Programmer Defined Method

CS 133N/ CS 161N

Mari Good

Objectives

- Introduce Modularization
- Introduce the Structure Chart
- Introduce you to C# syntax for defining and calling methods
- Practice with several examples

Modularization

- Is a software design technique that emphasizes separating the functionality of a program into independent modules.
  - This is necessary because professional programs consist of millions of lines of code.
  - It allows us to
    - manage complexity
    - work as a team
    - potentially reuse pieces of code in multiple programs

Modularization

- Two software engineering principles are important in this process
  - Cohesion – refers to the degree to which the elements inside a module belong together. Modules should be highly cohesive. That means that a module should do just 1 thing.
  - Coupling – refers to the degree of interdependence between the modules. Modules should be loosely coupled. That means that modules should share data by passing parameters and returning values rather than relying on global data.

Let's start with an example

- Remember the gross pay problem from the selection topic? We asked the user to enter pay rate and hours worked and then calculated gross pay including overtime.
  - At that point we created an IPO chart, the processing steps of which looked something like what's on the next slide.
    - Originally we assumed that the user was well behaved. When the user isn't well behaved, even this relatively simple problem can become complicated.

IPO Chart

Input
- payRate
- hoursWorked

Processing
- get payRate
- get hoursWorked
- calculate the grossPay
- display grossPay

Output
- grossPay

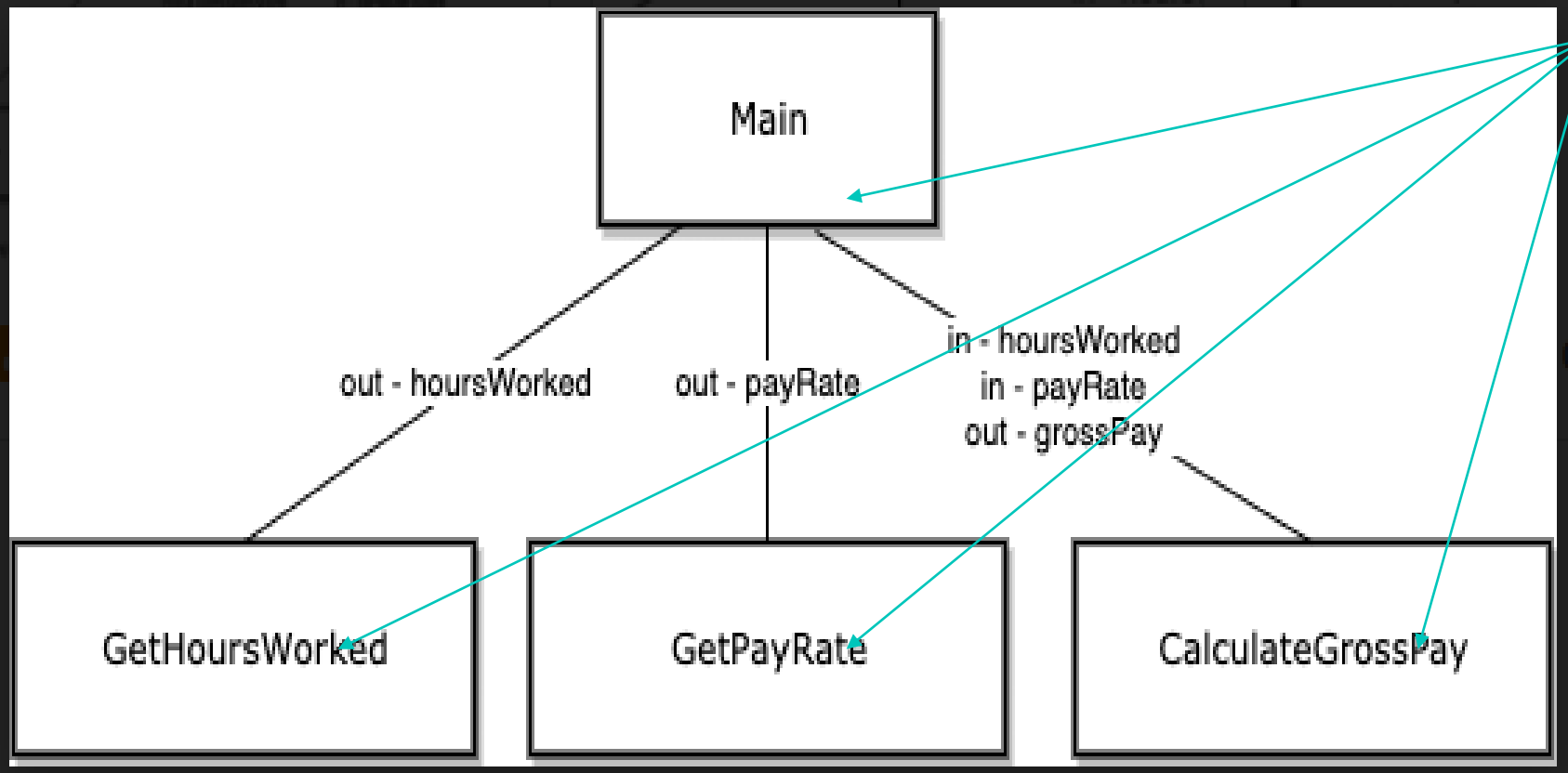
Need a loop here to validate input

Need another loop here to validate input

Need an if statement here to determine overtime

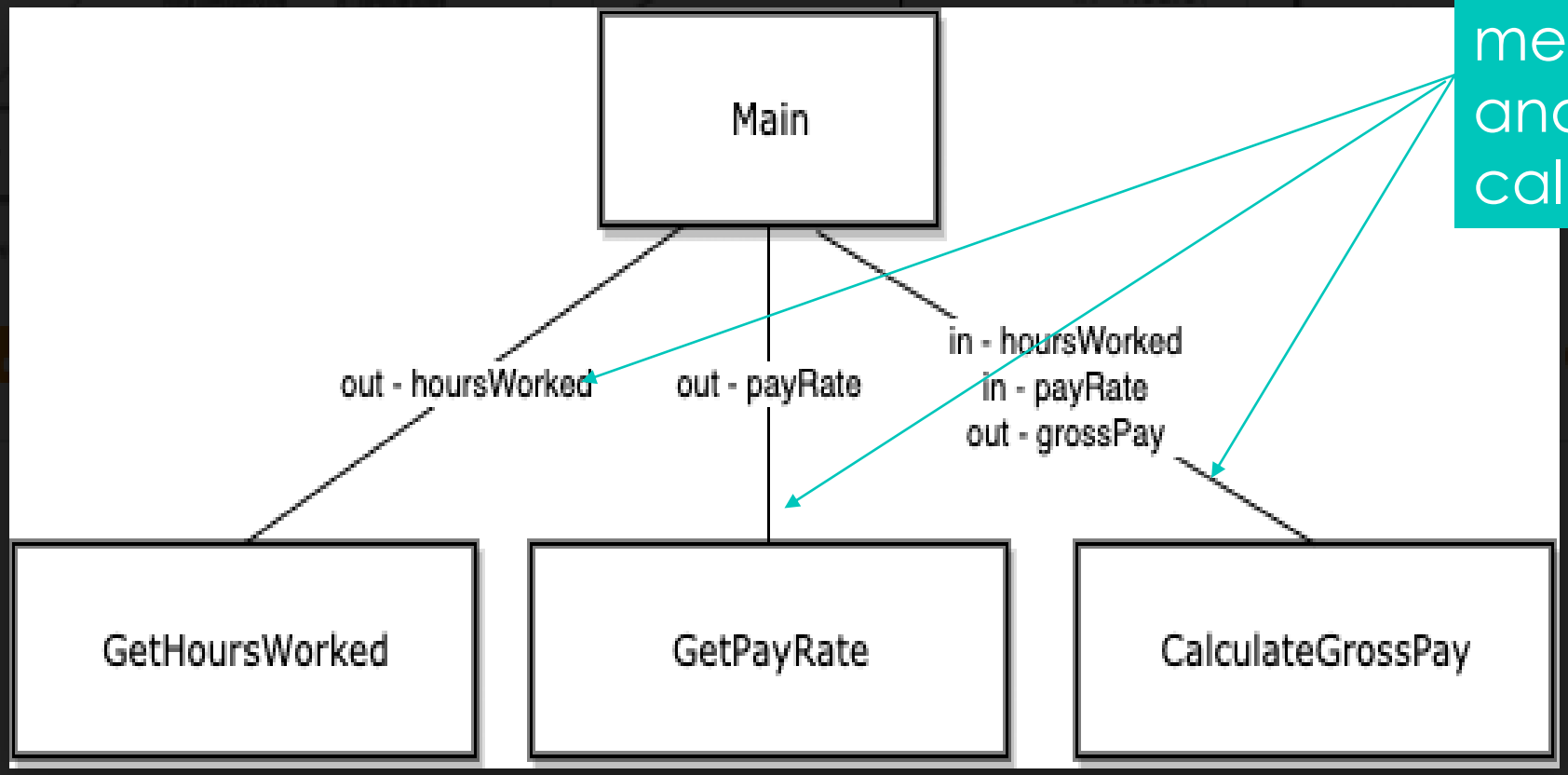
If we wanted to process multiple employees we'd need another loop around the whole thing

Structure Chart



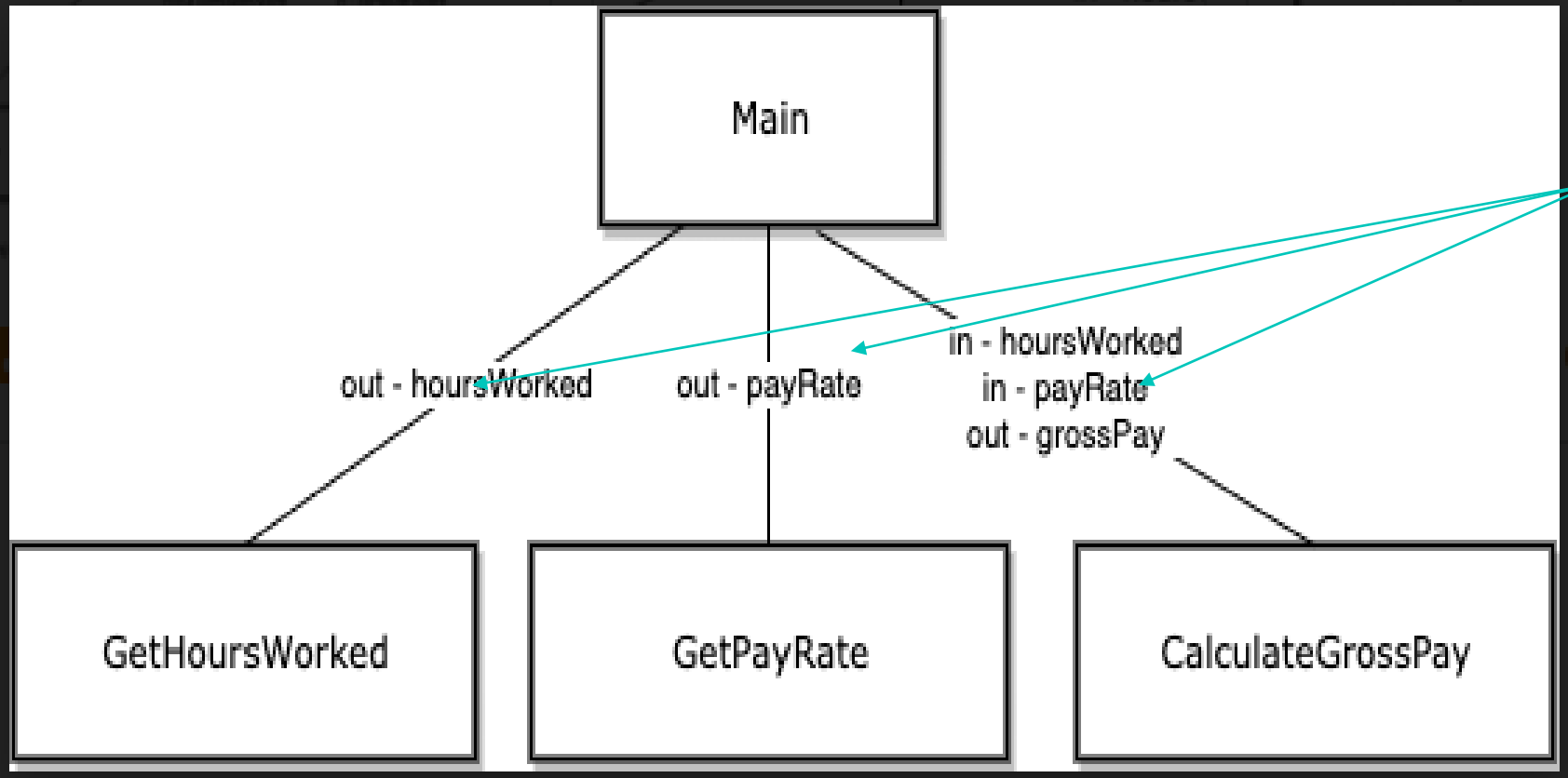
Each box indicates a module (method). There are 4.

Structure Chart



Each line indicates that a method executes or calls another method. Main calls the other 3 methods.

Structure Chart



The words on the call lines indicate data passed into the method (parameter) and data passed out of the method (return value).

- GetHoursWorked and GetPayRate have no parameters but return a value.
- CalculateGrossPay has 2 parameter AND returns a value.

Pseudocode and Modularization

- You'll still write algorithms but not for the program as a whole. Now we'll write algorithms for any module (method) that is logically complex.
  - I won't do that now because we've already written algorithms for
    - Validating input from the user
    - Calculating grossPay

Syntax For Defining a Method

Datatype of the return value

Heading

```
static double GetHoursWorked()
{
    bool isDouble = false;
    double hoursWorked;
    do
    {
        Console.Write("Please enter the number of hours you worked: ");
        string input = Console.ReadLine();
        isDouble = double.TryParse(input, out hoursWorked);
    } while (!(isDouble && hoursWorked > 0.0) );
    return hoursWorked;
}
```

No Parameters

Local Variables

Body

Return value.

TryParse

double.Parse "blows up" when the input is not a double. double.TryParse returns a bool (true or false) instead.

It takes 2 parameters. The first is the string you want to try to parse

```
isDouble = double.TryParse(input, out hoursWorked);
```

A method can return 1 value. Because the 2nd parameter is "indirectly returned" from the call to TryParse, it is an output parameter.

The second is the variable in which the double value will be placed when the string can be parsed.

Syntax For Defining a Method

Datatype of the return value

Heading

```
static decimal GetPayRate()
{
    bool isDecimal = false;
    decimal payRate;
    do
    {
        Console.Write("Please enter your pay rate: ");
        string input = Console.ReadLine();
        isDecimal = decimal.TryParse(input, out payRate);
    } while (!(isDecimal && payRate > 0.0M) );
    return payRate;
}
```

No Parameters

Local Variables

Body

Return value.

Syntax For Defining a Method

Datatype of the return value

Heading

```
static decimal CalculateGrossPay(double hoursWorked, decimal payRate)
{
    decimal grossPay;

    if (hoursWorked <= 40)
        grossPay = (decimal)hoursWorked * payRate;
    else
        grossPay = 40 * payRate + ((decimal)hoursWorked - 40) * payRate * 1.5M;

    return grossPay;
}
```

Parameter List

Local Variables

Body

Return value.

Parameters used like local variables

Syntax For Calling Methods

Heading

```
public static void Main()
{
        Console.WriteLine("Enter your hours worked and your pay rate and I will calculate you
gross pay for the week.");
```

Local variables store the value returned from each call

Body

```
        double hoursWorked = GetHoursWorked();
        decimal payRate = GetPayRate();
        decimal grossPay = CalculateGrossPay(hoursWorked, payRate);
```

Call or execute a method

Pass parameters into method

```
        Console.WriteLine("Your gross pay is: " + grossPay.ToString("c"));
}
```

The Debugger In Visual Studio

- Before we go on to another problem, I want to demonstrate this program first in dotnetfiddle and then in Visual Studio using the debugger.
- Remind me to tell you about
  - Breakpoints
  - Step over and step into
  - Locals window

Let's Try One Together

- Design and implement a program that asks the user to enter 3 quiz scores (out of 50), calculates and displays the average score and the average percentage as well as a letter grade. >= 90 is an A, between 80 and 90 is a B, between 70 and 80 is a C, below 70 is an F. The program should display an error message and allow the user to reenter any scores that are not whole numbers between 0 and 50. This is the first problem from your lab.

IPO Chart

Input
- score1, score2, score3

Processing
- get score1, get score2, get score3
- calculate averageScore
- calculate averagePercentage
- determine letterGrade
- display results

Output
- averageScore
- averagePercentage
- letterGrade

Any of the processing steps that are "complicated" should be a method.
- GetScore
- DetermineLetterGrade
- Maybe even CalculateAverage, CalculatePercentage, and DisplayResults

Let's create a structure chart and then write code together in VS.

You Try One

- Design and implement a program that can be used to provide users about a car loan. The program should allow a user to enter the amount of the loan and the annual interest rate. The program should validate data entered by the user in a reasonable way and should force the user to reenter until the data is correct. The program should display the monthly payment for a 4 year loan, a 5 year loan and a 6 year loan in an attractively formatted table. This is the first problem from your lab.

You Try One

- Start with the IPO chart. In the processing steps anything that's complicated becomes a method.
- Create a structure chart
  - Name the method
  - List it's parameters
  - Identify the return data type
  - The CalculatePayment method should be a method. The formula is on the next slide. What pieces of input does it need from main? Those will be the parameters.

You Try One

- Here's the formula for calculating the payment
- $$\text{payment} = \frac{(\text{rate per period} * \text{loan amount})}{(1 - (1 + \text{rate per period})^{-\text{periods}})}$$

Here's another example

- Remember the perfect number problem from the repetition topic? We asked the user to enter a number and determined if it was perfect.
- The IPO chart is on the next slide

IPO Chart

Input
- number

Processing
- get number
- determine if number is perfect
- display results

Output
- Perfect or not

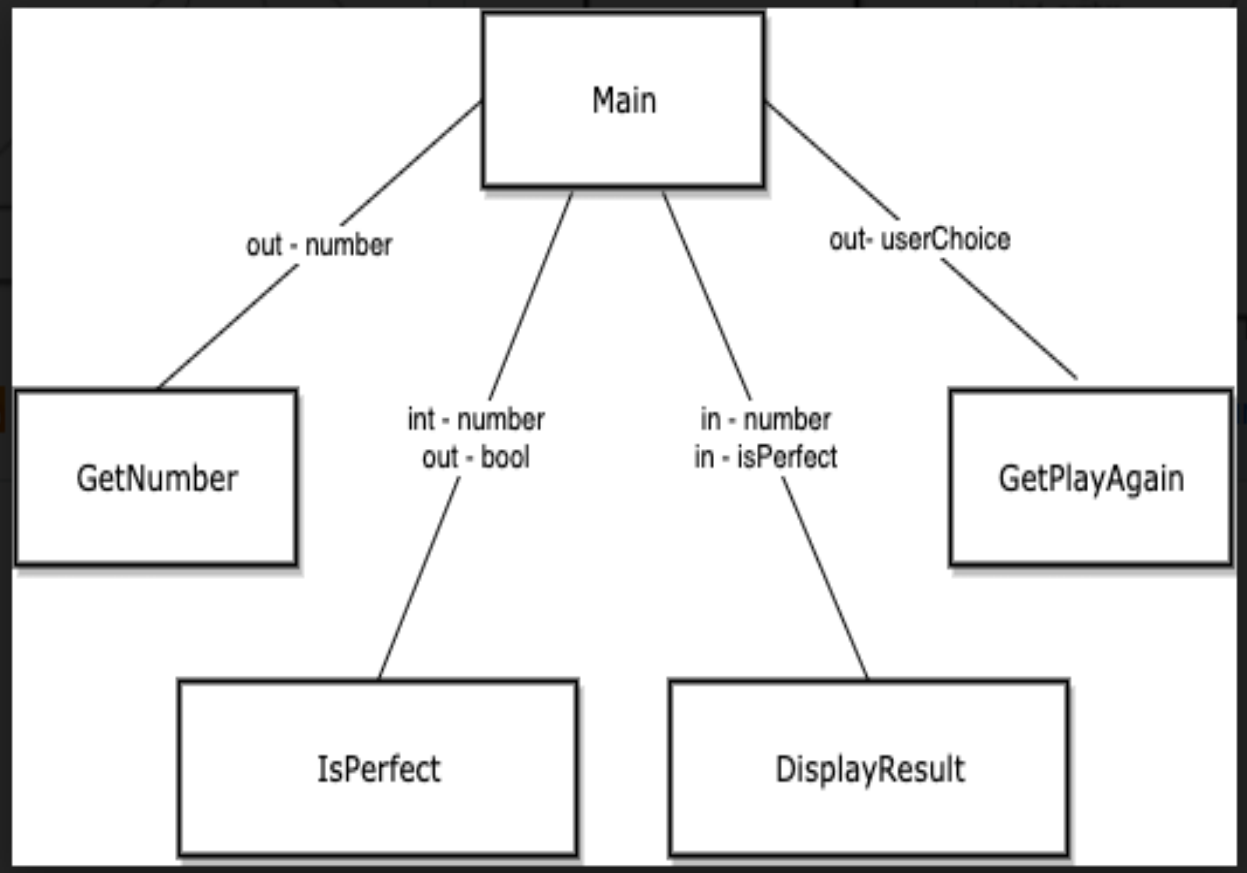
Hey! I notice that we've been writing methods that get an int from the user. I wonder if we can generalize?

Should this method return true or false? Or display output? Can a method return true or false?

This method doesn't return a value. Is that OK?

What's with all of these parameters and return values? Why not use global variables?

Structure Chart



There are 2 CS principles that I keep in mind when designing methods

- Cohesion – each method does one thing. IsPerfect and DisplayResults are 2 methods
- Coupling – methods "share" data by passing parameters and returning values. Global variables should be avoided.

New Syntax

```
static bool IsPerfect(int number)
{
    int sum = 0;
    for (int divisor = 1; divisor < number; divisor++)
        if (number % divisor == 0)
            sum += divisor;
    if (sum == number)
        return true;
    else
        return false;
}
```

Returns either true or false. Return datatype is bool.

Return value.

New Syntax

```
static void DisplayResult(int number, bool isPerfect)
{
    if (isPerfect)
        Console.WriteLine(String.Format("{0} IS perfect!", number));
    else
        Console.WriteLine(String.Format("{0} IS NOT perfect!", number));
}
```

Doesn't return a value.

No return statement

Test Method

```
static void TestIsPerfect()
{
    Console.WriteLine("Testing IsPerfect.  6 should be perfect.  8 shoud not be perfect.");
    if (IsPerfect(6))
        Console.WriteLine(String.Format("{0} IS perfect!", 6));
    else
        Console.WriteLine(String.Format("{0} IS NOT perfect!", 6));
    if (IsPerfect(8))
        Console.WriteLine(String.Format("{0} IS perfect!", 8));
    else
        Console.WriteLine(String.Format("{0} IS NOT perfect!", 8));
}
```

Calling Test Method in Main

```
public static void Main()
{
    //Testing
    //TestGetInt();
    //TestIsPositiveInt();
    //TestGenericIsInt();
    //TestIsPerfect();
    //TestDisplayResult();
}
```

You Try The Next Two

- You've done 1 and 2 from Lab 5. I'd like you to try 3 and 4 on your own or in small groups.
- Create an IPO chart. It will help you identify methods.
- Create a structure chart
  - Name the method
  - List it's parameters
  - Identify the return data type
- Write ONE METHOD AT A TIME and TEST IT.
- Write Main and test your application

Our last example

- Remember the Rock Paper Scissors problem from the selection and repetition topics?
- The IPO chart is on the next slide.

IPO Chart

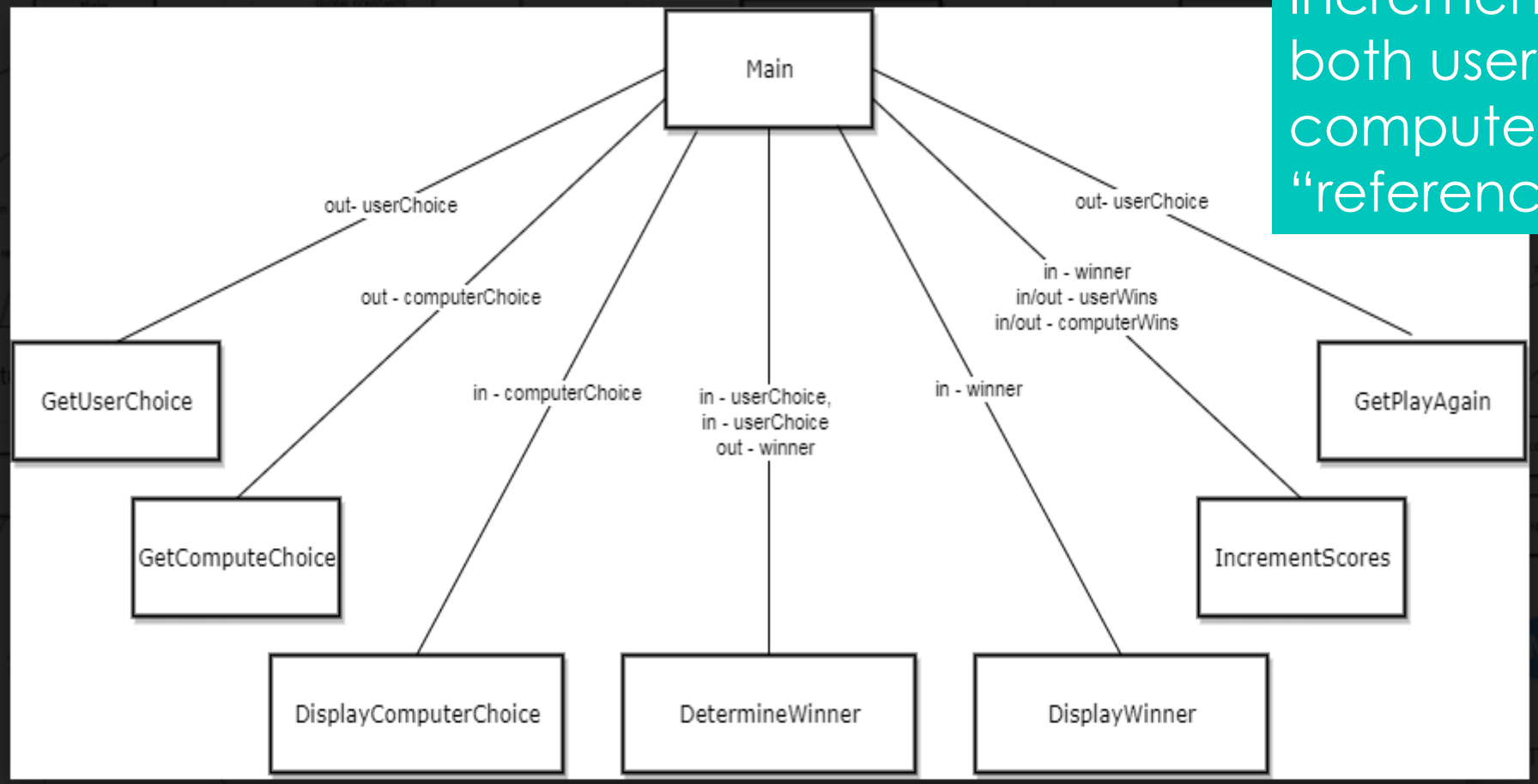
Input
- userChoice

Processing
- Get userChoice
- Generate computerChoice
- Display computerChoice
- Determine winner
- Display winner

Output
- computerChoice
- winner

In our final version we asked the user to play again and we kept track of scores. Should DetermineWinner, DisplayWinner and IncrementScores be one method or several?

Structure Chart



Increment scores changes both userWins and computerWins. They are "reference" parameters.

Parameter Passing Mechanisms

- Most simple parameters are passed into a method by value or by copy. If you change the value in the parameter inside the method, the variable in the calling code remains unchanged because it's a copy.
- userWins and computerWins MUST be altered by the method because the method can directly return only one value. They are passed by reference rather than by copy. A reference to the memory location is passed to the method. Any changes are reflected in the calling code because it's NOT a COPY.

New Syntax

```
static void IncrementScores(int winner,
                            ref int userWins, ref int computerWins)
{
    if (winner == USERWON)
        userWins++;
    else if (winner == COMPUTERWON)
        computerWins++;
}
```

Reference parameters

Doesn't return a value directly

GLOBAL CONSTANTS are safe. Why?

Changes are reflected in Main.

New Syntax

```
const int ROCK = 1;
const int PAPER = 2;
const int SCISSORS = 3;
const int USERWON = 1;
const int COMPUTERWON = 2;
const int TIE = 0;
static void Main(string[] args)
{
    …
    IncrementScores(winner, ref userWins, ref computerWins);
```

What makes these global?

In C# you need the ref in both heading of the method and the call.

The Whole Thing

- Let's look at my solution in Visual Studio and use the debugger to make sure you understand how reference parameters work.

You Try The Next Two

- You've done 1 – 4 from Lab 5. I'd like you to try 5 and 6.
- Create an IPO chart. It will help you identify methods.
- Create a structure chart. I've given you my structure charts in moodle. You don't have to use mine BUT you must write at least 3 methods for each problem.
  - Name the method
  - List it's parameters
  - Identify the return data type
- Write ONE METHOD AT A TIME and TEST IT.
- Write Main and test your application

What's Next

- Arrays
- Don't forget
  - Reading Quiz 5
  - Programming Quiz 5
  - Lab 5 – 6 problems