المقاييس البرمجية

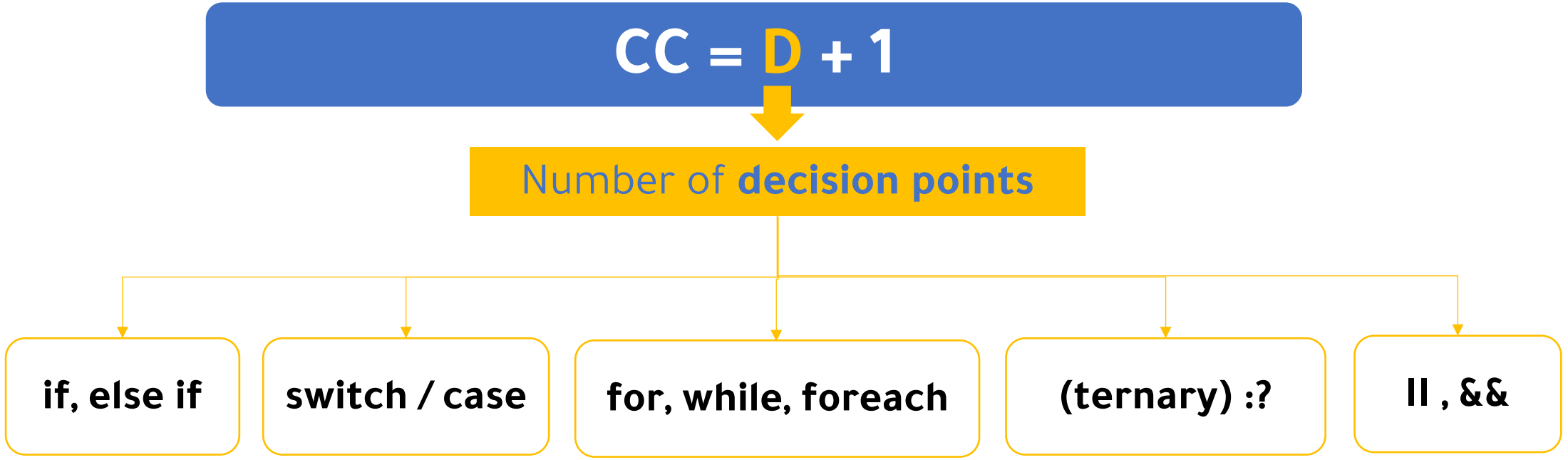
مقياس Cyclomatic complexity ، وعلاقته بتحديد حالات الاختبار

ما هو Cyclomatic Complexity (CC) ؟

- مقياس يُستخدم لتحديد عدد المسارات المنطقية (Logical Paths) في الدالة.
- كل قرار منطقي (شرط، تكرار، حالة...) يزيد التعقيد.
- هناك عدة طرق لحسابه، تعطي نفس النتيجة

ما هو Cyclomatic Complexity (CC) ؟

- طريقة 1:



ما هو Cyclomatic Complexity (CC) ؟

- طريقة 2: من مخطط التدفق

$$CC = E - N + 2P$$

- حيث:
  - E: (Number of Edges)
  - N: (Number of Nodes)
  - P: (Number of connected Graphs) (usually = 1 in a function)

جدول تقريبي لفهم درجات CC

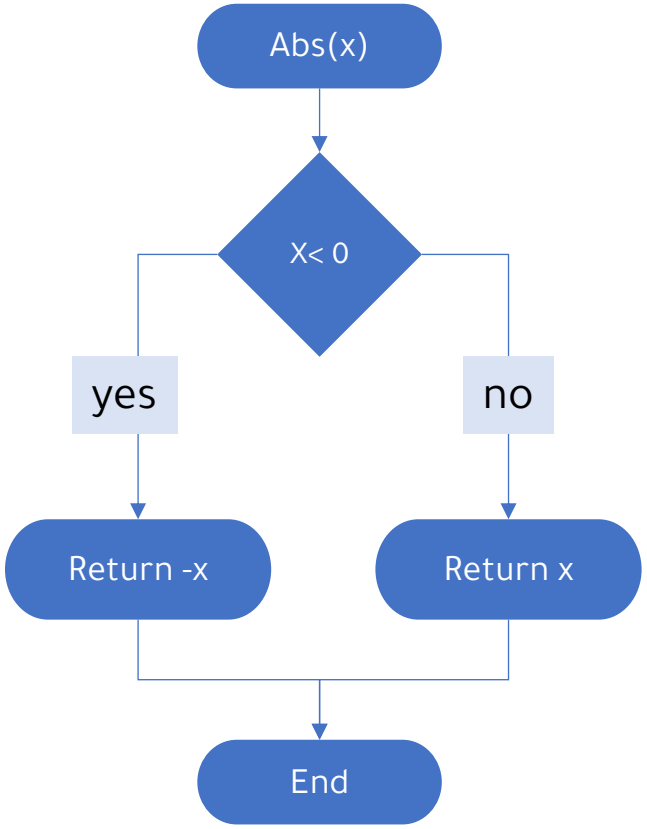
CC	مستوى التعقيد	ماذا يعني؟
1	بسيط جدًا	دالة خطية بلا فروع
2-5	مقبول	سهل الاختبار والفهم
6-10	متوسط	قابل للإدارة لكنه بحاجة مراقبة
11-20	معقد	يحتاج إعادة تنظيم
+20	عالي جدًا	خطر على المشروع ويصعب اختباره

جدول تقريبي لفهم درجات CC

كلما ارتفعت قيمة CC كلما ازدادت صعوبة الاختبار والفهم، وتناقصت قابلية إعادة الاستخدام، والصيانة.

أمثلة لحساب CC

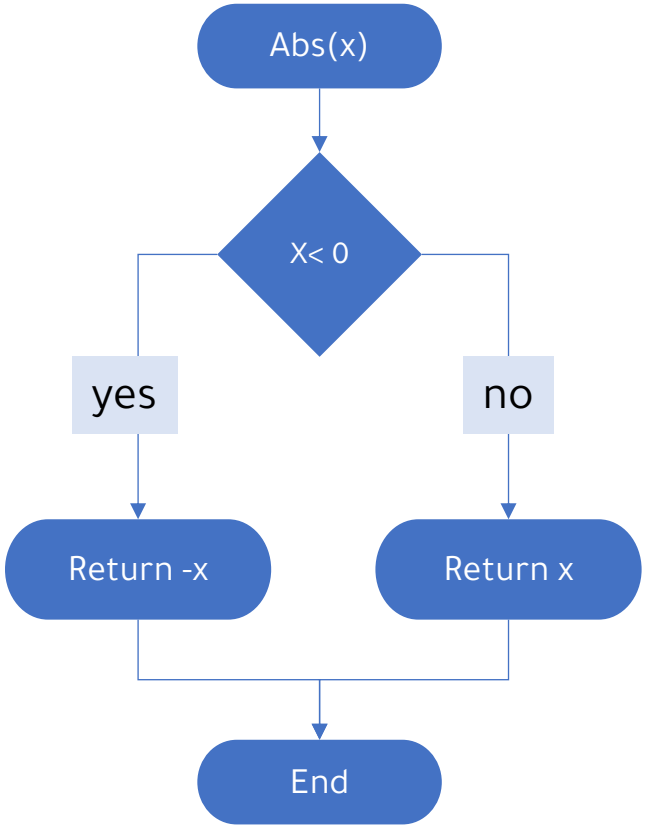
- مثال (1)




```
public int Abs(int x)
{
    if (x < 0)
        return -x;
    return x;
}
```

أمثلة لحساب CC

CC = E - N + 2P

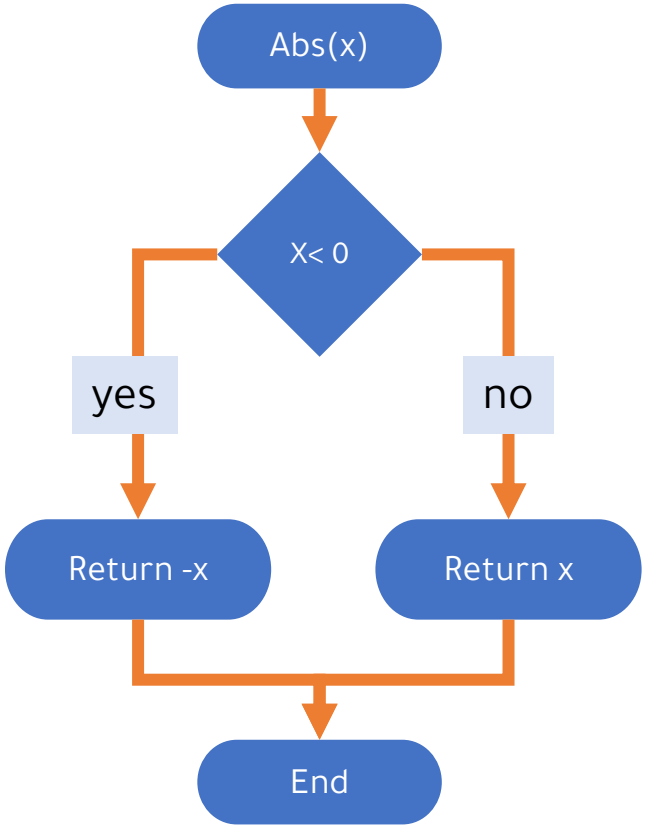


- مثال (1)

```
public int Abs(int x)
{
    if (x < 0)
        return -x;
    return x;
}
```

أمثلة لحساب CC

CC = E - N + 2P



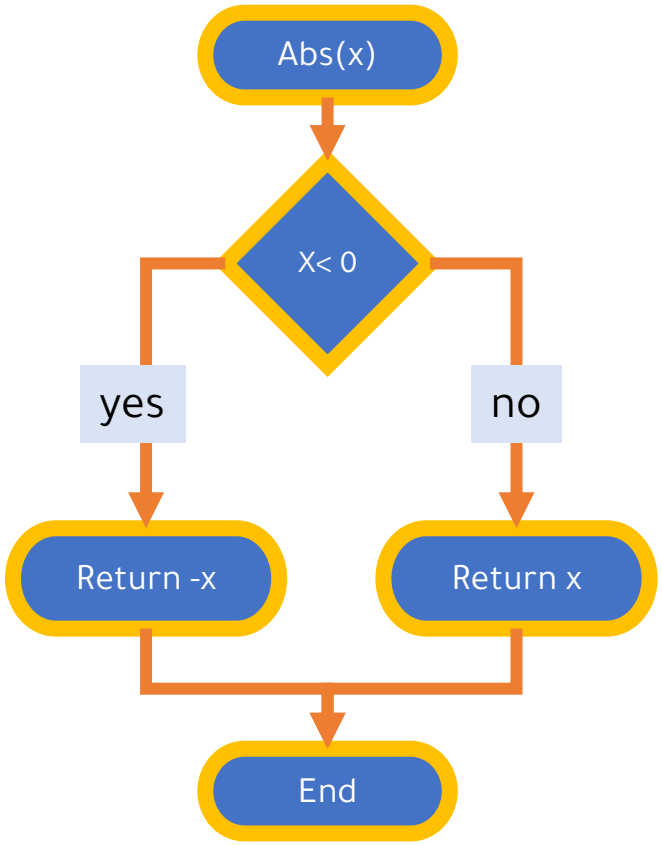
E = 5

- مثال (1)

```
public int Abs(int x)
{
    if (x < 0)
        return -x;
    return x;
}
```

أمثلة لحساب CC

CC = E - N + 2P



E = 5

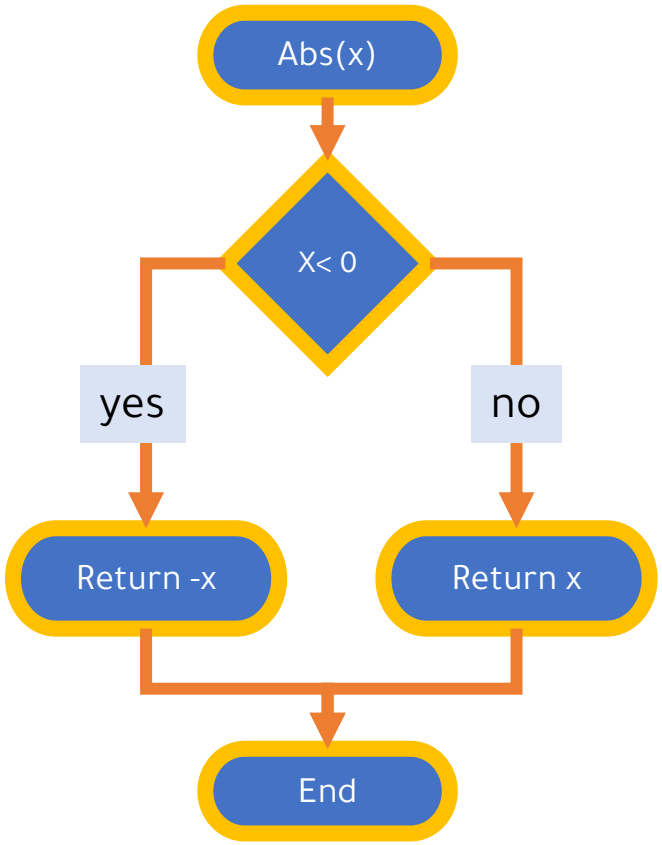
N = 5

- مثال (1)

```
public int Abs(int x)
{
    if (x < 0)
        return -x;
    return x;
}
```

أمثلة لحساب CC

CC = E - N + 2P



E = 5

N = 5

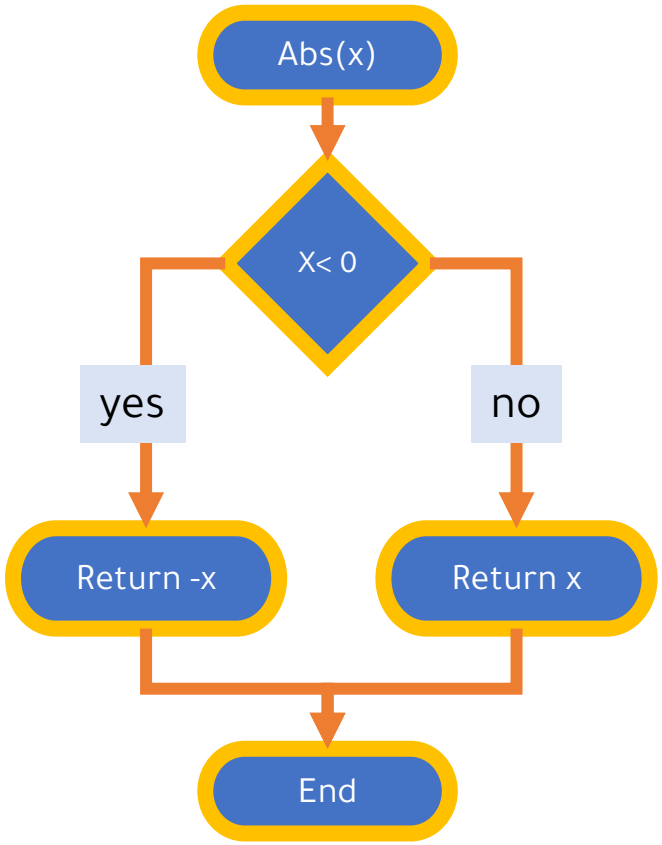
P = 1

- مثال (1)

```
public int Abs(int x)
{
    if (x < 0)
        return -x;
    return x;
}
```

أمثلة لحساب CC

CC = E - N + 2P



E = 5

N = 5

P = 1

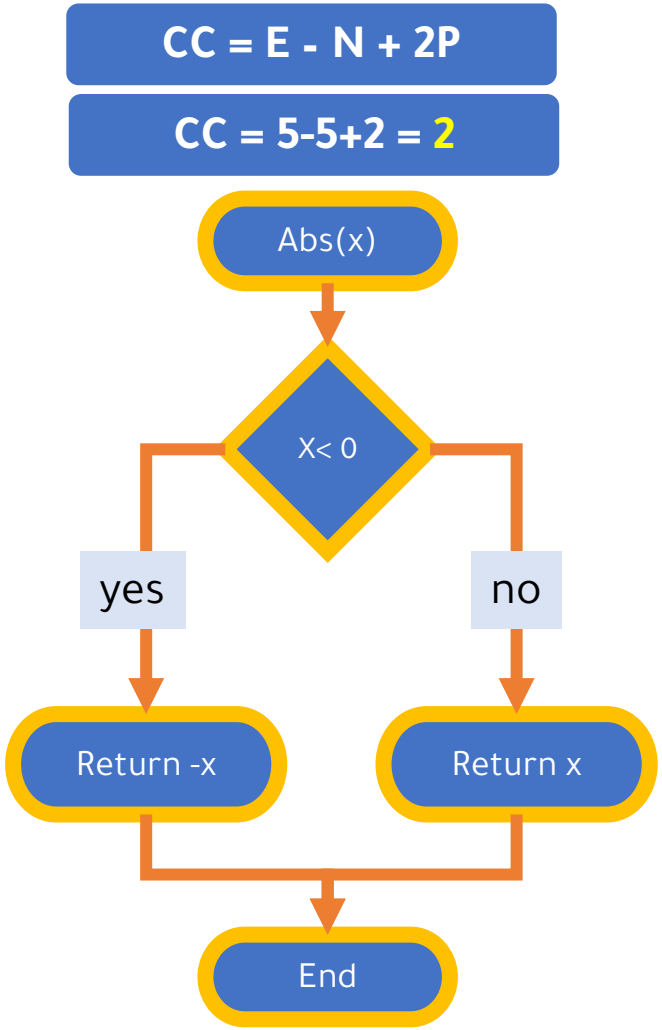
CC = 5 - 5 + 2 = 2

- مثال (1)

```
public int Abs(int x)
{
    if (x < 0)
        return -x;
    return x;
}
```

أمثلة لحساب CC

- مثال (1)

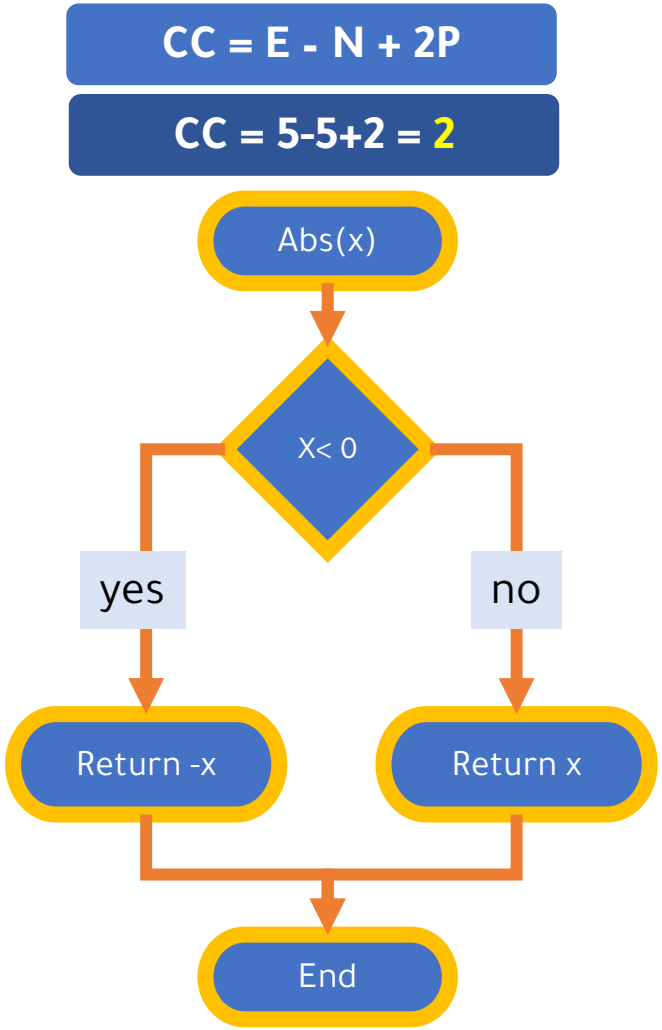



```
public int Abs(int x)
{
    if (x < 0)
        return -x;
    return x;
}
```

CC = D + 1

أمثلة لحساب CC

- مثال (1)




```
public int Abs(int x)
{
    if (x < 0)
        return -x;
    return x;
}
```

(1)

CC = D + 1

CC = 1 + 1 = 2

أمثلة لحساب CC

- مثال (2)

```
public string GetAgeCategory(int age)
{
    if (age < 13)
        return "Child";
    else if (age < 20)
        return "Teen";
    else if (age < 60)
        return "Adult";
    else
        return "Senior";
}
```

أمثلة لحساب CC

```
public string GetAgeCategory(int age)
{
    if (age < 13)
        return "Child";
    else if (age < 20)
        return "Teen";
    else if (age < 60)
        return "Adult";
    else
        return "Senior";
}
```

(1)

(2)

(3)

- مثال (2)

CC = D + 1

أمثلة لحساب CC

```
public string GetAgeCategory(int age)
{
    if (age < 13)
        return "Child";
    else if (age < 20)
        return "Teen";
    else if (age < 60)
        return "Adult";
    else
        return "Senior";
}
```

(1)

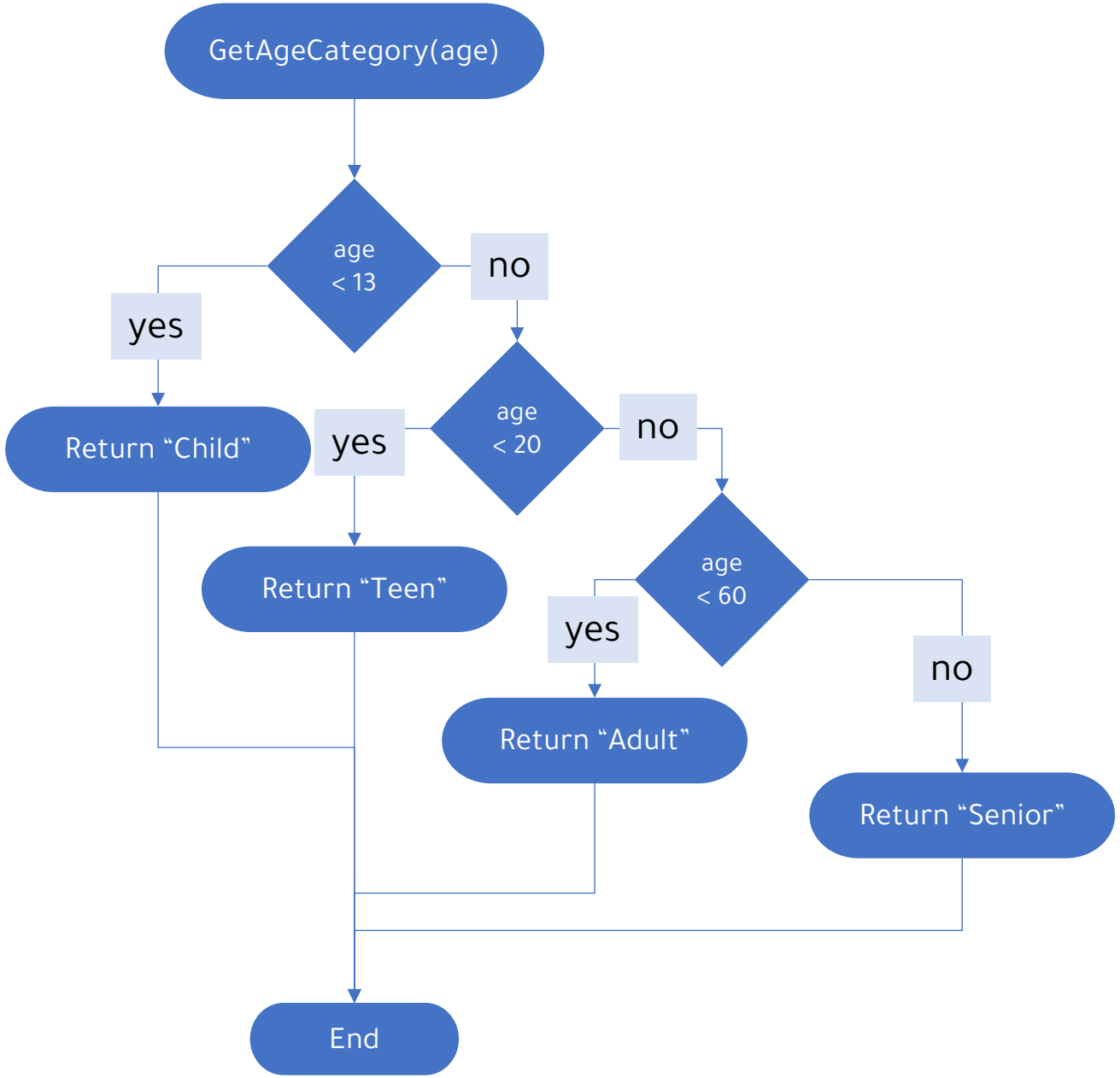
(2)

(3)

- مثال (2)

CC = D + 1

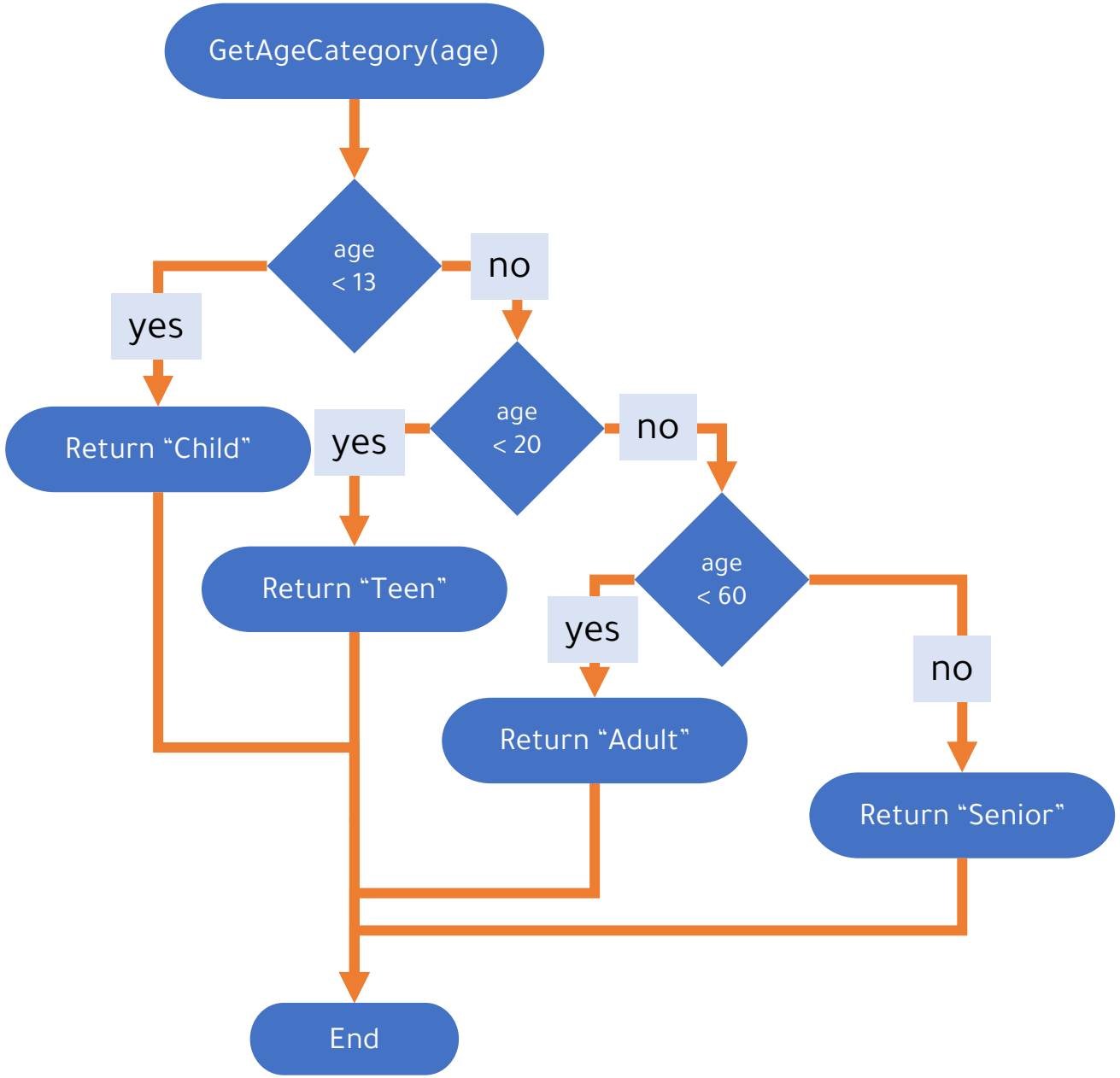
CC = 3 + 1 = 4



أمثلة لحساب CC

- مثال (2)

CC = E - N + 2P

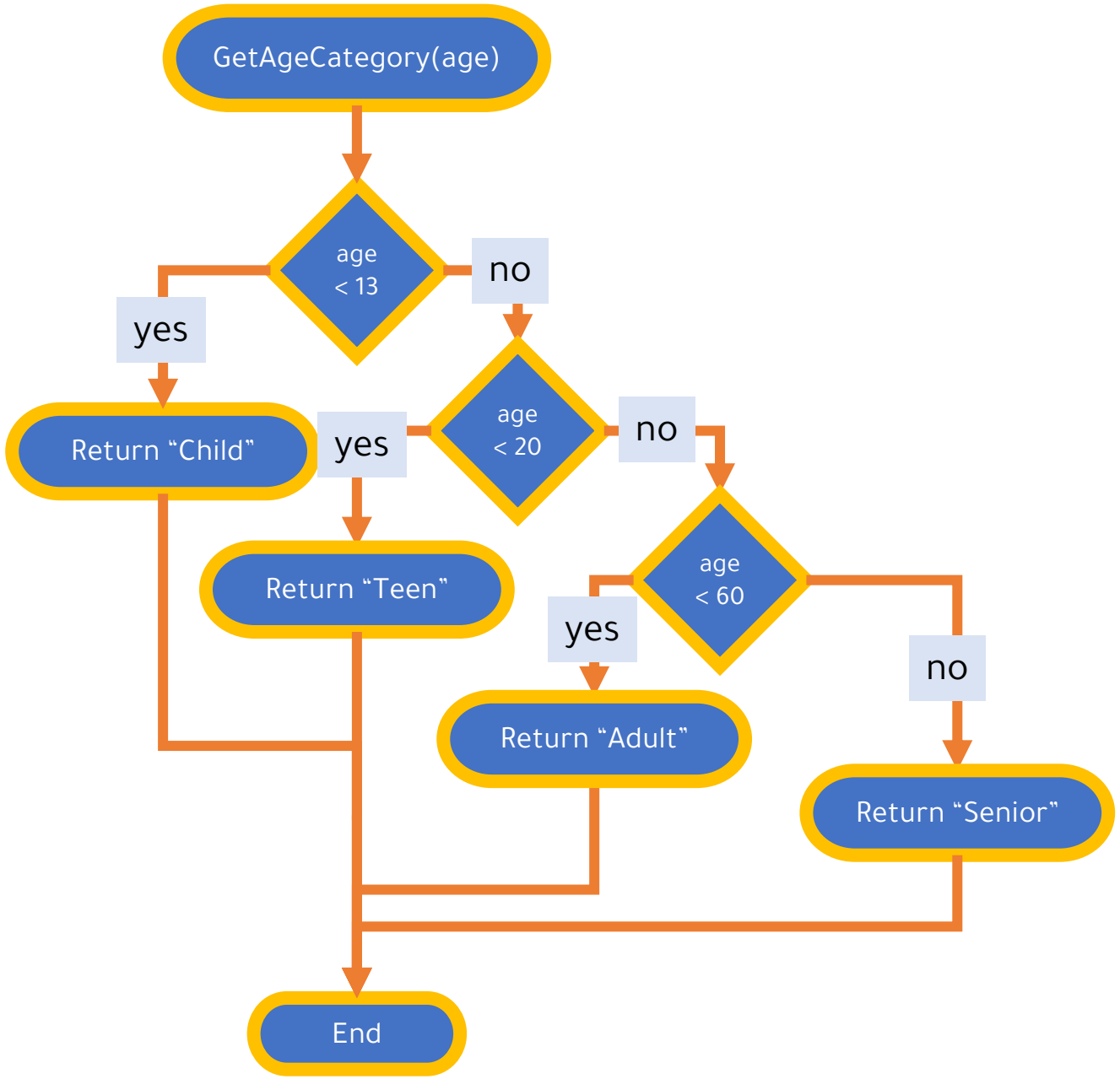


أمثلة لحساب CC

- مثال (2)

CC = E - N + 2P

E = 11

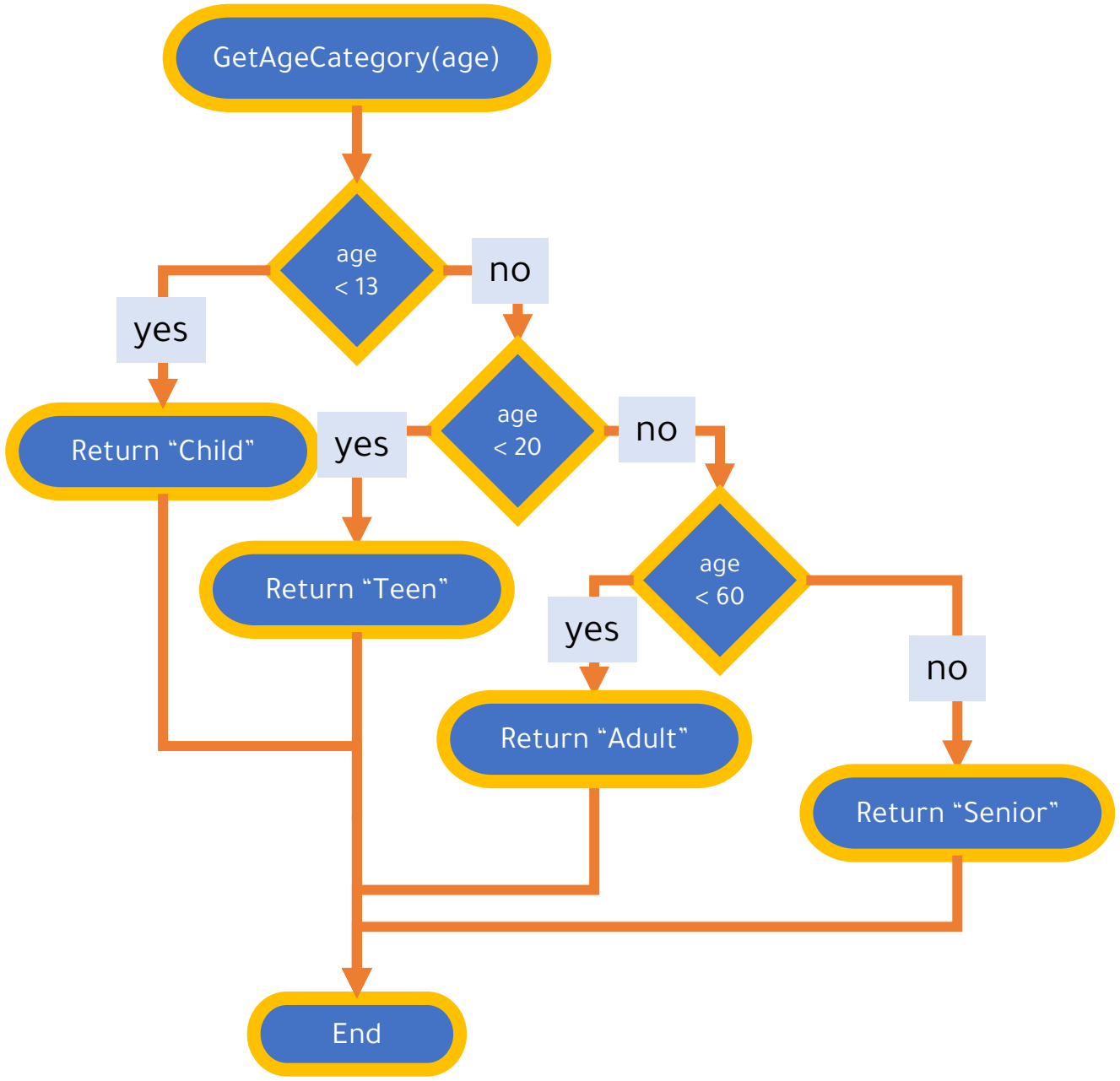


أمثلة لحساب CC

- مثال (2)

CC = E - N + 2P

E = 11 **N = 9**



أمثلة لحساب CC

- مثال (2)

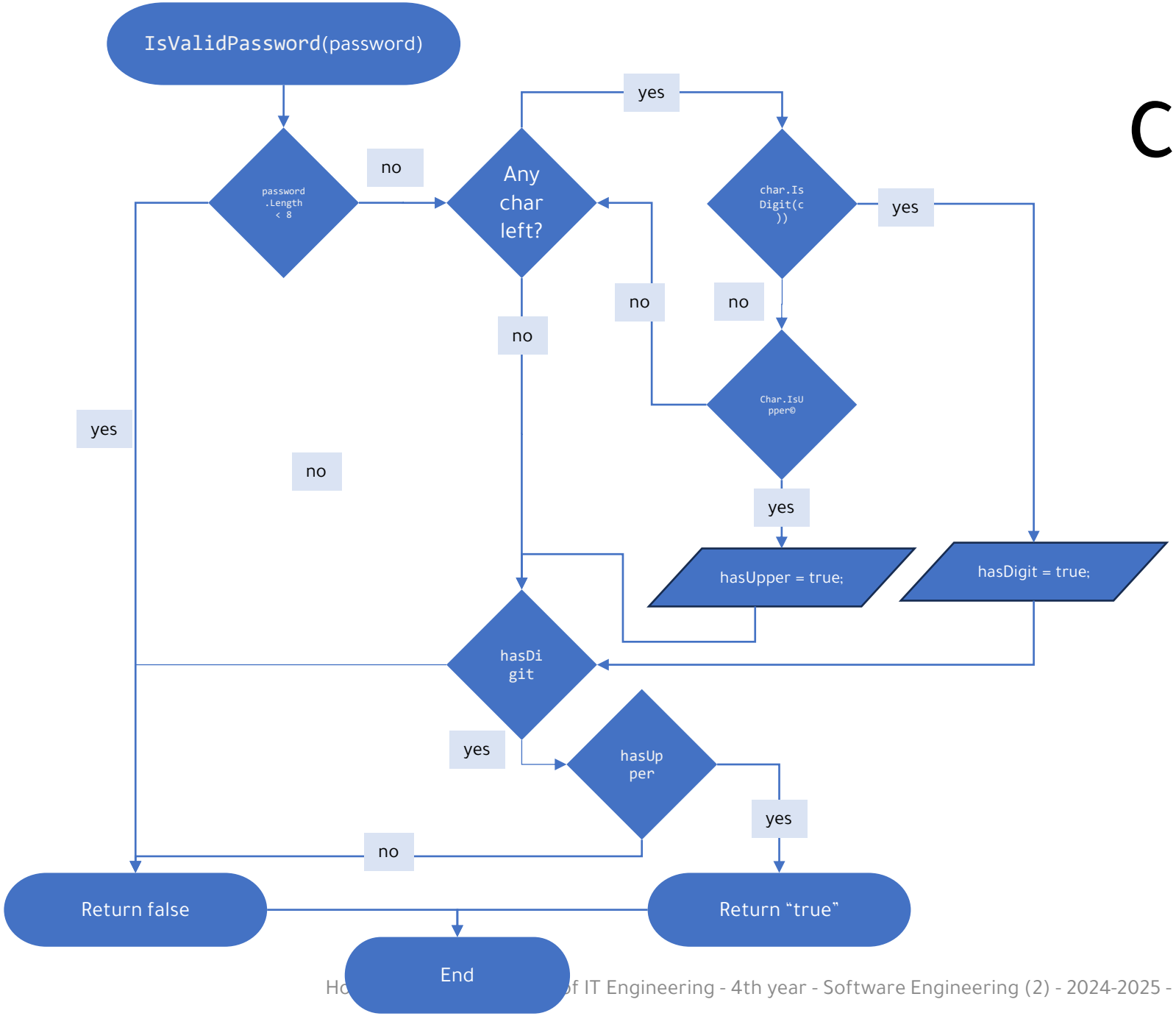
CC = E - N + 2P

E = 11 **N = 9** **P = 1**

CC = 11 - 9 + 2 = 4

أمثلة لحساب CC

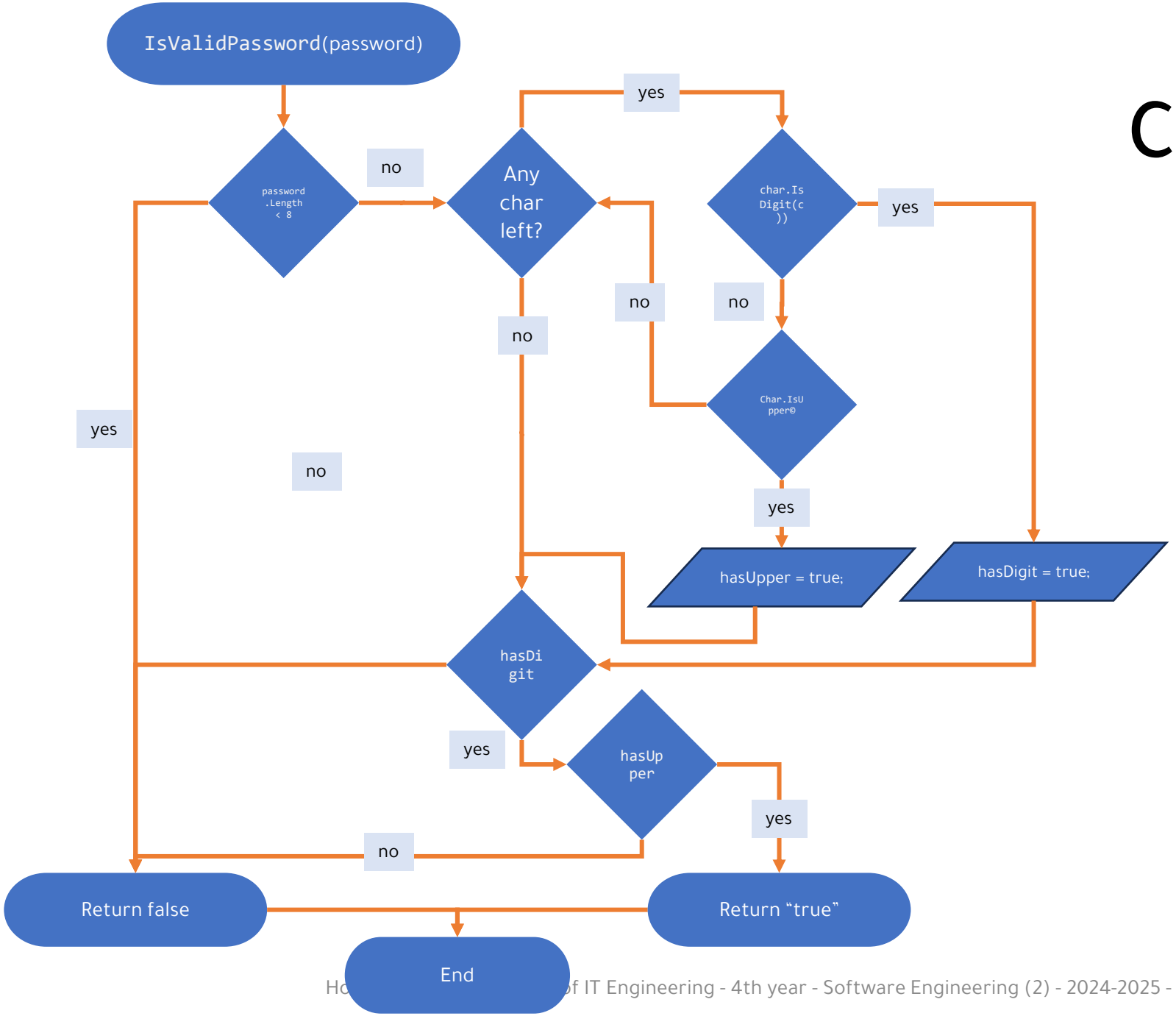
- مثال (3)




```
public bool IsValidPassword(string
password)
{
    if (password.Length < 8)
        return false;

    bool hasDigit = false;
    bool hasUpper = false;

    foreach (char c in password)
    {
        if (char.IsDigit(c))
            hasDigit = true;
        else if (char.IsUpper(c))
            hasUpper = true;
    }
    if (hasDigit && hasUpper)
        return true;

    return false;
}
```

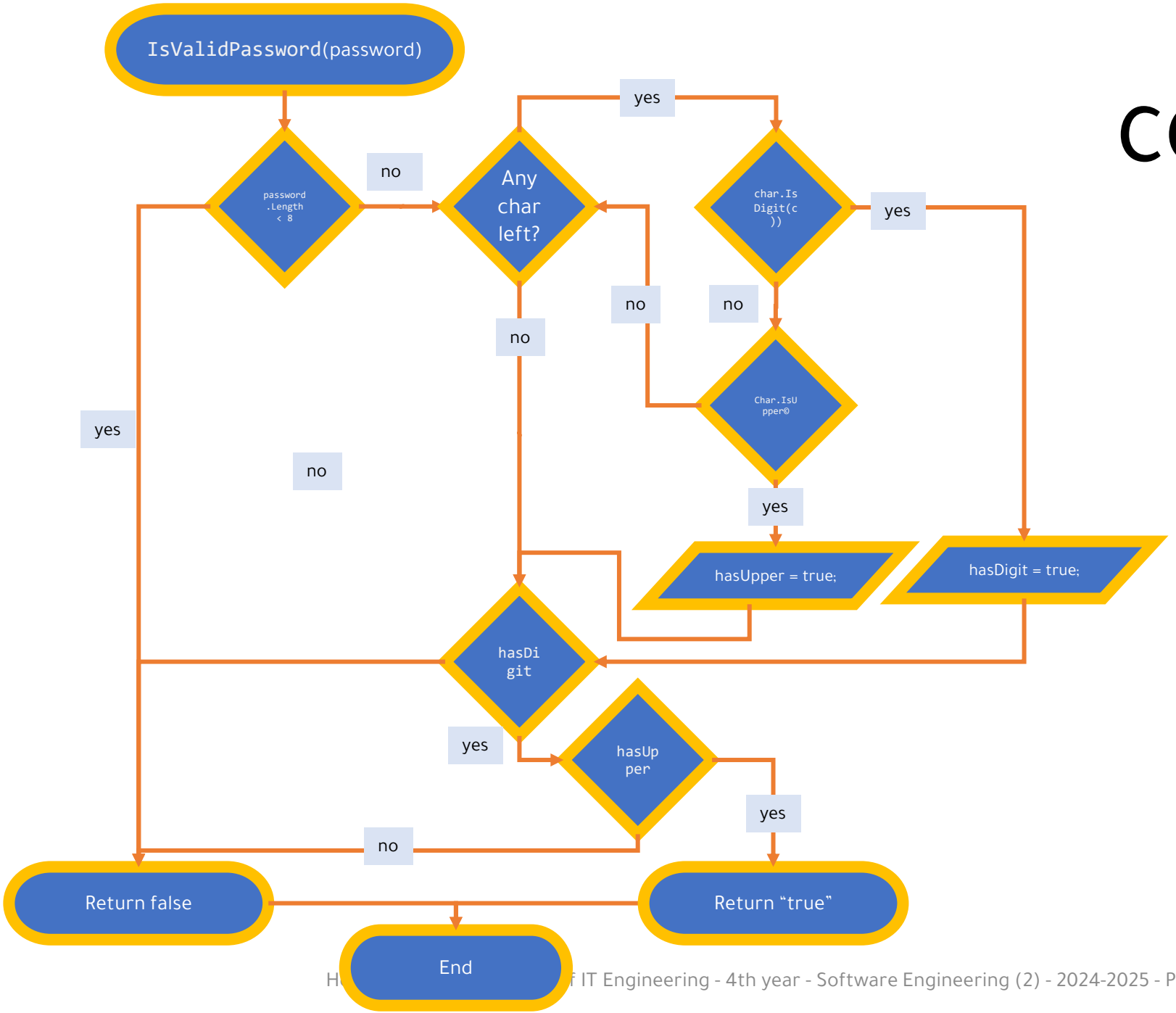
أمثلة لحساب CC

- مثال (3)



CC = E - N + 2P

E = 17

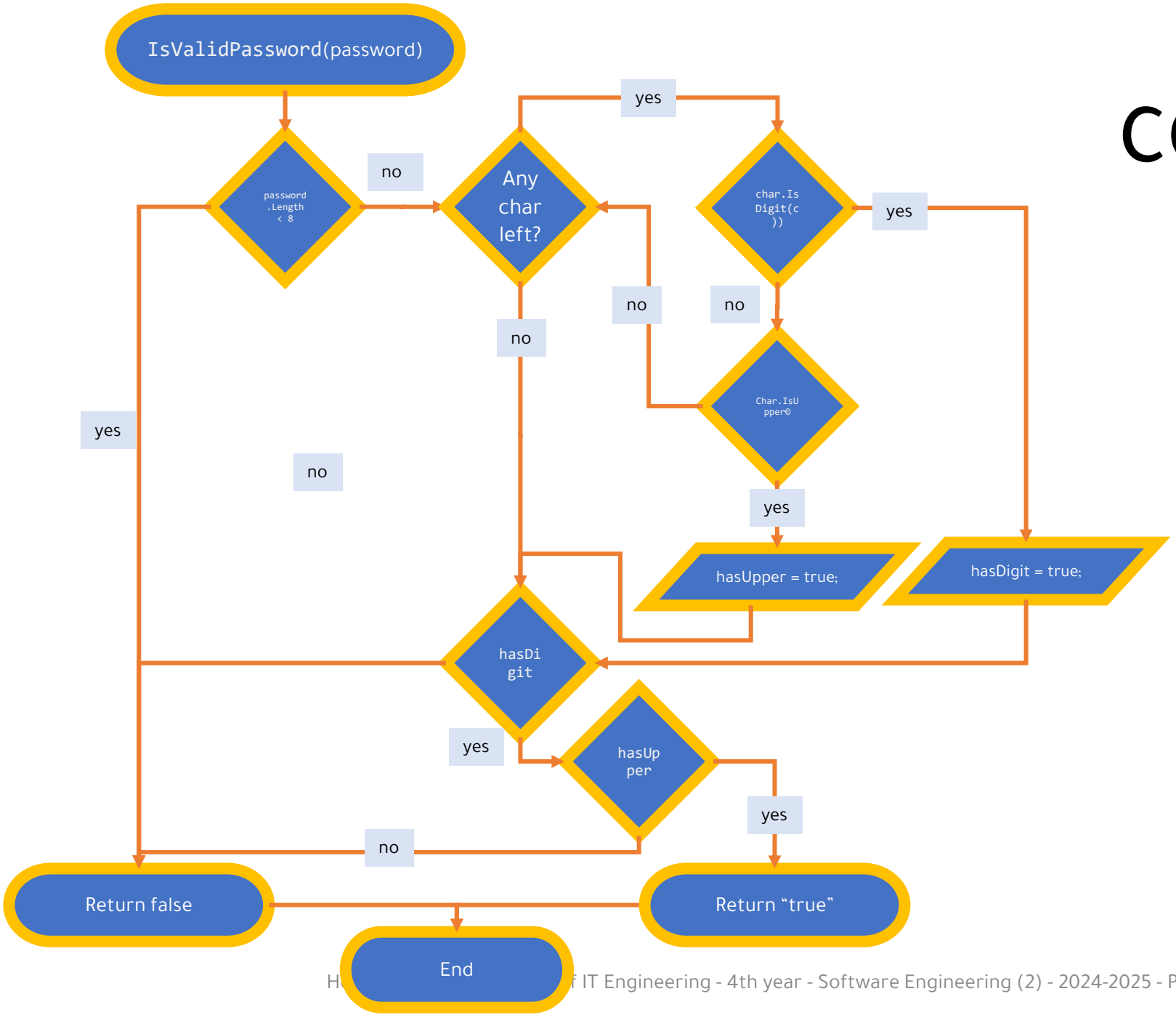


أمثلة لحساب CC

- مثال (3)

CC = E - N + 2P

E = 17 **N = 12**



أمثلة لحساب CC

- مثال (3)

CC = E - N + 2P

E = 17 **N = 12** **P = 1**

CC = 17 - 12 + 2 = 7

أمثلة لحساب CC

- مثال (3)

```
public bool IsValidPassword(string password)
{
    if (password.Length < 8)
        return false;

    bool hasDigit = false;
    bool hasUpper = false;

    foreach (char c in password)
    {
        if (char.IsDigit(c))
            hasDigit = true;
        else if (char.IsUpper(c))
            hasUpper = true;
    }
    if (hasDigit && hasUpper)
        return true;

    return false;
}
```

أمثلة لحساب CC

- مثال (3)

```
public bool IsValidPassword(string password)
{
    if (password.Length < 8)                    (1)
        return false;

    bool hasDigit = false;
    bool hasUpper = false;

    foreach (char c in password)                (2)
    {
        if (char.IsDigit(c))                    (3)
            hasDigit = true;
        else if (char.IsUpper(c))               (4)
            hasUpper = true;
    }
    if (hasDigit && hasUpper)                   (5)  (6)
        return true;

    return false;
}
```

CC = D + 1

CC = 6 + 1 = 7

```
public string GetRiskLevel(int age, bool smoker, bool hasCondition)
{
    if (age < 18)
        return "Low";
    if (age >= 18 && age < 40)
    {
        if (smoker)
        {
            if (hasCondition)
                return "Medium";
            else
                return "Low";
        }
        else
        {
            if (hasCondition)
                return "Medium";
            else
                return "Low";
        }
    }
    if (age >= 40)
    {
        if (smoker && hasCondition)
            return "High";
        else if (smoker || hasCondition)
            return "Medium";
        else
            return "Low";
    }
    return "Unknown";
}
```

أمثلة لحساب CC

- مثال (4)

```
public string GetRiskLevel(int age, bool smoker, bool hasCondition)
{
    if (age < 18)                                          (1)
        return "Low";
    if (age >= 18 && age < 40)                             (2)  (3)
    {
        if (smoker)                                        (4)
        {
            if (hasCondition)                              (5)
                return "Medium";
            else
                return "Low";
        }
        else
        {
            if (hasCondition)                              (6)
                return "Medium";
            else
                return "Low";
        }
    }
    if (age >= 40)                                         (7)
    {
        if (smoker && hasCondition)                        (8)  (9)
            return "High";
        else if (smoker || hasCondition)                   (10) (11)
            return "Medium";
        else
            return "Low";
    }
    return "Unknown";
}
```

أمثلة لحساب CC

- مثال (4)

CC = D + 1

CC = 11 + 1 = 12

```
public string GetRiskLevel(int age, bool smoker, bool hasCondition)
{
    if (age < 18)                                          (1)
        return "Low";
    if (age >= 18 && age < 40)                             (2)  (3)
    {
        if (smoker)                                        (4)
        {
            if (hasCondition)                              (5)
                return "Medium";
            else
                return "Low";
        }
        else
        {
            if (hasCondition)                              (6)
                return "Medium";
            else
                return "Low";
        }
    }
    if (age >= 40)                                         (7)
    {
        if (smoker && hasCondition)                        (8)  (9)
            return "High";
        else if (smoker || hasCondition)                   (10) (11)
            return "Medium";
        else
            return "Low";
    }
    return "Unknown";
}
```

أمثلة لحساب CC

- مثال (4)

CC = D + 1

CC = 11 + 1 = 12

تعقيد عال!

لنقم بإعادة تصميم التابع (Refactoring)

أمثلة لحساب CC

```
public string GetRiskLevel(int age, bool smoker, bool hasCondition)
{
    if (IsChild(age))                                    (1)
        return "Low";

    if (IsYoungAdult(age))                               (2)
        return EvaluateYoungAdult(smoker, hasCondition);

    if (IsOlder(age))                                    (3)
        return EvaluateOlder(smoker, hasCondition);

    return "Unknown";
}

private bool IsChild(int age) => age < 18;
private bool IsYoungAdult(int age) => age >= 18 && age < 40;
private bool IsOlder(int age) => age >= 40;

private string EvaluateYoungAdult(bool smoker, bool hasCondition)
{
    return (hasCondition) ? "Medium" : "Low";
}

private string EvaluateOlder(bool smoker, bool hasCondition)
{
    if (smoker && hasCondition)
        return "High";
    if (smoker || hasCondition)
        return "Medium";
    return "Low";
}
```

CC = 3 + 1 = 4

- مثال (4)

لنقم بإعادة تصميم التابع (Refactoring)

أمثلة لحساب CC

- مثال (4)

```
public string GetRiskLevel(int age, bool smoker, bool hasCondition)
{
    if (IsChild(age))                                          (1)
        return "Low";

    if (IsYoungAdult(age))                                     (2)
        return EvaluateYoungAdult(smoker, hasCondition);

    if (IsOlder(age))                                          (3)
        return EvaluateOlder(smoker, hasCondition);

    return "Unknown";
}

private bool IsChild(int age) => age < 18;                     (0)
private bool IsYoungAdult(int age) => age >= 18 && age < 40;   (1)
private bool IsOlder(int age) => age >= 40;                    (0)

private string EvaluateYoungAdult(bool smoker, bool hasCondition)
{
    return (hasCondition) ? "Medium" : "Low";                  (1)
}

private string EvaluateOlder(bool smoker, bool hasCondition)
{
    if (smoker && hasCondition)                                (1)  (2)
        return "High";
    if (smoker || hasCondition)                                (3)  (4)
        return "Medium";
    return "Low";
}
```

CC = 3 + 1 = 4

CC = 0 + 1 = 1

CC = 1 + 1 = 2

CC = 1 + 1 = 1

CC = 1 + 1 = 2

CC = 4 + 1 = 5

لنقم بإعادة تصميم التابع (Refactoring)

أمثلة لحساب CC

- مثال (4)

الفائدة من عملية Refactoring التي قمنا بها

خفضنا قيمة cc للتابع الأساسي من 12 إلى 4

حققنا مبدأ Single Responsibility

المعيار	تصميم قديم (دالة وحدة)	تصميم حديث (تفكيك)
الاختبار	صعب	سهل لكل جزء
القابلية للفهم	منخفضة	مرتفعة
إعادة الاستخدام	مستحيلة تقريبًا	ممكنة بسهولة
التوسعة لاحقًا	خطرة	آمنة عبر إضافة توابع
CC عالية أين؟	مركزية	موزعة

ما علاقة مقياس CC بالاختبار

- كل مسار منطقي يمثل حالة اختبار يجب تغطيتها لضمان عمل الكود بشكل صحيح.
- **عدد حالات الاختبار الدنيا المطلوبة ≤ قيمة CC**

- إذا كان **CC = 8**، نحتاج إلى تغطية **8 حالات اختبار (على الأقل)**

تطبيق عملي:

- إنشاء المشروع الأساسي ومشروع الاختبار ضمن ال solution الأساسي وإضافة المرجعيات

```
dotnet new sln -n RiskLevelApp
dotnet new classlib -n RiskLevelApp.Core
dotnet new xunit -n RiskLevelApp.Tests

dotnet sln add RiskLevelApp.Core/RiskLevelApp.Core.csproj
dotnet sln add RiskLevelApp.Tests/RiskLevelApp.Tests.csproj

dotnet add RiskLevelApp.Tests/RiskLevelApp.Tests.csproj reference
RiskLevelApp.Core/RiskLevelApp.Core.csproj
```

تطبيق عملي:

نضيف الملف:

RiskLevelApp.Core/RiskEvaluator.cs

```
namespace RiskLevelApp.Core;

public static class RiskEvaluator
{
      static string GetRiskLevel(int age, bool smoker, bool hasCondition)
      {
          if (age < 18 )
              return "Low";

          if (age >= 18 && age < 40)
          {
              if (smoker)
              {
                  if (hasCondition)
                      return "Medium";
                  else
                      return "Low";
              }
              else
              {
                  if (hasCondition)
                      return "Medium";
                  else
                      return "Low";
              }
          }
          if (age >= 40)
          {
              if (smoker && hasCondition)
                  return "High";
              else if (smoker || hasCondition)
                  return "Medium";
              else
                  return "Low";
          }
          return "Unknown";
      }
}
```

```
namespace RiskLevelApp.Core;

public static class RiskEvaluator
{
      static string GetRiskLevel(int age, bool smoker, bool hasCondition)
      {
          if (age < 18 && age > 0 )
              return "Low";

          if (age >= 18 && age < 40)
          {
              if (smoker)
              {
                  if (hasCondition)
                      return "Medium";
                  else
                      return "Low";
              }
              else
              {
                  if (hasCondition)
                      return "Medium";
                  else
                      return "Low";
              }
          }
          if (age >= 40)
          {
              if (smoker && hasCondition)
                  return "High";
              else if (smoker || hasCondition)
                  return "Medium";
              else
                  return "Low";
          }
          return "Unknown";
      }
}
```

تطبيق عملي:

- لنحسب **CC** باستخدام إحدى أدوات تحليل الكود البرمجي.
- لنجرب أداة **Codalyze**
- بعد تنصيب الـextension ضمن **VS Code**
- نضغط **CTRL + SHIFT + P**
- نكتب: **Codalyze** ونختار الأمر الأول الذي يظهر ضمن قائمة الأوامر.

Code Complexity Report

تطبيق عملي:

> So if you want to go fast, if you want to get done quickly, if you want your code to be easy to write, make it easy to read.
>
> — Clean Code: A Handbook of Agile Software Craftsmanship *Robert C. Martin*

Function Name	Start Line	End Line	Cyclomatic Complexity (Threshold: 10)	Lines of Code (Threshold: 50)	Parameter Count (Threshold: 4)
Program::Main	5	8	1	4	0
Program::GetRiskLevel	10	41	⚠ 12	32	3

Source Code: 2da899a4-4956-11f0-8edb-263d64ae3d6b.cs

Generated by Codalyze on 2025-06-14 19:31

التقرير الناتج

تطبيق عملي:

Line	End Line	Cyclomatic Complexity (Threshold: 10)
8		1
41		⚠ 12

تعقيد مرتفع !

```
namespace RiskLevelApp.Core;

public static class RiskEvaluator
{
    public static string GetRiskLevel(int age, bool smoker, bool hasCondition)
    {
        if (IsChild(age))
            return "Low";

        if (IsYoungAdult(age))
            return EvaluateYoungAdult( hasCondition);

        if (IsOlder(age))
            return EvaluateOlder(smoker, hasCondition);

        return "Unknown";
    }

    private static bool IsChild(int age) => age > 0 && age < 18;
    private static bool IsYoungAdult(int age) => age >= 18 && age < 40;
    private static bool IsOlder(int age) => age >= 40;

    private static string EvaluateYoungAdult(bool hasCondition)
    {
        return (hasCondition) ? "Medium" : "Low";
    }

    private static string EvaluateOlder(bool smoker, bool hasCondition)
    {
        if (smoker && hasCondition)
            return "High";
        if (smoker || hasCondition)
            return "Medium";
        return "Low";
    }
}
```

تطبيق عملي:

نعمل على خفض التعقيد من خلال إعادة تصميم الكود (Refactoring)

تطبيق عملي:

Code Complexity Report

> So if you want to go fast, if you want to get done quickly, if you want your code to be easy to write, make it easy to read.
>
> — Clean Code: A Handbook of Agile Software Craftsmanship *Robert C. Martin*

Function Name	Start Line	End Line	Cyclomatic Complexity (Threshold: 10)	Lines of Code (Threshold: 50)	Parameter Count (Threshold: 4)
RiskEvaluator::GetRiskLevel	5	17	4	10	3

Source Code: aae69dec-495d-11f0-8edb-263d64ae3d6b.cs

تطبيق عملي:

Cyclomatic Complexity (Threshold: 10)	Lines of Code (Threshold
4	10

انخفضـــــــت قيمـــــــة CC بشكل ملحوظ

تطبيق عملي:

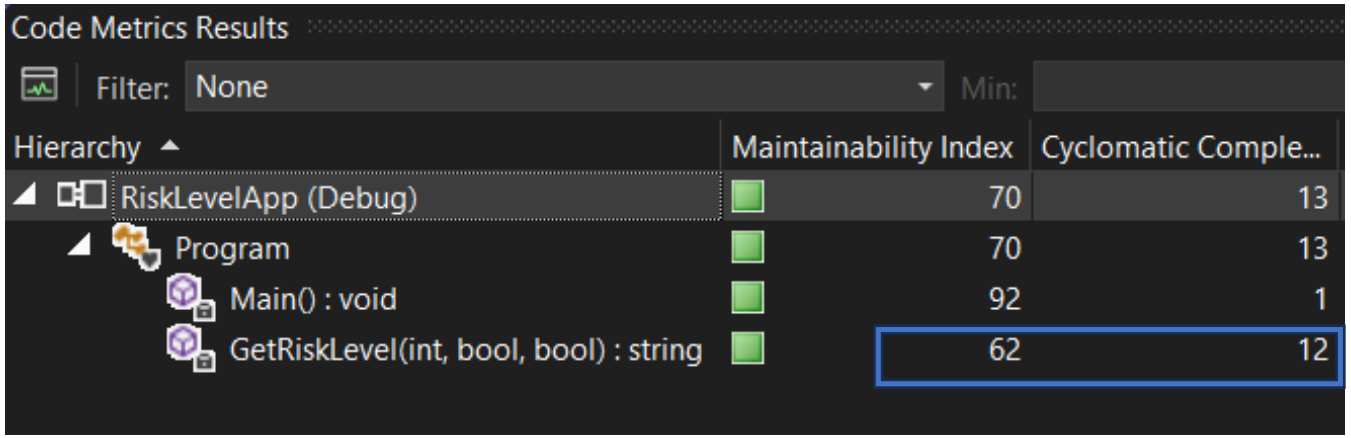
- Visual studio

أو

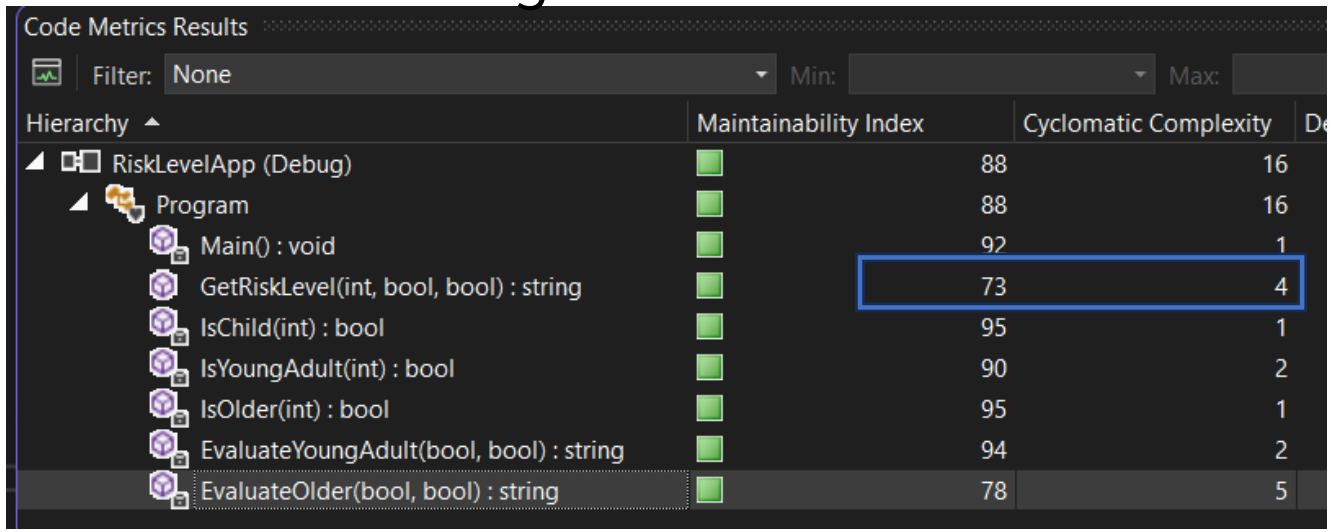
- Analyze -> Calculate Code Metrics -> For Solution
- Analyze -> Calculate Code Metrics -> For <ProjectName>

تطبيق عملي:

Before Refactoring



After Refactoring



انخفض التعقيد (CC)

وازدادت قابلية الصيانة (Maintainability Index)

تطبيق عملي:

- بعد أن قمنا بإعادة تصميم الكود، سنقوم بكتابة unit tests من أجل التوابع التي عرفناها ضمنه
- بعد حساب CC يمكننا بسهولة معرفة العدد الأدنى لحالات الاختبار التي يجب تغطيتها
- CC = 4 ، أي أن علينا كتابة 4 حالات اختبار على الأقل.

تطبيق عملي:

RiskLevelApp.Tests/RiskEvaluatorTests.cs

```
using Xunit;
using RiskLevelApp.Core;

namespace RiskLevelApp.Tests;

public static class RiskEvaluatorTests
{
    [Fact]
    public static void GetRiskLevel_Should_ReturnLow_When_AgeUnder18()
    {
        var result = RiskEvaluator.GetRiskLevel(12, false, false);
        Assert.Equal("Low", result);
    }

    [Fact]
    public static void GetRiskLevel_Should_ReturnMedium_When_YoungAdultWithCondition()
    {
        var result = RiskEvaluator.GetRiskLevel(25, false, true);
        Assert.Equal("Medium", result);
    }

    [Fact]
    public static void GetRiskLevel_Should_ReturnHigh_When_OlderSmokerAndHasCondition()
    {
        var result = RiskEvaluator.GetRiskLevel(65, true, true);
        Assert.Equal("High", result);
    }

    [Fact]
    public static void GetRiskLevel_Should_ReturnUnknown_When_AgeIsNegative()
    {
        var result = RiskEvaluator.GetRiskLevel(-1, false, false);
        Assert.Equal("Unknown", result);
    }

}
```

تطبيق عملي:

- لنختبر التوابع الفرعية كل على حده
- حاليا لا يمكننا اختبارها كونها معرفة private
- لننقلها إلى صف آخر ونسميه: RiskEvaluationHelpers.cs
- ثم ننشئ صفا جديدا للاختبار أيضاً باسم: RiskEvaluationHelpersTests.cs

تطبيق عملي:

```
namespace RiskLevelApp.Core;
public static class RiskEvaluationHelpers
{
    public static string EvaluateYoungAdult(bool hasCondition)
    {
        return (hasCondition) ? "Medium" : "Low";
    }

    public static string EvaluateOlder(bool smoker, bool hasCondition)
   {
        if (smoker && hasCondition)
            return "High";
        if (smoker || hasCondition)
            return "Medium";
        return "Low";
    }
}
```

تطبيق عملي:

- يمكننا اعتبار اختيار التابع الرئيسي (GetRiskLevel) شكلاً مبسطاً من أشكال اختبار التكامل ، حيث نتأكد أنّ نتيجة التكامل منطقية من حيث توقع النهاية، بدون الحاجة لمعرفة تفاصيل كل تابع داخلي.

وظيفة

- أعطى أحد العملاء التابع التالي لتحديد درجة الأهلية للحصول على قرض. طُلب منك:
- تحليل التعقيد الحلقي Cyclomatic Complexity للتابع.
- تحسين الكود عبر تفكيكه لتوابع فرعية لخفض التعقيد Refactoring.
- كتابة اختبارات وحدة للتوابع المفككة.
- كتابة اختبار تكامل للتابع الرئيسي بعد إعادة هيكلته.

وظيفة

LoanEvaluator/LoanEvaluator.cs

```
namespace LoanApp
{
    public class LoanEvaluator
    {
        public  string GetLoanEligibility(int income, bool hasJob, int creditScore, int dependents, bool ownsHouse)
        {
            if (income < 2000)
                return "Not Eligible";

            if (hasJob)
            {
                if (creditScore >= 700)
                {
                    if (dependents == 0)
                        return "Eligible";
                    else if (dependents <= 2)
                        return "Review Manually";
                    else
                        return "Not Eligible";
                }
                else if (creditScore >= 600)
                {
                    if (ownsHouse)
                        return "Review Manually";
                    else
                        return "Not Eligible";
                }
                else
                {
                    return "Not Eligible";
                }
            }
            else
            {
                if (creditScore >= 750 && income > 5000 && ownsHouse)
                    return "Eligible";
                else if (creditScore >= 650 && dependents == 0)
                    return "Review Manually";
                else
                    return "Not Eligible";
            }
        }
    }
}
```

LoanEvaluator.Tests/LoanEvaluatorTests.cs

```
using Xunit;
using LoanApp;

namespace LoanApp.Tests
{
    public class LoanEvaluatorTests
    {
        [Fact]
        public void GetLoanEligibility_Should_Return_NotEligible_When_Income_Low()
        {
            var evaluator = new LoanEvaluator();
            var result = evaluator.GetLoanEligibility(1500, true, 800,  5, true);
            Assert.Equal("Not Eligible", result);
        }
    }
}
```

ملاحظات للوظيفة

- يمكن استخدام VS Code أو Visual Studio.
- يجب استخدام xUnit في الاختبار.
- يجب حساب Cyclomatic Complexity يدويًا أولًا، ثم استخدام أدوات مثل Codelyze للتأكيد.
- يجب تغطية جميع حالات التنفيذ Branches في الاختبارات.
- يجب شرح السبب وراء أي تحسين تم إجراؤه.