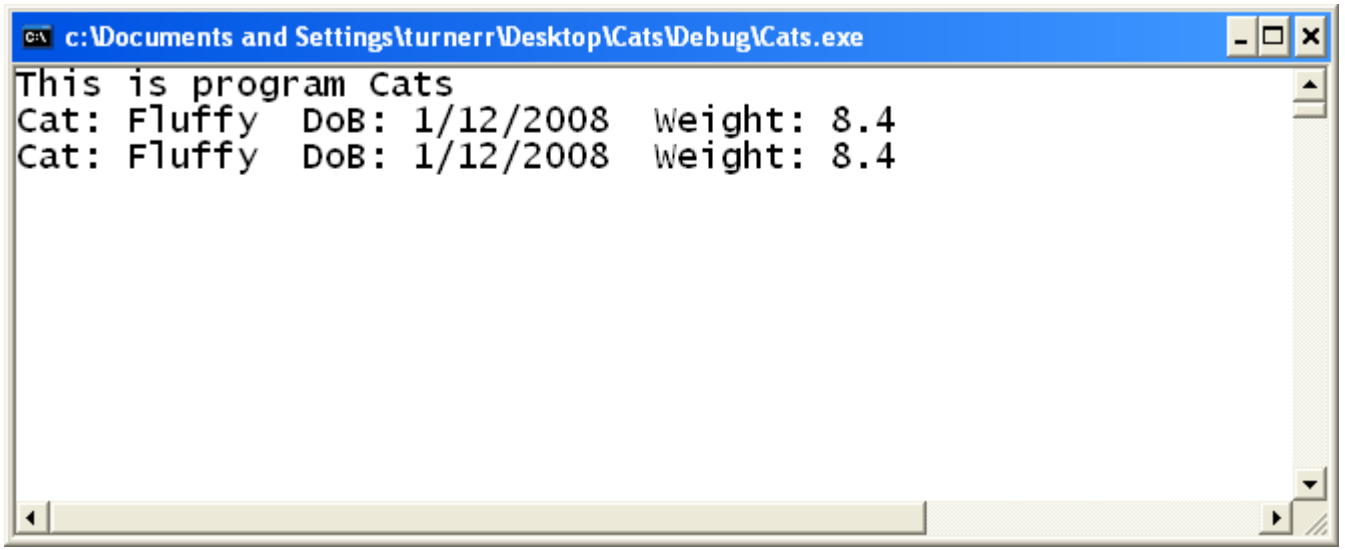
```
#include <iostream>
#include "Cat.h"

...

void Cat::Display() const
{
    cout << "Cat: " << name << "   ";
    cout << "DoB: "
        << date_of_birth.Month << "/"
        << date_of_birth.Day << "/"
        << date_of_birth.Year << "   ";
    cout << "Weight: " << weight;
}
```

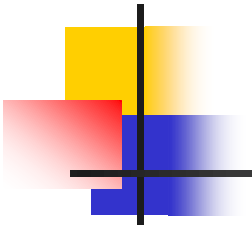
main.cpp

```
#include <iostream>
#include "Cat.h"

using namespace std;

int main()
{
    cout << "This is program Cats\n";
    Date dob = {12,1,2008};
    Cat Fluffy("Fluffy", dob, 8.4);
    …
    cout << endl;
    Fluffy.Display();
    cout << endl;

    cin.get();    // Hold window open.
    return 0;
}
```

Build and run.

Program Running





Dynamic Allocation

- We can dynamically create objects of any class using **`new`**.
- Similar to malloc in C.
- Allocates space on the heap.
- Invokes constructor to initialize the object.
- Returns a *pointer* to the object.

Dynamic Allocation

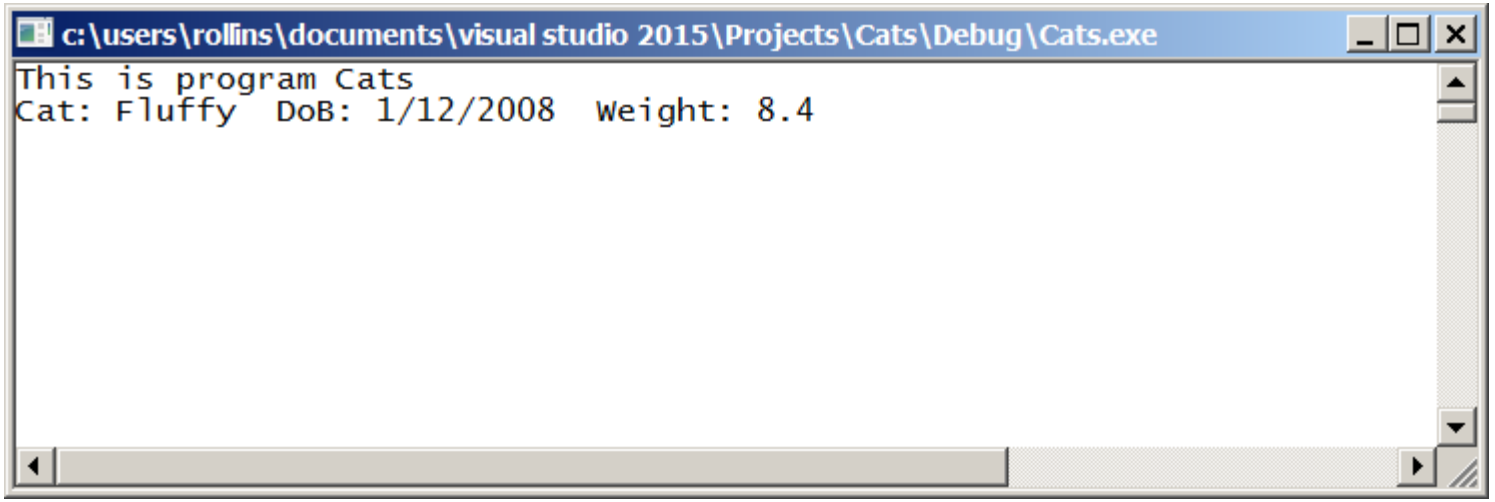
```
#include <iostream>
#include "Cat.h"

using namespace std;

int main()
{
    cout << "This is program Cats\n";
    Date dob = {12,1,2008};
    Cat* Fluffy = new Cat("Fluffy", dob, 8.4);

    Fluffy->Display();
    cout << endl;

    cin.get();     // Hold window open.
    return 0;
}
```

Note arrow.

Fluffy is now a *pointer* .

Build and run

Program Cats



Object Lifetime

- Objects allocated by declaration are deallocated when the function exits.
  - Like all local variables.

- Objects allocated with *new* continue to exist until explicitly deleted

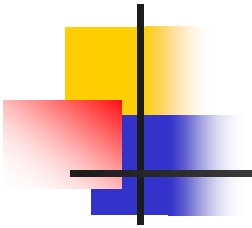
  (or until the program ends.)

  - `delete c1;`

  - c1 is a *pointer* to the object that is to be deleted.

Explicitly Deleting an Object

```
#include <iostream>
#include "Cat.h"
using namespace std;

int main()
{
    cout << "This is program Cats\n";
    Date dob = {12,1,2008};
    Cat* Fluffy = new Cat("Fluffy", dob, 8.4);

    Fluffy->Display();
    cout << endl;

    delete Fluffy;
    Fluffy = 0;

    cin.get();    // Hold window open.
    return 0;
}
```



Getting User Input

- What if we want the *user* to specify the attributes of a Cat.
  - Could overload the constructor and provide a version that asks for input.
  - Better to provide a separate function outside the class definition.
    - *Separate UI from class logic.*
- Let's write a function that asks the user for information about a Cat, creates a Cat object, and returns a pointer to the object.

main.cpp

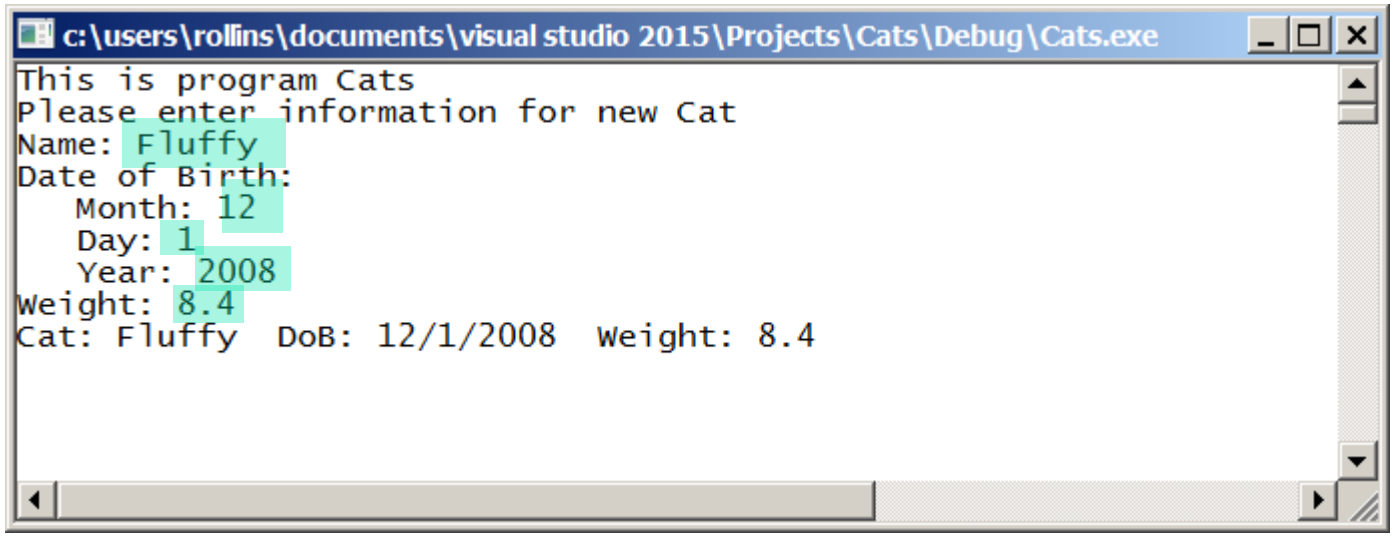
```
Cat* Create_Cat()
{
    string name;
    Date date_of_birth;
    double weight;
    cout << "Please enter information for new Cat\n";
    cout << "Name: ";
    cin >> name;
    cout << "Date of Birth:\n";
    cout << "   Month: ";
    cin >> date_of_birth.Month;
    cout << "   Day: ";
    cin >> date_of_birth.Day;
    cout << "   Year: ";
    cin >> date_of_birth.Year;

    cout << "Weight: ";
    cin >> weight;
    Cat* cat = new Cat(name, date_of_birth, weight);
    return cat;
}
```

Getting User Input

In file main.cpp

```
int main()
{
    cout << "This is program Cats\n";

    Cat* Fluffy = Create_Cat();

    Fluffy->Display();
    cout << endl;

    delete Fluffy ;
    Fluffy = 0;

    cin.get();    // Hold window open.
    cin.get();
    return 0;
}
```

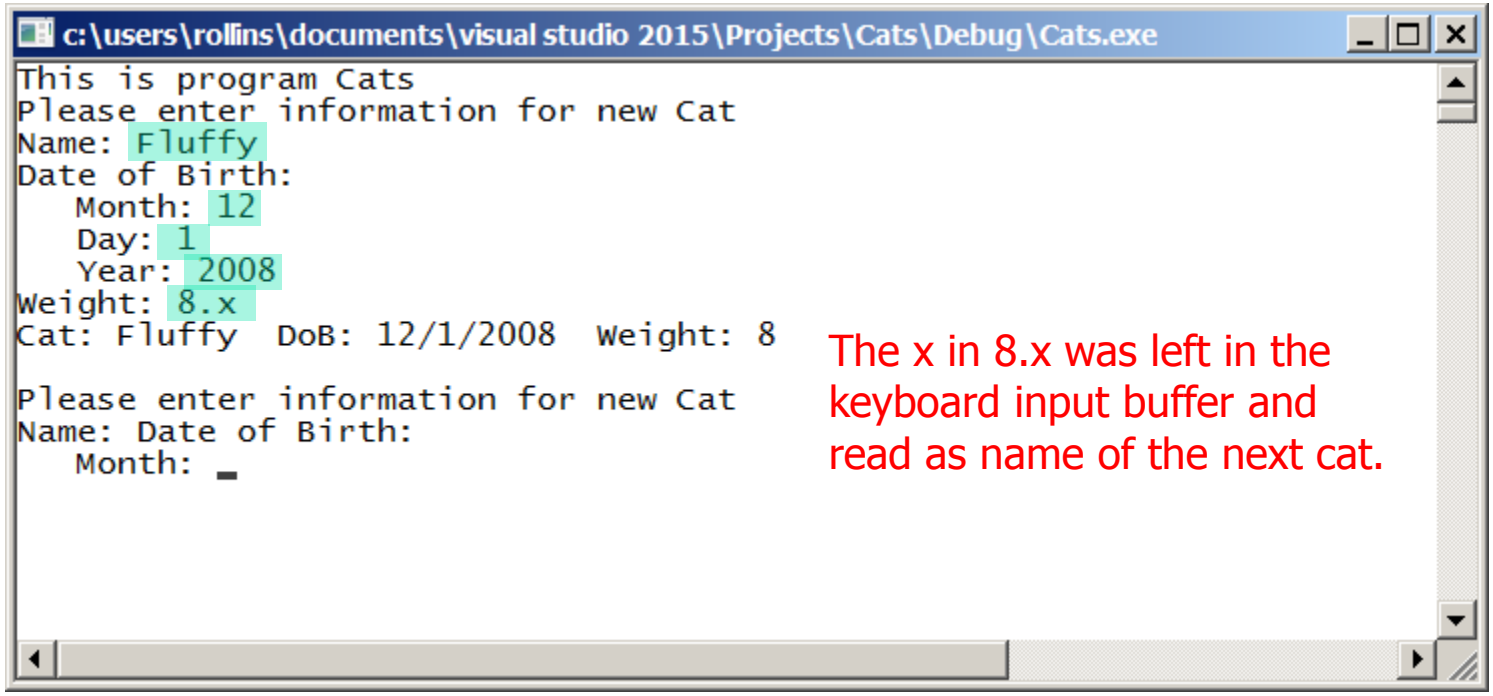
Running Program Cats



Multiple Cats

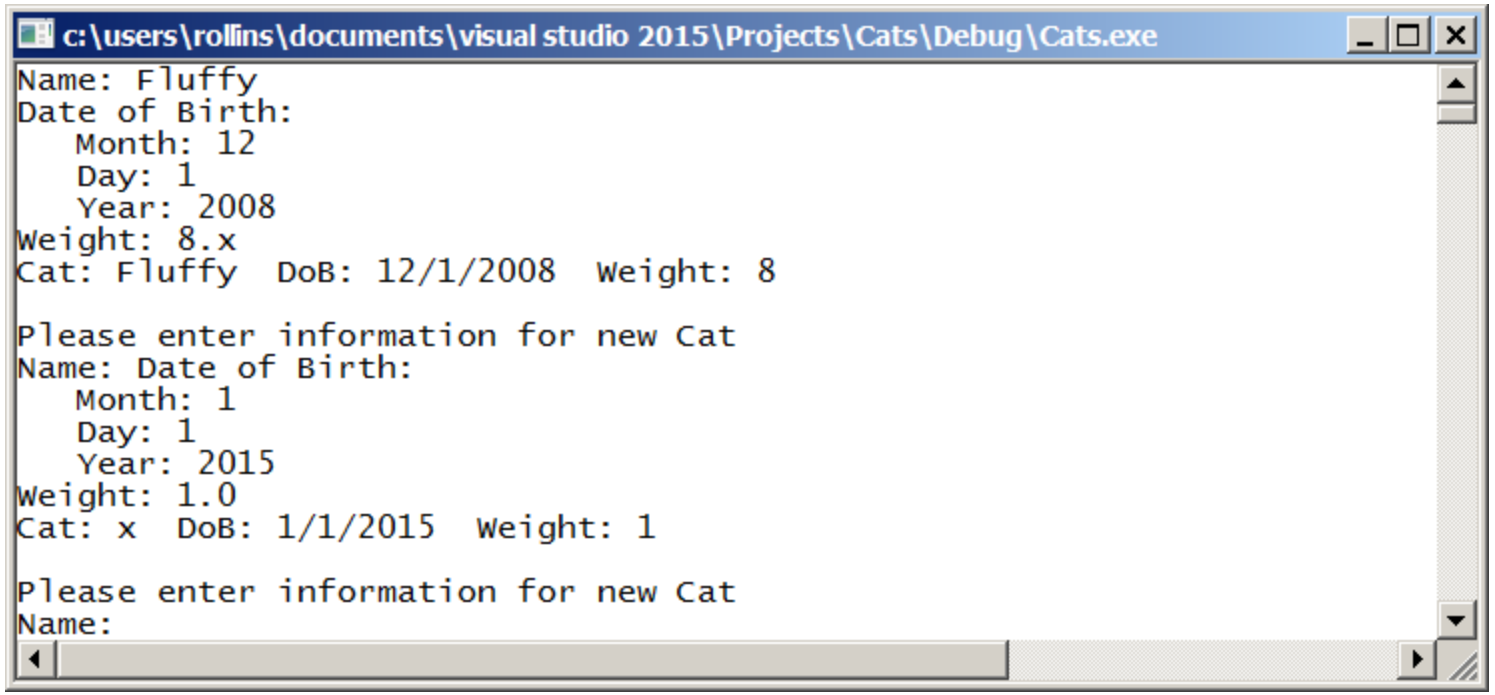
```
int main()
{
    cout << "This is program Cats\n";

    while (true)
    {
        Cat* Fluffy = Create_Cat();

        Fluffy->Display();
        cout << endl;

        delete Fluffy;
        Fluffy = 0;
    }
    ...
```

A Problem with Numeric Input



The x in 8.x was left in the keyboard input buffer and read as name of the next cat.

A Problem with Numeric Input



Solution

- After numeric input, clear the keyboard input buffer before doing the next real input.

```
while (cin.get() != '\n') ;
```

Clear_Keyboard_Input_Buffer()

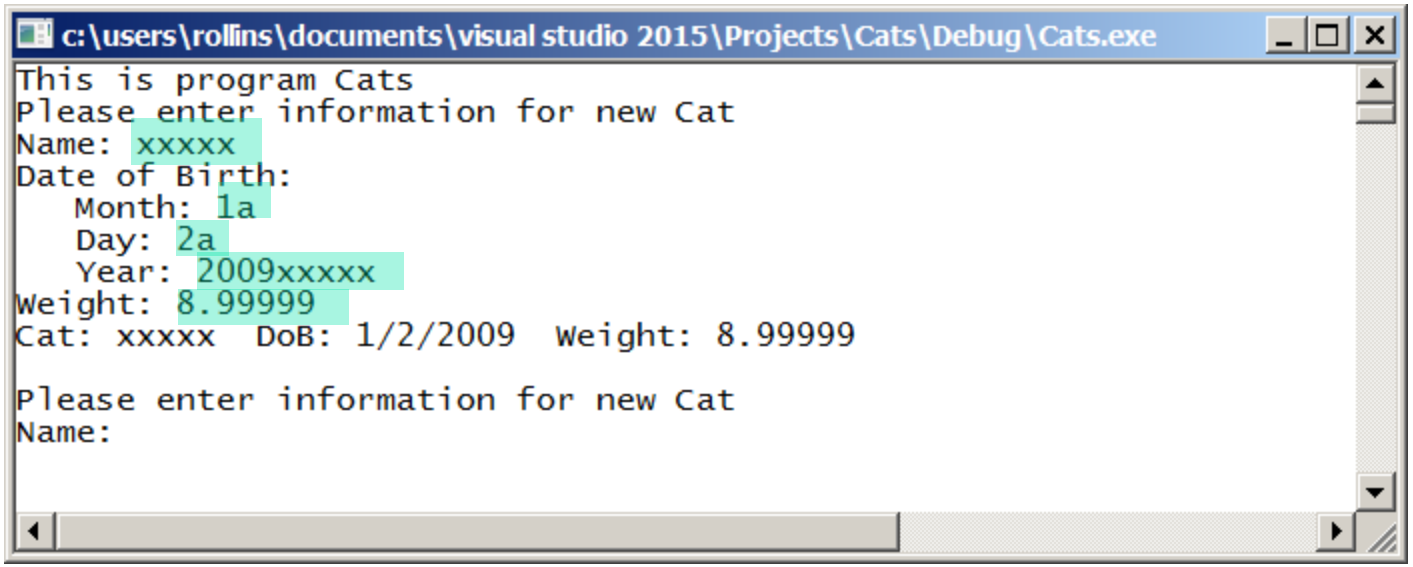
```
void Clear_Keyboard_Input_Buffer()
{
    while (cin.get() != '\n') ;
}
```

- Call this function after getting a numeric value to clear any characters following the numeric text.

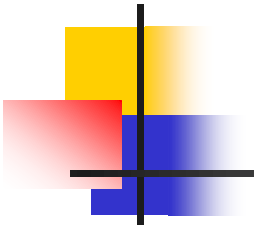
Using Clear_Keyboard_Input_Buffer

```
Cat* Create_Cat()
{
     ...
     cout << "Date of Birth:\n";
     cout << "   Month: ";
     cin >> date_of_birth.Month;
     Clear_Keyboard_Input_Buffer();
     cout << "   Day: ";
     cin >> date_of_birth.Day;
     Clear_Keyboard_Input_Buffer();
     cout << "   Year: ";
     cin >> date_of_birth.Year;
     Clear_Keyboard_Input_Buffer();

     cout << "Weight: ";
     cin >> weight;
     Clear_Keyboard_Input_Buffer();
     Cat* cat = new Cat(name, date_of_birth, weight);
     return cat;
}
```

Program in Action



Garbage at end of numeric inputs was silently ignored.



Assignment

- Do today's example for yourself
  - if you have not done it in class.
- Read Chapter 10.