Systems and Control Eng. Dept.

Computer Programming

First Year Class

Lecture 1

Programming Introduction

Mr. Abdulhameed N. Hameed

Syllabus

Computer Programming	**First class/Semester II**
❖ Introduction to programing and types of programming languages.	
❖ Study of program development steps and flowchart.	
❖ Introduction to C++ programming.	
❖ Structured Programming (Variables, Input, Memory).	
❖ Operators: Assignment operators, Arithmetic operators, Increment and decrement operators, Decision making operators, Conditional operator.	
❖ Selection statements-1: if single-selection statement, if..else.	
❖ Selection statements-2: Nested if..else, switch multiple-selection statement.	
❖ Repetition Statements: while repetition statement, for repetition statement, do..while, break and continue statements.	
❖ One-dimensional Arrays: Array elements, Approaches of initialization arrays, Array of characters, Searching arrays.	
❖ Multidimensional Arrays: Array elements, Approaches of initialization arrays, Array of characters, Searching arrays.	

Textbook and Software:

- The C++ Programming Language (4th Edition) By: Bjarne Stroustrup.

- Programming: Principles and Practice Using C++ (2nd Edition) By: Bjarne Stroustrup.

- The software that is used in this course is the Code::Blocks as a C++ compiler.

صياغة البرامج ولغات البرمجة
Programming & programming languages

بعد أن تكلمنا في الكورس الاول عن برامج نظم التشغيل و برامج التطبيقات يبقى سؤال

من يكتب هذه البرامج؟

وكيف كتبت؟

وبأي لغة؟

وما هي الخطوات العملية التي اتبعت لبناء هذه البرامج؟

بعض المفاهيم والتعاريف الاساسية

- جهاز الحاسب الآلي هو آلة تنفذ ما يأتيها من أوامر بدقة عالية.
- حيث تكون هذه الأوامر مكتوبة فيما يسمى (برنامج).
- وجميع البرامج تكون مكتوبة على هيئة سلسلة من الأوامر اليسيرة التي ينفذها الحاسب الآلي لتحقيق هدف ما او حل مشكلة معينة.
- هذه الأوامر تكتب بلغة معينة يفهمها الانسان اضافة الى جهاز الحاسب.

لغات البرمجة Programming Languages:

هي مجموعة القواعد والاوامر التي توفر طريقة لصياغة تعليمات الانسان (المبرمج) لتوصيلها الى المعالج وهي حلقة الوصل بين الانسان وجهاز الحاسب.

البرنامج Program:

هي مجموعة من التعليمات المتسلسلة والمترابطة والتي توجه الحاسوب لأداء عمل معين على البيانات الخام وتنفيذ مهمة محددة و تكتب بأحدى لغات البرمجة.

بعض المفاهيم والتعاريف الاساسية

المبرمج Programmer:

هو صائغ البرامج او هو الشخص الذي يقوم بكتابة البرامج بعد اختياره للغة برمجة معينة لتحقيق هدف اوحل مسالة معينة. حيث تمر عملية البرمجة بالنسبة للمبرمج بعدة مراحل هي :

1) مرحلة فهم وتحليل المشكلة.

2) افتراض حل منطقي ووضع خطوات لذلك.

3) كتابة سلسلة من الأوامر لحل المشكلة.

4) اختبار البرنامج والتأكد من صحة عمله.

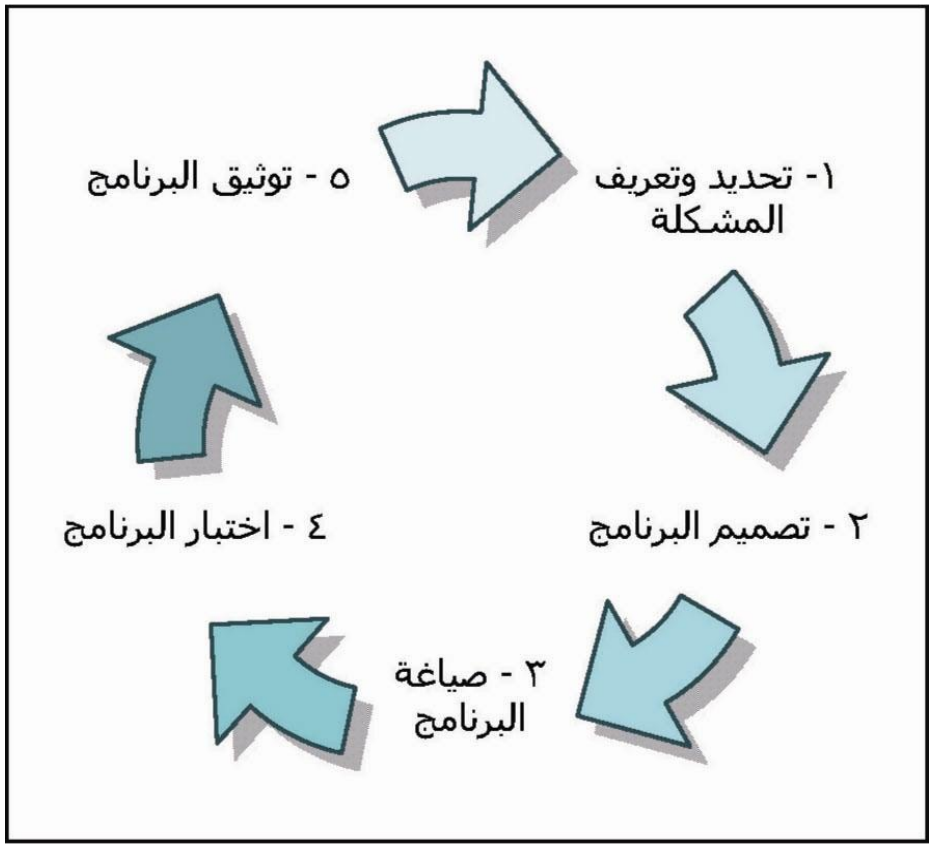
5) تحويل البرنامج إلى صيغة تنفيذية ، تمثل الشكل النهائي الذي يحتوي على الواجهة التي يراها المستخدم وتوثيق البرنامج.

المستخدم User:

هو من يستخدم البرنامج، حيث تظهر له واجهة البرنامج ولن تظهر له الأوامر التي كتبها المبرمج.

خطوات صياغة وتطوير البرامج

Program Development Steps



مخطط يبين خطوات صياغة وتطوير البرامج

خطوات صياغة وتطوير البرامج

Program Development Steps

1. تحديد وتعريف المشكلة Defining the Problem

في هذه الخطوة يقوم المبرمج بتحديد وتعريف المشكلة وتتضمن هذه الخطوة تحديد التالي بالترتيب:

1. الهدف من البرنامج (حساب ارباح، فواتير استهلاك الماء والكهرباء، أو حساب معدل الطالب التراكمي)
2. نوع وحجم المخرجات ووسائل الإخراج (تقارير – فواتير – شيكات – نقود .../ طابعة ــ شاشة ....)
3. نوع وحجم البيانات المدخلة ووسائل الإدخال(اسعارمواد, جملة اومفرد ـ درجات طلاب../ لوحة المفاتيح ـ الكاميرا ـ ...).
4. مستخدمي البرامج والمستفيدين منه.(عام لكل الناس ــ موظف مختص ـ مكاتب اهلية ــ مدارس ـ جامعات.....).

خطوات صياغة وتطوير البرامج

Program Development Steps

2. تصميم البرنامج Design the Program

- يتم هنا تحديد المواصفات والخطوات الدقيقة والمرتبة منطقيا والتي تم فهمها ودراستها في الخطوة الأولى.
- ويتم ذلك بإستخدام عدة طرق منها
- كتابة **الخوارزميات Algorithems** وهي عبارة عن كتابة سلسلة من الخطوات المنطقية المؤدية لحل المشكلة.
- **او استخدام المخططات الانسيابية Flowchart** ويطلق عليها أيضاً خرائط سير العمليات وهي مجموعة من الرموز المتعارف عليها تستخدم لتوضيح الخطوات المنطقية اللازمة لحل مشكلة ما.

خطوات صياغة وتطوير البرامج

Program Development Steps

- **Design the Program : تصميم البرنامج باستخدام الخوارزميات Algorithms**

Example: *Write an algorithm to input tow number and print witch of them greater than other.*

Detailed Algorithm

Step 1: Input X, Y

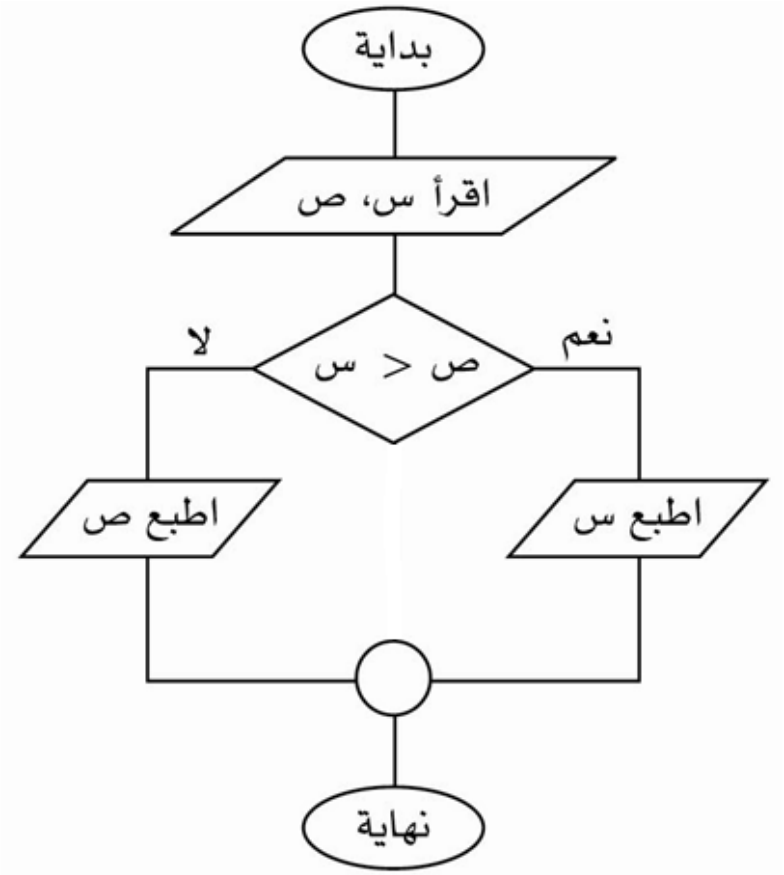
Step 2: if (X>Y) then

Step 3: Print "X greater than Y"
else
Print "Y greater than X"

خطوات صياغة وتطوير البرامج

Program Development Steps

- تصميم البرنامج Design the Program: باستخدام المخطط الانسيابي



أهم الرموز المستخدمة في خرائط التدفق

الاسم	الرمز
بداية – نهاية Start - End	
مدخلات – مخرجات Input - Output	
معالجة Processing	
قرار Decision	

خطوات صياغة وتطوير البرامج

Program Development Steps

3. صياغة البرنامج Coding the Program

- بعد الانتهاء من تصميم البرنامج يتم اختيار إحدى لغات البرمجة المناسبة لصياغة أوامر البرنامج Coding وذلك بالاستعانة بالمخطط الانسيابي Flowchart أوغيرها.
- يجب عند صياغة البرنامج اتباع قواعد صياغة لغة البرمجة المستخدمة حيث ان لكل لغة برمجة قواعد خاصة بها ولا يعمل البرنامج اذا كان هنالك اخطاء املائية او اخطاء في قواعد اللغة **Syntax Errors**.

خطوات صياغة وتطوير البرامج

Program Development Steps

4. اختبار البرنامج وتصحيح الأخطاء Program Debugging and Testing

- يسمى البرنامج بعد صياغتة باحدى لغات البرمجة **البرنامج المصدر** **Source Program** ولا يتم تنفيذه مباشرة على الحاسوب بل يتم ترجمته الى **برنامج مكتوب بلغة الآلة** **Machine Languages** وهو ما يسمى بالبرنامج الهدف **Target Program**.
- تسمى عملية تحويل البرنامج المصدر الى برنامج الهدف **بالترجمة Compilation** ويقوم بها برنامج يسمى **المترجم Compiler**.

خطوات صياغة وتطوير البرامج

Program Development Steps

- خلال عملية الترجمة Compilation قد تظهر اخطاء في صياغة البرنامج المصدر ينبغي على المبرمج تصحيحها.

- هناك ثلاث انواع من الأخطاء:

1. **اخطاء في قواعد اللغة Syntax Errors:** اخطاء املائية او لغوية في كتابة الأوامر وتظهر رسالة بان هناك خطا املائي.
2. **اخطاء اثناء تشغيل البرنامج Run-Time Errors:** تظهر عند تنفيذ البرنامج مثل عدم حجز مساحة كافية للمدخلات او الدخول في دوران بلا نهاية، وتظهر رسالة بنوع الخطاء.

2. **اخطاء منطقية Logical Errors:** لا يكتشفها الحاسوب ولا تظهر رسالة عن الخطأ وتظهر عند تنفيذ البرنامج على عينه من البيانات فنحصل على نتائج خاطئه او غير متوقعة، ويقوم المبرمج بتتبع خطوات البرنامج لمعرفة مصدر الخطأ وتصحيحه وتسمى عملية التتبع ب Tracing.

خطوات صياغة وتطوير البرامج

Program Development Steps

5. توثيق البرنامج Documenting the Program

- في هذه المرحلة تتم كتابة وصف تفصيلي لصياغة البرنامج، ويشمل هذا التوثيق:

1. كتابة أصل المشكلة (او الهدف من البرنامج)
2. وخطوات الحل وخرائط الحل Flowchart
3. وتعليمات التشغيل Help
4. ومتطلبات التشغيل (نظام التشغيل واللغة المطلوبة وسرعة المعالج وحجم الذاكرة...)
5. والمدخلات والمخرجات Inputs Outputs

تصنيف لغات البرمجة

تصنف لغات البرمجة إلى ثلاثة أنواع هى:

1. لغات برمجة ذات مستوى منخفض Low Level Languages
2. لغات برمجة ذات مستوى عال High Level Languages
3. لغات الجيل الرابع Fourth Generation Languages

تصنيف لغات البرمجة

1. **لغات البرمجة ذات المستوى المنخفض Low Level Languages**

- تعتبر لغات البرمجة ذات المستوى المنخفض من أوائل لغات البرمجة ومنها:
  - لغة الآلة Machine Language
  - لغة التجميع Assembly language
- سميت باللغات المنخفضة المستوى نظراً لأن المبرمجين يكتبون أوامر البرنامج بمستوى قريب من مستوى فهم الآلة (الحاسوب)، حيث تستخدم هذه اللغة (0 , 1) في كتابة البرامج
- تكتب الأوامر في لغة الآلة على شكل سلسلة من الأرقام الثنائية (0 , 1) حتى يفهمها جهاز الحاسب الآلي وهي اللغة الوحيدة التي يفهمها الحاسب.
- تُحول جميع اللغات الى لغة الآلة حتى تتمكن معدات الحاسب الآلي من التفاهم معها.
- **مميزاتها :**
  - سرعة التنفيذ لأنها تخاطب وحدة المعالجة مباشرة
- **عيوبها :**

-غير مرنة (صعوبة كتابة وتصحيح برامجها).

-غير عمومية (برامجها تعتمد على نوع الآلة).

تصنيف لغات البرمجة

2. لغات البرمجة ذات المستوى العالي High Level Languages

- بظهور اللغات ذات المستوى العالي أصبحت عملية التخاطب والتعامل مع الحاسب أسهل نسبياً وذلك لأن لغة التعامل مع الحاسب أصبحت قريبة من لغة البشر.
- سميت بهذا الاسم لأنه اصبح بامكان المبرمج كتابة البرامج دون معرفة تفاصيل كيفية قيام الحاسب بهذه العمليات.

بعض مميزات هذه اللغات:

- قريبة من لغة الانسان.
- مرنة(سهولة في كتابة وتعديل وتصحيح البرامج).
- عمومية (عدم الارتباط بآلة معينة).
- توفير الوقت والجهد

عيوبها :

- بطء التنفيذ لاحتياجها لوسيط يقوم بتحويل البرنامج المصدر (Source Code) المكتوب باحدى هذه اللغات الى البرنامج الهدف (Target Code) المكتوب بلغة الآلة.

تصنيف لغات البرمجة

3. لغات الجيل الرابع Fourth Generation Languages

- تسمى هذه اللغات أيضاً **باللغات ذات المستوى العالي جداً Very High Level Languages** حيث إنها لغات سهلة الاستخدام والفهم وقريبة جداً من لغة الإنسان.
- يستطيع المبرمج القيام بكثير من العمليات بسهولة تغنيه عن صياغة Coding صفحات عديدة من أوامر البرنامج. ويهتم المبرمج بماذا يريده من الكومبيوتر دون ان يوجهه بكيفية القيام بذلك.

ومن ضمنها ظهر مايسمى **بلغات البرمجة موجهة الاهداف**
(Object Oriented Programming Language)

- تدعم اسلوب البرمجة المرئية (تصميم الواجهات الرسومية)
- من امثلتها : visual basic , visual c++ , java builder

أنواع لغات البرمجة ذات المستوى العالي

1. لغة البيسك BASIC Language ولغة فيجوال بيسك Visual Basic
2. لغة سي ولغة سي بلس بلس C & C++ Language
3. لغة الجافا Java Language
4. لغة الكوبل COBOL Language
5. لغة الباسكال PASCAL Language
6. لغة اللوجو LOGO Language
7. لغة البايثون Python Language
8. لغات الذكاء الاصطناعي Artificial Intelligence Languages

Computer Programming

First Year Class

Lecture 2

Flowchart

Abdulhameed N. Hameed

Flowchart

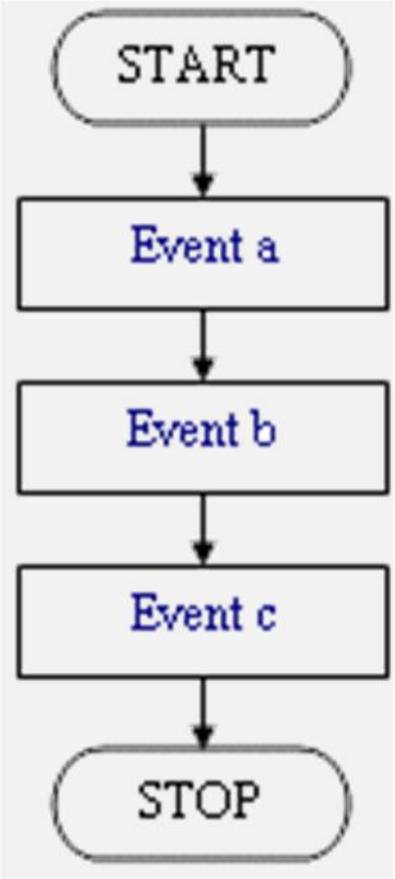
نستخدم مجموعة من الأشكال الرمزية الاصطلاحية المبينة في الجدول التالي:

	الرمز	الحدث الذي يمثله	مثال
1		حدث طرفي Terminal لبيان بدء (Start) أو انتهاء (Stop) خريطة سير العمليات	START STOP
2		عملية حسابية (Process)	LET X+Y
3		إدخال/ إخراج INPUT \ OUTPUT لبيان إدخال/ إخراج معلومات من/ إلى الحاسب	PRINT Z INPUT X, Y
4		اتخاذ قرار Decision	NO X=Y YES
5		اتجاه تدفق (سريان) Flow line	
6		تكرار أو دوران Loop	FOR I= 1 to 10

يمكن تصنيف flowcharts بما يلي:

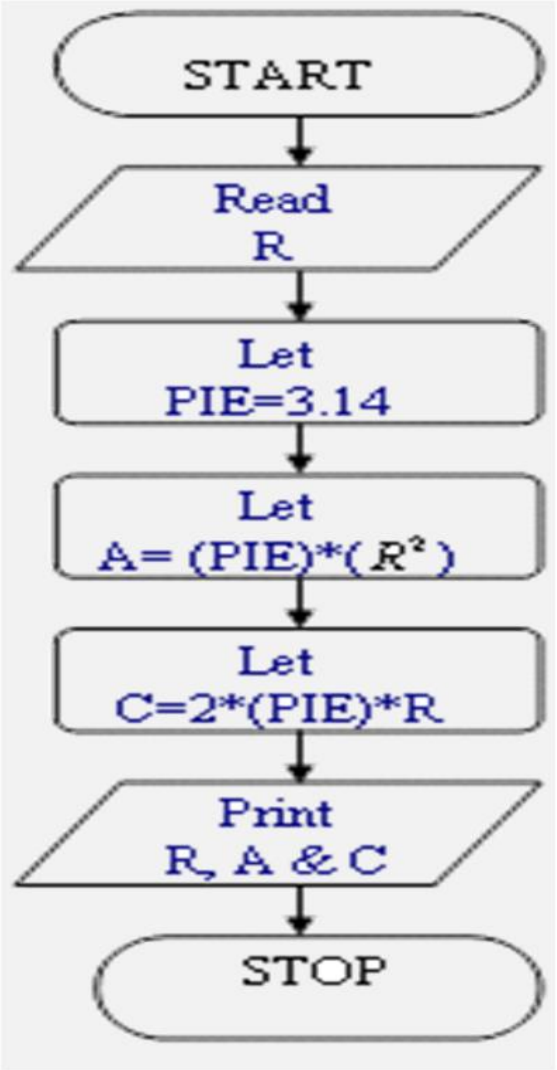
- خرائط التتابع البسيط (Simple Sequential Flowchart)
- خرائط التفرع (Branched Flowchart)
- خرائط الدوران البسيط (Loop Flowchart)
- خرائط الدورانات المتداخلة (Nested)

Simple Sequential Flowchart



- يخلو هذا النوع من التفرعات Branches و الدورانات loops .

Example: Draw a flowchart to find the area and circumference of a circle with a known radius R :



Area= ΠR^2
Circumference=$2\Pi R$
$\Pi \approx 3.14$

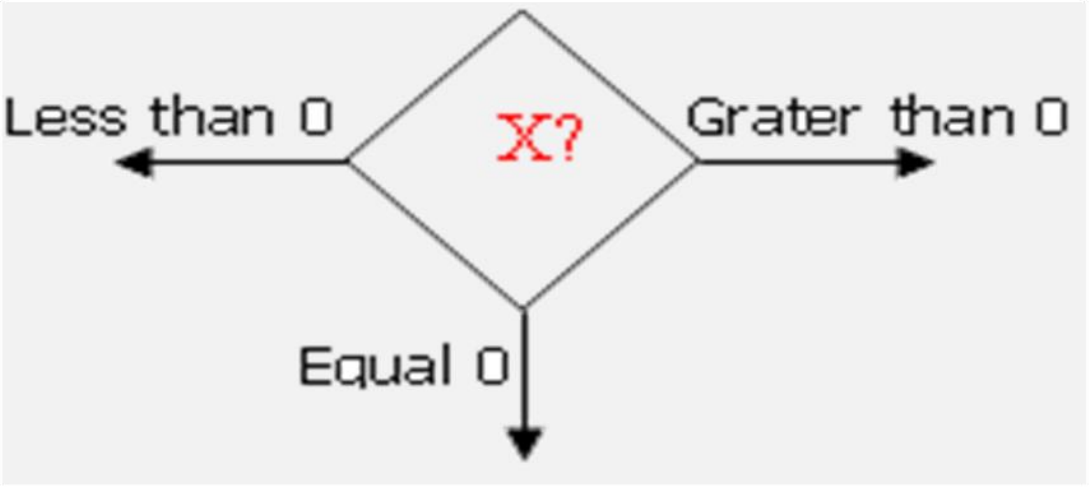
Input: R

Output: R,A,C

الخطوات :

1. ابدأ.
2. اقرأ قيمة R .
3. ضع قيمة PIE =3.14
4. احسب المساحة A من المعادلة A =(PIE)*R*R
5. احسب المحيط C من المعادلة C =2*(PIE)*R
6. اطبع قيم كل من C, A, R.
7. توقف.

Branched Flowchart



ويحدث التفرع في البرامج بسبب الحاجة لاتخاذ قرار أو مفاضلة بين اختيارين أو أكثر، وهناك أسلوبان في تنفيذ القرار .

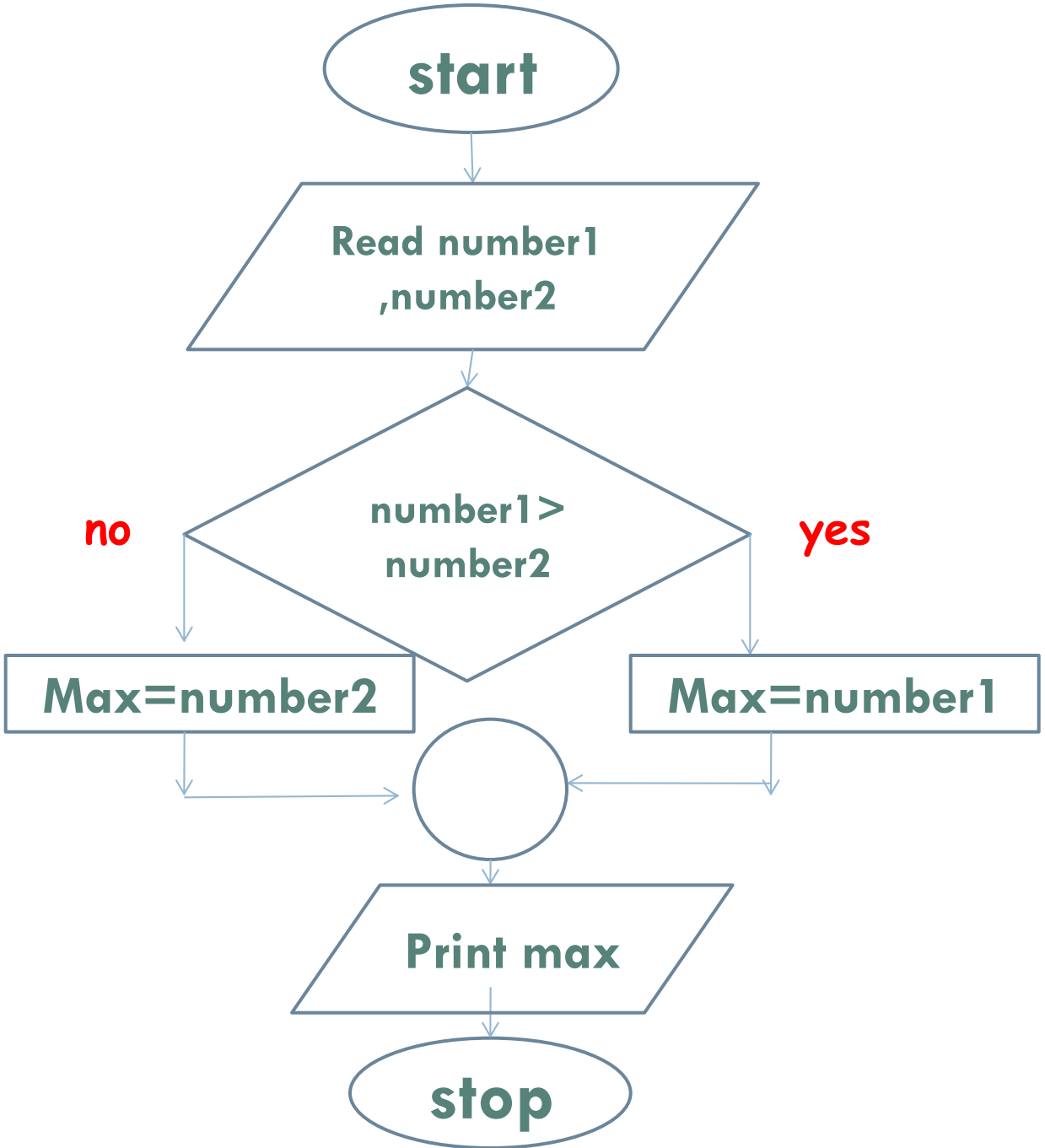
Example: compare between tow numbers and print the max :

Input : number1 ,number2

Output: max

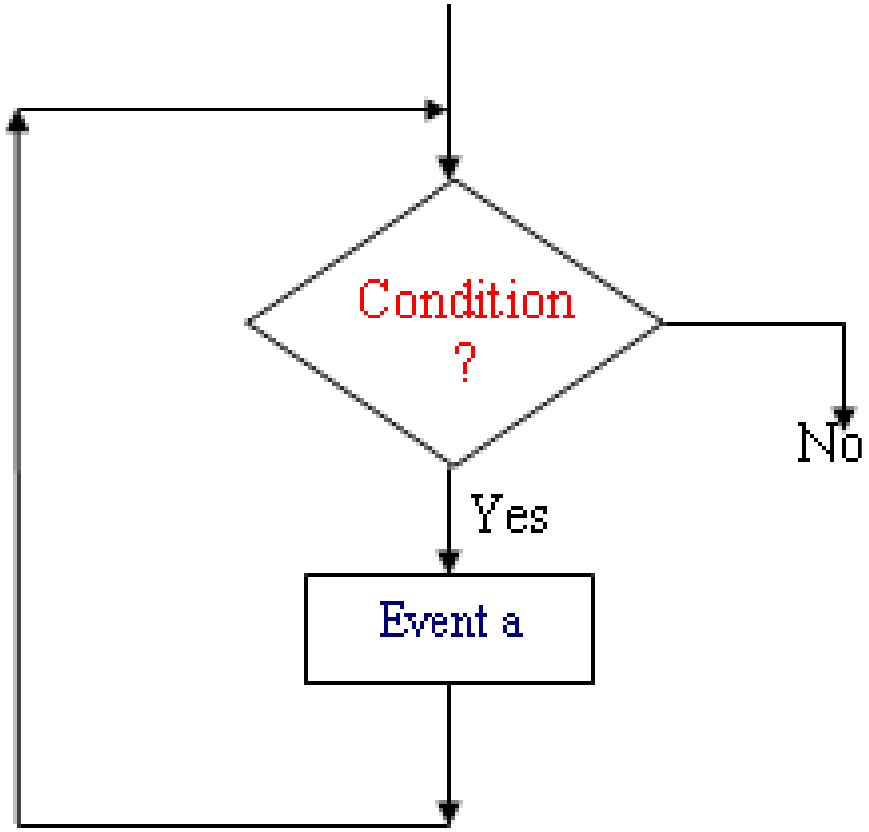
الخطوات :

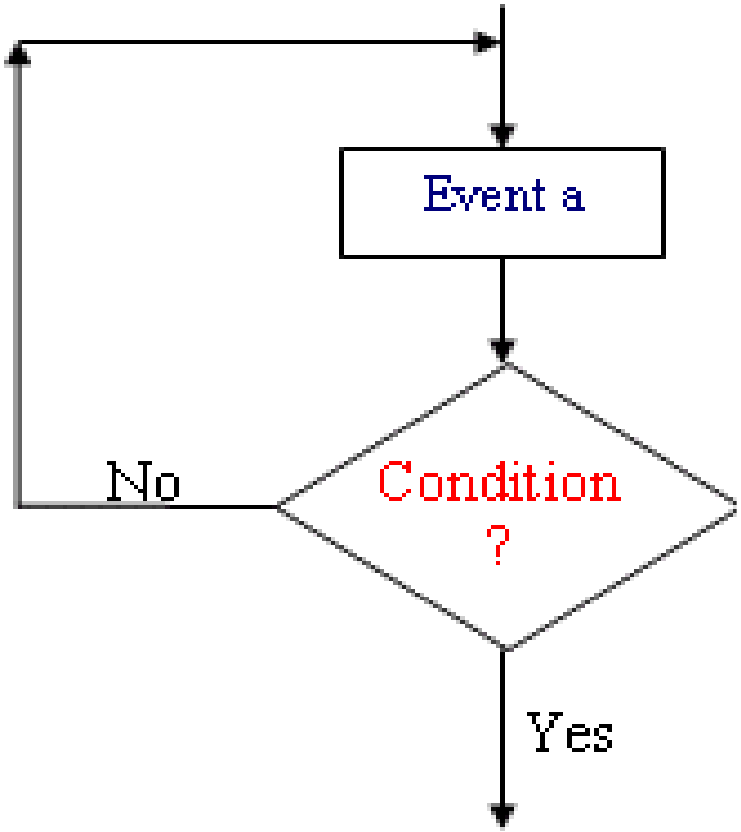
1. ابدا
2. ادخل العدد الاول number1 والعدد الثاني number2
3. اذا كان number1>number2 اجعل max=number1 واذا لا اجعل max=number2
4. اطبع max
5. توقف


start
Read number1
,number2
number1>
number2
no
yes
Max=number2
Max=number1
Print max
stop

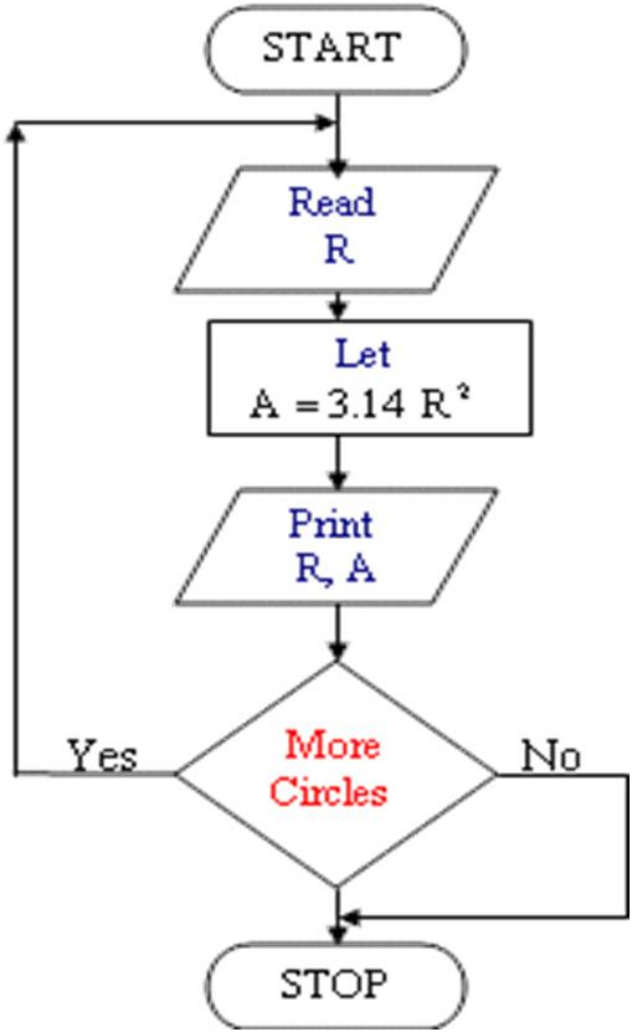
Nested Flowchart

نحتاج إليها لإعادة عملية أو مجموعة من العمليات في البرنامج عددًا محدودًا أو غير محدود من المرات،
ويكون الشكل العام لمثل هذه الخرائط كما يلي:





Example: Draw a flowchart to find the area to group of circles with known semi-diameter:



Input: R

Output: R, A

- خطوات الحل :

1. ابدأ.
2. اقرأ نصف قطر الدائرة R
3. أوجد مساحة الدائرة A
4. اطبع قيم كل من A, R
5. هل هناك مزيد من الدوائر؟
   فإن كان نعم فعد إلى الخطوة(2)
   وإن كان لا فعد إلى الخطوة (6).
6. توقف.

Computer Programming

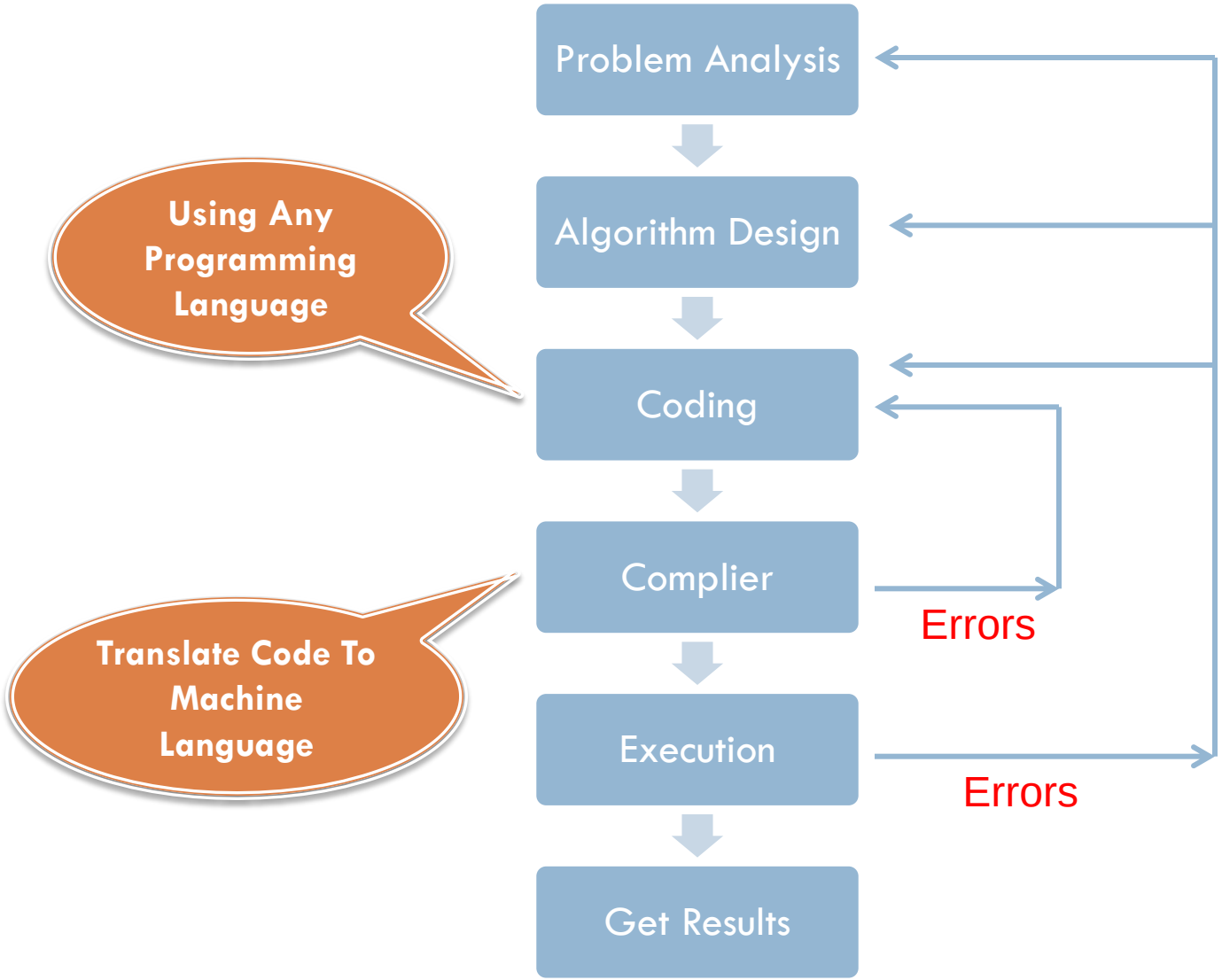
First Year Class

Lecture 3

Structured Programming in C++

Abdulhameed N. Hameed

The Problem Cycle
Analysis–Coding–Execution



The famous **Hello world** program

When learning a new language, the first program people usually write is one that salutes the world :

Here is the Hello world program in C++.

```
#include <iostream.h>
int main()
{

cout << "Hello world!";

return 0;
}
```

A C++ program


```
1. #include <iostream.h>
2. int main()
3.   {
4.       //variable declaration
5.      //read values input from user
6.      //computation
7.      //and print output to user
8. return 0;
9. }
```

1. تعريف البرامج الثانوية (من المكتبة) التي يحتاجها البرنامج لكي يعمل.
2. تعريف اسم البرنامج الرئيسي (الدالة الرئيسية).
3. علامة الابتداء , لكتابة اوامر البرمجة بعدها.
4. تعريف المتغيرات
5. او قراءة قيم من المستخدم
6. اواجراء حسابات معينة
7. او طباعة المخرجات.
8. دليل على انتهاء البرنامج باعادة القيمة صفر.
9. علامة انتهاء اوامر البرمجة (اي حصر اوامر البرمجة بداخل علامة البراكيت)

Complete Program


```
1  // Fig. 2.1: fig02_01.cpp
2  // Text-printing program.
3  #include <iostream.h> // allows program to output data to the screen
4
5  // function main begins program execution
6  int main()
7  {
8     cout << "Welcome to C++!\n"; // display message
9
10    return 0; // indicate that program ended successfully
11
12 } // end function main
```

comments

preprocessor directive

string literal

statement

main function

braces

```
Welcome to C++!
```

output on screen

```
#include <iostream>
int main()
{   Cout << "welcome to c++!\n"
     return 0;     }
```

C++ Program Structure

1. **Comments:** remarks that are ignored by the compiler
2. **Compiler directives:** commands for compiler, which are needed to compile and run program effectively
3. **Main function:** where every program begins
4. **Braces:** mark the beginning and ending code blocks
5. **Statement:** a line of C++ code

1. Comments

- Explain programs, their purpose, authors names and future modification notes to other programmers
  - Improve program readability
- Ignored by compiler
- Single-line comment
  - Begin with (**//**)
- Multi-line comment
  - Start with /*
  - End with */

```
// Fig. 2.1: fig02_01.cpp
// Text-printing program

/* Fig. 2.1: fig02_01.cpp
Text-printing program  */
```

2. Preprocessor Directives

- Instructions to the compiler rather than part of C++ language
- Tells preprocessor stage to include the input/output stream header file <iostream.h>
- Begin with #

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

name of header file

- Forgetting to include the file will result in a compilation error.

3. `main` Function

- Part of every C++ program
  - ONLY one function in a program must be **main**
- Can or can't be "return" a value
  - Returns an integer; once return
- Body is delimited by braces { }
- **`return`** statement
  - The value 0 indicates the program terminated successfully

```
int main()
{

    return 0;

}
```

Rules of building a program in C++

1. Must include `<iostream.h>` for `cout` to work properly
2. C++ is case sensitive.
    1. Make sure you don't capitalize any of the letters in C++ keywords.

        (Main is different that main)
3. Every statement ends with a statement terminator: semicolon(;).
    1. Except for function header, function braces and preprocessor directives.
4. String literals must be enclosed in " ".
5. Main function must return a value to the OS.
6. Every opening brace { must have an enclosing brace }.

Variable declaration

Type-name variable-name

Meaning: variable <variable-name> will be a variable of type <type>

Where type can be:

- `int` `//integer number`
- `float` `//decimal number`
- `double` `//real number`
- `char` `//character`

Example:

```
int a, b, c;
double x;
float sum;
char my-name;
```

Declaration Stage

Input statements

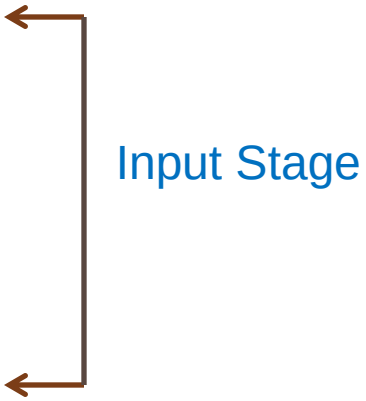
- Standard Input stream object

  `cin >> variable-name;`

  - Connected to the Keyboard
  - Defined in input/output stream header file **`<iostream.h>`**
  - Meaning: read the value of the variable called <variable-name> from the user,

Example:

```
cin >> a;
cin >> b >> c;
cin >> x;
cin >> my-name;
```



Output statements

- Standard output stream object
  - `cout <<`
    - "Connected" to screen
    - Defined in input/output stream header file `<iostream.h>`
  - Example
    - `cout << "Hello";`
      - Inserts the string "Hello" into the standard output then displays to the screen

Output statements

cout << variable-name;

Meaning: print the value of variable <variable-name> to the user.

cout << "any message ";

Meaning: print the message within quotes to the user.

cout << endl;

Meaning: print a new line.

Example:

```
cout << a;
cout << b << c;
cout << "This is my name: " << my-name ;
cout      << endl;
```

Output Stage

Escape Sequences

- Escape characters
  - A character preceded by a backslash **"\"** Indicates “special” character output.
  - Example: **"\n”** → Cursor moves to beginning of next line on the screen.

Escape sequence	Description
`\n`	**Newline. Position the screen cursor to the beginning of the next line.**
`\t`	**Horizontal tab. Move the screen cursor to the next tab stop.**
`\r`	**Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line.**

Modifying First Program - 1


```
1  // Fig. 2.3: fig02_03.cpp
2  // Printing a line of text with multiple statements.
3  #include <iostream.h> // allows program to output data to the screen
4
5  // function main begins program execution
6  int main()
7  {
8     cout << "Welcome ";
9     cout << "to C++!\n";
10
11    return 0; // indicate that program ended successfully
12
13 } // end function main
```

Multiple stream insertion statements produce one line of output

```
Welcome to C++!
```

Modifying First Program - 2


```
1  // Fig. 2.4: fig02_04.cpp
2  // Printing multiple lines of text with a single statement.
3  #include <iostream.h> // allows program to output data to the screen
4
5  // function main begins program execution
6  int main()
7  {
8     cout << "Welcome\nto\n\nC++!\n";
9
10    return 0; // indicate that program ended successfully
11
12 } // end function main
```

Use newline characters to print on multiple lines

```
Welcome
to

C++!
```

Exercise - 1

Write a program that input and display your name, class, section, computer science degree, and your age.

Example (screen output):

Ahmad Ali
First year/Group A
Systems & Control Department
My computer science degree= 70
My age= 20

Computer Programming

First Year Class

Lecture 4

Operators in C++

Abdulhameed N. Hameed

Contents

- Arithmetic operators
- Precedence and associativity of operators
- Increment and decrement operators
- Decision making operators (Equality, relational and logical)
- Conditional operators
- Common errors

Arithmetic Operations

- Operators: Data connectors within expression or equation

- Operators types based on their mission:

1. Arithmetic : addition + , subtraction -, modulo division / , ...etc (x+y)

2. Equality and Relational: equal to == , less than <, grater than >, …etc (x<y)

3. Logical : AND, OR, NOT (x && y)

Operators

- **Operators divided to types based on the number of operands**
  - Unary operators
    - Have only one operand
    - May be prefix or postfix
    - e.g. -- ++ ! (x++ , y--)
  - Binary operators
    - Have two operands
    - Infix
    - e.g. == && + - (x && y , x+y)
  - Ternary operators
    - Have three operands
    - e.g. ? : (**i > j ? max=i : max=j**)

Assignment statement

- Assignment statement takes the form below

```
varName = expression;
```

- Expression is evaluated and its value is assigned to the variable on the left side

- Shorthand notation
  - `varName = varName operator expression;`
  - `varName operator = expression;`

```
c = c + 3;

c += 3;
```

Arithmetic Operators

- Arithmetic Operators: All of them are binary operators

C++ operation	C++ arithmetic operator	Algebraic expression	C++ expression
Addition	+	*f + 7*	f + 7
Subtraction	-	*p - c*	p - c
MUltiplication	*	*bm or b . m*	b * m
Division	/	*x / y or x ÷ y*	x / y
Modulus	%	*r mod s*	r % s

Assignment between objects

- Assignment between objects of the same type is always supported

Assignment operator	Sample expression	Explnation	Assigns
Assume: int c = 3, d = 5, e = 4, f = 6, g = 12;			
+=	**c += 7**	*c = c + 7*	10 to c
-=	**d -= 4**	*d = d – 4*	1 to d
***=**	**e *= 5**	*e = e * 5*	20 to e
/=	**f /= 3**	*f = f / 3*	2 to f
%=	**g %= 9**	*g = g % 9*	3 to g

Arithmetic Operators Precedence

- Operators in parentheses evaluated first
  - Nested/embedded parentheses (Operators in innermost pair first)
- Multiplication, division, modulus applied next
- Addition, subtraction applied last

(Operators applied from left to right)

Operator(s)	Operation(s)	Order of evaluation (precedence)
`()`	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they are evaluated left to right.
`*`, `/`, or `%`	Multiplication Division Modulus	Evaluated second. If there are several, they re evaluated left to right.
`+` or `-`	Addition Subtraction	Evaluated last. If there are several, they are evaluated left to right.

Example

- The statement is written in algebra as

$$z = pr \% q + w / (x - y)$$

How can we write and evaluate the previous statement in C++ ?

```
z = p * r % q + w / (x - y);
```

6 2 3 5 4 1

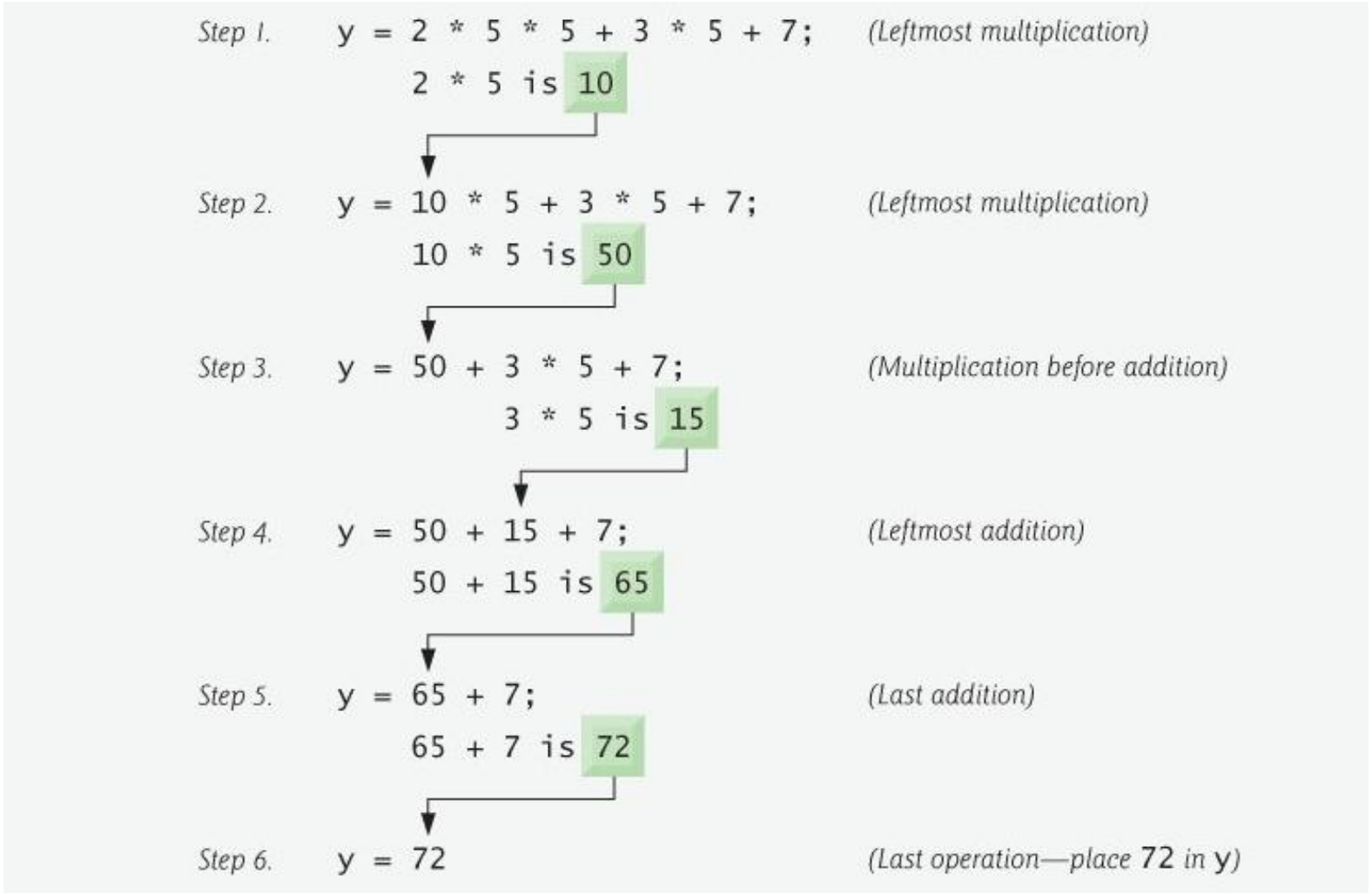
Algebra: $m = \dfrac{a + b + c + d + e}{5}$

C++:
```
m = ( a + b + c + d + e ) / 5;
```

Algebra: $y = mx + b$

C++:
```
y = m * x + b;
```

Example:


Step 1.
y = 2 * 5 * 5 + 3 * 5 + 7;
(Leftmost multiplication)
2 * 5 is 10
Step 2.
y = 10 * 5 + 3 * 5 + 7;
(Leftmost multiplication)
10 * 5 is 50
Step 3.
y = 50 + 3 * 5 + 7;
(Multiplication before addition)
3 * 5 is 15
Step 4.
y = 50 + 15 + 7;
(Leftmost addition)
50 + 15 is 65
Step 5.
y = 65 + 7;
(Last addition)
65 + 7 is 72
Step 6.
y = 72
(Last operation—place 72 in y)

Exercise 1

- State the order of evaluation of the operators in each of the following C++ statements and show the value of x after each statement is performed.
- `x = 7 + 3 * 6 / 2 - 1;`
- `x = 2 % 2 + 2 * 2 - 2 / 2;`
- `x = ( 3 * 9 * ( 3 + ( 9 * 3 / ( 3 ) ) ) );`

Increment and Decrement Operators

- Unary operators
  - Adding 1 to or (subtracting 1 from) variable's value
- Increment operator gives the same result of **`(c=c+1) or (c+=1)`**
- Decrement operator gives the same result of **`(c=c-1) or (c-=1)`**

Operator	Called	Sample expression	Explanation
++	Preincrement	**`++a`**	Increment a by 1, then use the new value of a in the expression in which a resides.
++	Postincrement	**`a++`**	Use the current value of a in the expression in which a resides, then increment a by 1.
__	Pridecrement	**`--b`**	Decrement b by 1, then use the new value of b in the expression in which b resides.
__	Postdecrement	**`b--`**	Use the current value of b in the expression in which b resides, decrement b by 1.

Examples

Example # 1

```
int    x = 10;

cout << "x = " << ++x << endl;
cout << "x = " << x << endl;
```

output # 1

```
x = 11
x = 11
```

Example # 2

```
int    x = 10;

cout << "x = " << x++ << endl;
cout << "x = " << x << endl;
```

output # 2

```
x = 10
x = 11
```

Examples

Example # 1

```
int   x = 10 , y;

y = ++x;

cout << "x = " << x << endl;
cout << "y = " << y << endl;
```

output # 1

```
x = 11
y = 11
```

Example # 2

```
int   x = 10 , y;

y = x++;

cout << "x = " << x << endl;
cout << "y = " << y << endl;
```

output # 2

```
x = 11
y = 10
```

Relational and Equality Operators

- Binary operators
- Used in decision -making statements

Standard algebraic equality operator or relational operator	C++ equality or relational operator	Example of C++ condition	Meaning of C++ condition
Relational operators			
>	`>`	`x > y`	`x` is greater than `y`
<	`<`	`x < y`	`x` is less than `y`
≥	`>=`	`x >= y`	`x` is greater than or equal to `y`
≤	`<=`	`x <= y`	`x` is less than or equal to `y`
Equality operators			
=	`==`	`x == y`	`x` is equal to `y`
≠	`!=`	`x != y`	`x` is not equal to `y`

Relational and Equality Operators (cont.)

- Have the same level of precedence
- Applied from left to right
- Used with conditions
- Return the value `true` or `false`
- Used only with a single condition

Logical Operators

- Used to combine between multiple conditions
- **&&** (logical **AND**)
  - **true** if both conditions are **true**

1st condition — 2nd condition

```
gender == 1  &&  age >= 65
```

- || (logical **OR**)
  - **true** if either of condition is **true**

```
semester_Average >= 90 || final_Exam >= 90
```

Logical Operators (cont.)

- **!** (logical **NOT**, logical negation)
  - Returns **`true`** when its condition is **`false`**, and vice versa

```
!( grade == sentinelValue )
```

Also can be written as

```
grade != sentinelValue
```

Conditional operator (? :)

- Ternary operator requires three operands
  - Condition
  - Value when condition is **`true`**
  - Value when condition is **`false`**
- Syntax

```
Condition  ?   condition's true value : condition's false value
```

Examples

Example # 1

```
grade >= 60 ? cout<<"Passed" : cout<<"Failed";
```

Can be written as

```
cout << (grade >= 60 ? "Passed" : "Failed");
```

Example # 2

```
int i = 1, j = 2, Max;
Max = ( i > j ? i : j );
```

Can be written as .. ?

Additional Operators:

- C++ does not define operators for finding square roots, absolute values, or for raising numbers to a power.

- It provides **predefined program units called functions** that do these and other necessary calculations.

Examples

- $\sqrt{x}$ = sqrt (x)
- $\sqrt{a-b}$ = sqrt (a-b)
- | x | = abs (x)
- x^y = pow (x, y)

Common Compilation Errors

- Attempt to use % with non-integer operands
- Spaces between pair of symbols e.g. (= =, ! =, …etc)
- Reversing order of pair of symbols e.g. =!
- Confusing between equality (==) and assignment operator (=)

Exercise - 2

What is the output of the following program?

```
 1    #include <iostream>
 2
 3
 4
 5
 6    int main()
 7    {
 8      int x;   int y;  int z;
 9
10      x = 30;  y = 2;  z = 0;
11
12      cout << (++++x && z ) << endl;
13      cout << x * y + 9 / 3 << endl;
14      cout << x << y << z++ << endl;
15
16      return 0;
17
18    } // end main
```

Exercise - 3

What is wrong with the following program?

```
1       int main()
2       {
3           int a,b,c,sum;
4           sum = a + b + c ;
5           return 0;
6       }
```

Computer Programming

First Year Class

Lecture 5

Selection Statements

2024-2025

Abdulhameed N. Hameed

Content

- Control structures
- Types of selection statements
- `if` single-selection statement
- `if..else` double-selection statement
- Nested `if..else` statements
- `switch` multiple-selection statement
- Common errors

Control Structure (Logic Structure)

- Used to design data flow in modules and program as a whole

- Basic structures

1. Sequential structure
   - Processes one instruction after another

2. Selection structures
   - Decision structure
     - Make choices by using relational or logical operators(Ex. If or if..else)
   - Case structure
     - Enable to pick up one of a set of tasks (Ex. switch)

3. Loop structure
   - Enable to repeat tasks (Ex. for)

`if` Single-Selection Statement

- Begin with `if` followed by condition; then action or group of actions are listed
- Syntax
  - Single action

```
if (condition)
   action;
```

  - Multiple actions

```
if (condition)
{
   action1;
   action2;
   ..
   actionN;
}
```

- If condition is true, action is performed
  - Otherwise, action is ignored

```
if(grade >= 60)

Cout << "Passed";
```

if..else Double-Selection Statement

- Begin with `if` followed by condition; then action or group of actions are listed
- End with `else` then action or group of actions are listed

```
if (condition)
    action1;
else
    action2;
```

```
if(grade >= 60)
    cout << "Passed";
else
    cout << "failed ";
```

- If condition is true, action that followed by `if` is performed

  Otherwise, action that followed by `else` is performed

Nested `if..else` Statements

- **Nested If : means to write an *if* statement within another *if* statement.**
- One inside another, test for multiple cases
- Once condition met, other statements skipped

```
if (condition1)
   action1;
else
   if (condition2)
      action2;
   else
      if (condition3)
         action3;
      ...

         else
            actionN;
```

```
if (condition1)
{
   if (condition2)
      action1;
  else
     {
        if (condition3)
           action2;
        else
           action3;
     }
}
else
    action4;
```

Example

- For example, the following code will print:

```
if ( grade >= 90 )
    cout << "Excellent";
  else if ( grade >= 80 )
       cout << "very good";
     else if ( grade >= 70 )
          cout << "good";
        else if ( grade >= 60 )
             cout << "median";
           else
             cout << "Fail";
```

- `Excellent` //for grades greater than or equal to 90.
- `very good` //for grades greater than or equal to 80.
- `good` //for grades greater than or equal to 70.
- `median` //for grades greater than or equal to 60.
- `Fail` // for all other grades.

***Home work*: Try to add "Pass" for grades greater than 50**

Dangling-`else` Problem

- Each `else` associated with immediately preceding `if`
- There is exception when placing braces { }

```
int x = 10,  y = 2;
if ( x > 5)
   if( y > 5)
      cout << "x and y are > 5"<< endl;
   else
      cout<<"x is <=5";
```

Have logic error

If the condition is not true

If the condition is true

Correctness

```
int x = 10,  y = 2;
if ( x > 5)
   {
    if( y > 5)
       cout << "x and y are > 5"<< endl;
    }
else
cout<<"x is <=5";

cout<<"x is >=5";
```

Combining more than one condition

- To combine more than one condition we use the logical operators.

Operator	Means	Description
&&	And	The Expression Value Is true If and Only IF both Conditions are true
||	OR	The Expression Value Is true If one Condition Is True

Example : check whether num1 is between 0 and 100

```
If ( (num1 >= 0) && (num1 <=100) )
    cout <<"The Number Is between 0 and 100" ;
else
    cout <<" The Number Is Larger Than 100";
```

`switch` Multiple-selection Statement

- ***The switch*** statement: that allows us to make a decision from a number of choices (Enable to pick up one of a set of tasks)
- Perform actions based on possible values of variable or expression
- Value of expression compared to `case` labels then execute action for that `case`
- No matching, the execution go to the optional `default` statement
- `break` causes immediate exit from `switch` statement

```
switch (expression)
{
case value1:
              action1;
              break;
case value2:
              action2;
              break;
...

case valueN:
              actionN;
              break;
default:
              action;
}
```

switch Multiple-selection Statement

```
Switch (Expression )
{
case constant 1 :
        Action statements;
        break ;
case constant 2 :
        Action statements;
        break ;
case constant 3 :
        Action statements;
        break ;
default :
        Action statements;
}
```

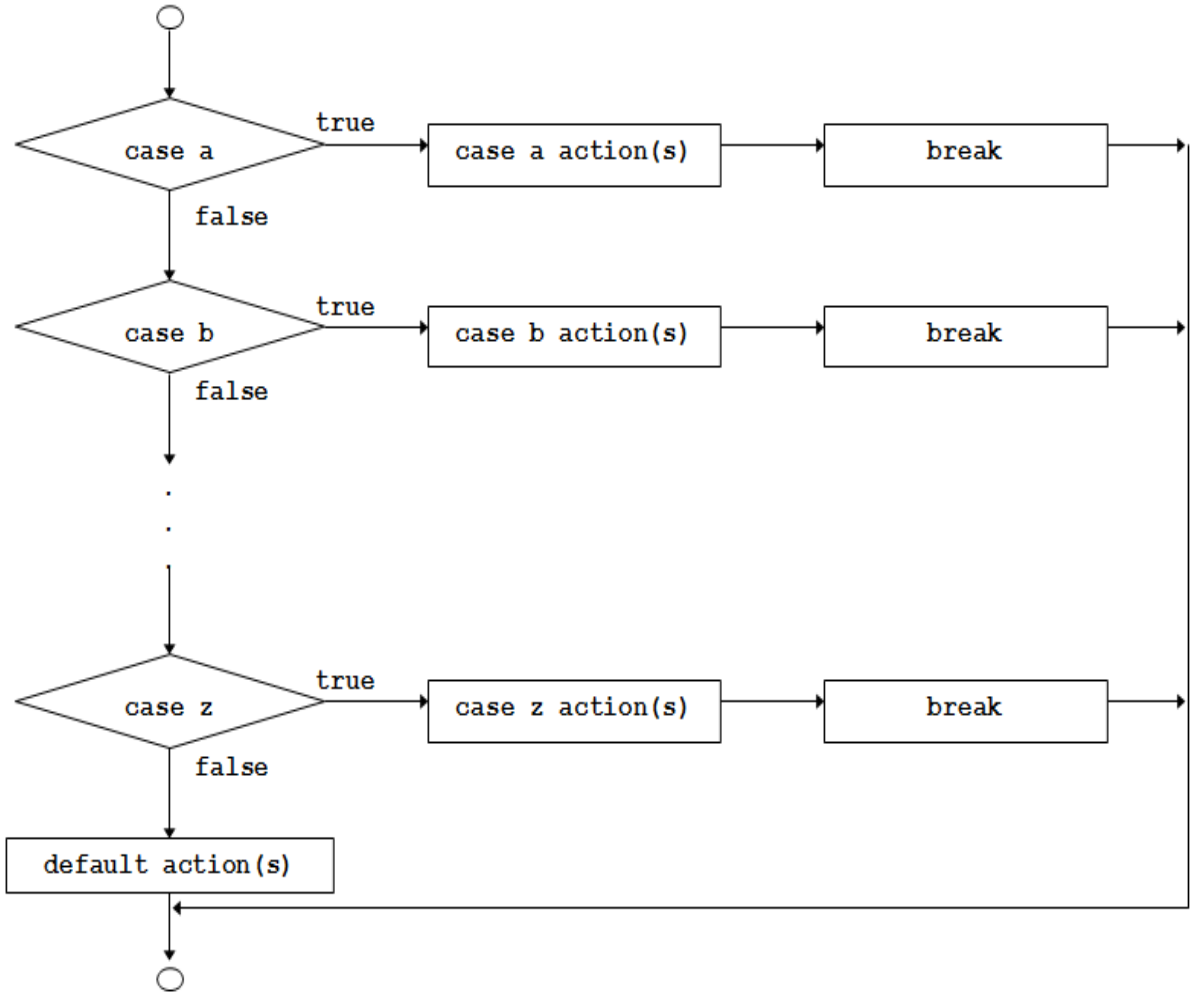
Expression It could be an integer constant like 1, 2 or 3, or an expression that evaluates to an integer .

Constant : is a specific value.

```
Switch (10.0);

float i ;
Switch (i);

switch ( i + j * k )

switch ( 23 + 45 % 4 * i)
```

```
Case  >= 50;
Case   a+ b;
```

switch Multiple-selection Statement




```
switch (grade)
{
case 90 :
cout <<"You Got A \n";   break;

case 80 :
cout <<"You Got B \n";   break;

case 70 :
cout <<"You Got C \n";   break;

case 60 :
cout <<"You Got D \n";   break;

default :
cout <<" Sorry , You Got F \n";
}
```

Common Compilation Errors

- Placing semicolon (;) after `if` condition or `else` keyword
- Omitting spaces between `case` keyword and value(`case_0:`)
- Specifying expression including variables (a + b) in `case` label of `switch` statement
- Providing identical `case` labels
- Forgetting a `break` statement when one is needed in a `switch`

Class work: Determine the output for each of the following when **x = 9** and **y = 11**, and when **x** and **y = 11.**

a)
```
if ( x < 10 )
    if ( y > 10 )
    cout << "*****" << endl;
else
cout << "#####" << endl;
cout << "$$$$$" << endl;
```

ANS:

b)
```
if ( x < 10 )
  {
    if ( y > 10 )
     cout << "*****" << endl;
      }
      else
      {
       cout << "#####" << endl;
       cout << "$$$$$" << endl;
      }
```

ANS:

Exercise – Home work

1) Write a program that asks for an integer and reports whether the number is odd or even. Use `if .. else` statement.

2) Write another version of program using `switch` statement.

Systems and Control Eng. Dept.

Computer Programming

First Year Class

Lecture 6

Repetition Statements

(while & do-while)

2024-2025

Abdulhameed N. Hameed

Content

- Types of repetition statements
- `while` repetition statement
- Counter-controlled repetition
- `do..while` repetition statement

Loop : Repetition Statements

- Some times we need to repeat a specific course of actions either a specified number of times or until a particular condition is being satisfied.
- For example :
  - To calculate the Average grade for 10 students.
  - To calculate the bonus for 10 employees.
  - To sum the input numbers from the user as long as he/she enters positive numbers.
- **A loop statement:-** allows us to execute a statement or group of statements multiple times.
- There are three methods of which we can repeat a part of a program. They are:

1) **`while`** statement
2) **`do..while`** statement
3) **`for`** statement

while Repetition Statements

- Actions repeated while condition remains true
- Syntax

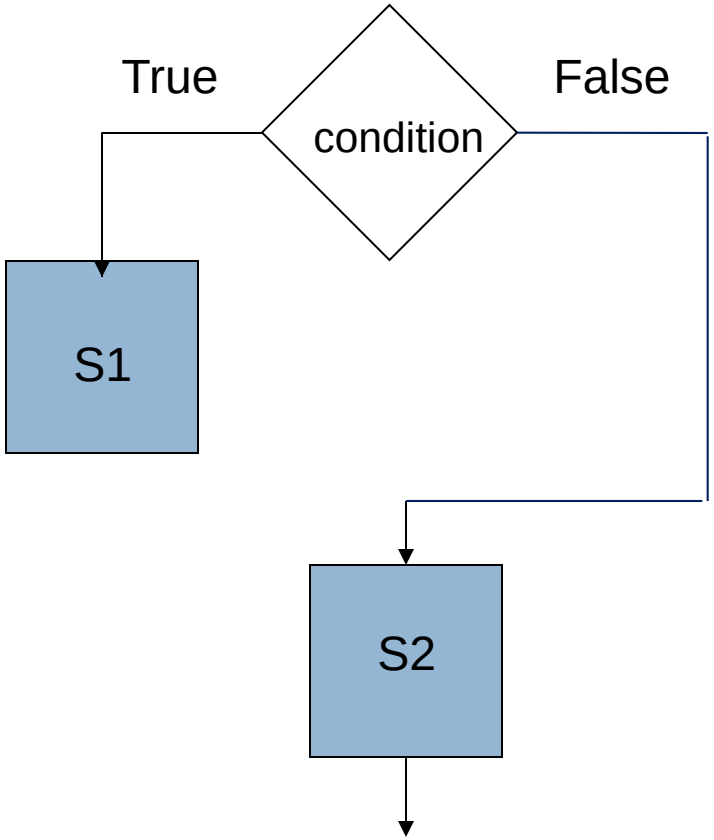
```
while (condition)
{
   action1;
   action2;
   .
   .
   actionN;
}
```

- One of the actions should causes condition to becomes false
- Example

```
int product = 3;
while ( product <= 30 )
{
      product *= 3;
}
```

`while` Repetition Statements (cont.)

```
while (condition)
 {
   S1;
 }

  S2;
```



Counter-Controlled Repetition

- Uses a variable to control number of loops' iterations
- Also known as definite repetition
  - Number of repetitions known beforehand
- Requires
  1. Name of loop control variable
  2. Initial value of loop control variable
  3. Condition to test of reaching final value
  4. Update loop control variable

```
control var Name;
Initialize control var ;

while (condition)
{
   action1;
   action2;
   .
   .
   actionN;
   update control var;
}
```

Counter-Controlled Repetition(cont.)

- **Example**: Write a program that calculates and prints out the Average grade for 6 students.

```
int counter = 1;

int grade=0 , sum=0;

while (counter <=6)
{
cout <<"Enter grade for student no. " << counter <<"\n";
cin >>grade;
sum += grade;
counter ++;
}
cout <<"Average Grade is " << sum/counter <<"\n";
```

1-Name 2-Initial for counter

3- Condition to test the counter

4- Update control variable

Example:
Write C++ program to type numbers from 1 to 10 using ***while*** statement.

```
1  // Fig. 5.1: fig05_01.cpp
2  // Counter-controlled repetition.
3  #include <iostream>
4  using std::cout;
5
6
7  int main()
8  {
9     int counter = 1; // declare and initialize control variable
10
11    while ( counter <= 10 ) // loop-continuation condition
12    {
13       cout << counter << " ";
14       counter++; // increment control variable by 1
15    } // end while
16
17    cout << endl; // output a newline
18    return 0; // successful termination
19 } // end main
```

```
1 2 3 4 5 6 7 8 9 10
```

Control-variable name is **counter** with variable initial value **1**

Condition tests for **counter**'s final value

Increment the value in **counter**

do..while Repetition Statement

- Unlike **for** and **while** loops (which test the loop condition at the **top** of the loop) the **do...while** loop checks its condition at the **bottom** of the loop.

- A **do...while** loop is similar to a **while** loop, except that a **do...while** loop is guaranteed to execute at least one time.

Syntax

```
do
{
    action1;
    action2;
    .
    .
    actionN;
} while (condition)
```

```
do
{
    this ;
    and this ;
    and this ;
    and this ;
} while ( this condition is true ) ;
```

```
while ( this condition is true )
{
    this ;
    and this ;
    and this ;
    and this ;
}
```

do..while Repetition Statement (cont.)

Consider the following two loops:

a/
```
i=11;
While (i<=10)
{
cout << i ;

i=i+5;

}

cout<< i <<endl;
```

Output:

b/
```
i=11;
do
{
cout<< i ;

i=i+5;

} While(i<=10);

cout << i << endl;
```

Output:

do..while Repetition Statement (cont.)

- **Example**: Write a program that calculates and prints out the Average grade for 6 students.

Using do…while

```
int counter=1;
float grade=0 , sum=0;
do
{
cout <<"Enter grade for student no "
    << counter <<"\n";
cin >>grade;
sum += grade;
counter ++;
} while (counter <=6) ;
counter --;
cout <<"Average Grade is " <<
    sum/counter <<"\n";
```

Using while

```
int counter = 1;
float grade=0 , sum=0;
while (counter <=6)
{
cout <<"Enter grade for student no "
<< counter <<"\n";
cin >>grade;
sum += grade;
counter ++;
}
counter --;
cout <<"Average Grade is " <<
sum/counter <<"\n";
```

Example:
Write C++ program to type numbers from 1 to 10 using (do…while) statement.

```
1  // Fig. 5.7: fig05_07.cpp
2  // do...while repetition statement.
3  #include <iostream>
4
5
6
7  int main()
8  {
9     int counter = 1; // initialize counter
10
11    do
12    {
13       cout << counter << " "; // display counter
14       counter++; // increment counter
15    } while ( counter <= 10 ); // end do...while
16
17    cout << endl; // output a newline
18    return 0; // indicate successful termination
19 } // end main
```

```
1 2 3 4 5 6 7 8 9 10
```

Declare and initialize control variable **counter**

do…while loop displays **counter**'s value before testing for **counter**'s final value

Hom work:

1. Write C++ program to type numbers from 20 to 30 using while statement.

2. Write C++ program to find the summation of odd numbers from 100 to 200 using do-while statement.

Systems and Control Eng. Dept.

Computer Programming

First Year Class

Lecture 7

Repetition Statements

(for)

2024-2025

Abdulhameed N. Hameed

for Repetition Statement

- Provide counter-controlled repetition details in a single statement

- Syntax

```
for (initialization; condition; increment)
 {
    action(s);
 }
-----------------------------------------------------
for (initialization; loop Continuation Condition; update)
           action1;

-----------------------------------------------------

for (initialization; loop Continuation Condition; update)
{
         action1; action2; … actionN;
}
```

- `for loop` repeats actions until condition becomes false

for Repetition Statement

- For Loop is probably the most popular looping instruction.

```
for ( initialise  counter ; test  counter ; increment  counter )
{
      do this ;
      and this ;
      and this ;
}
```

- **for allows us to specify three things about a loop in a single line:**

  (a) Setting a loop counter to an initial value.

  (b) testing the loop counter to detect whether its value reached the number of repetitions desired.

  (c) increasing the value of loop counter each time the program segment within the loop has been executed.

for Repetition Statement

```
for ( init; condition; increment )
   {
      action(s);
   }
```

1/ The init. step is executed first, and does not repeat.

2/ Next, the condition is evaluated, and the body of the loop is executed if the condition is true.

3/In the next step, the increment statement updates the loop control variable.

4/Then, the loop's body repeats itself, only stopping when the condition becomes false.

remember that the **semicolons** are mandatory.

```
Ex.: for (int x = 1; x < 10; x++)
        {
            // some code
        }
```

for Repetition Statement

Example : Write a program that calculates and prints out the Average grade for 6 students using *for* statement .

```
int grade=0, sum=0;
for (int counter =1 ; counter <=6 ; counter ++)
{
cout <<"Enter Grade \n";
cin>>grade;
sum += grade;
}
cout <<"The Average grades is “ << sum/6 <<"\n";
```

```
int counter = 1;
int grade=0 , sum=0;
while (counter <=6)
{
cout <<"Enter grade for student \n“ ;
cin >>grade;
sum += grade;
counter ++;
}
cout <<"Average Grade is "<< sum/6
<<"\n";
```

Example:

Write a C++ program to print numbers from 1 to 10 using ***for*** statement.

```
1  // Fig. 5.2: fig05_02.cpp
2  // Counter-controlled repetition with the for statement.
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6
7  int main()
8  {
9     // for statement header includes initialization,
10    // loop-continuation condition and increment.
11    for ( int counter = 1; counter <= 10; counter++ )
12       cout << counter << " ";
13
14    cout << endl; // output a newline
15    return 0; // indicate successful termination
16 } // end main
```

```
1 2 3 4 5 6 7 8 9 10
```

صيغة التحديث لمتغير السيطرة (لكي نصل إلى شرط التوقف)

Increment for **counter**

Condition tests for **counter**'s final value

Control-variable name is **counter** with initial value **1**

اسم متغير السيطرة وهو عبارة عن عداد مع قيمة ابتدائية=1

شرط الاستمرارية (أو شرط التوقف) للدارة

for Repetition Statement (cont.)

- When loop counter is declared in *initialization* expression, it can ONLY be used inside for statement (local variable)

- *initialization* and *update* expressions can be comma-separated lists of expressions

```
for (init.; Condition; update)
          action1;

for(int i=0, j=0; i<4 && j<8; i++,j++)

    cout << "*";
```

Examples Using `for` Statement

- Write a program that prints out numbers from 0 to 10 in descending order

```
for(int i = 10; i >= 0; i-- )
        cout << i << "\n";
```

- Write a program that prints out numbers from 7 to 77 in steps of 7

```
for(int i = 7; i <= 77; i += 7 )
        cout << i << "\n";
```

- Write a program that prints out the sequence: 99, 88, 77, 66, 55, 44, 33, 22, 11, 0

```
for(int i = 99; i >= 0; i -= 11 )
        cout << i << "\n";
```

// If we need the increments more than 1 we should use counter

- **Example** : Write a program that calculates the Factorial for any given positive number.

Ex : **Factorial (5) = 5 * 4 * 3 * 2 * 1**

```
int number, factorial=1;
cout <<"Enter a positive number\n";
cin >> number;
if (number < 0 )
cout <<" Enter Positive Numbers only\n";
else
for (int i= 1 ; i<=number ; i++)
      factorial = factorial * i;
cout <<" Factorila  = “ << factorial <<"\n";
```

Common Errors

- Compilation errors
  - Using commas instead of the two required semicolons in a `for` header

- Logic errors
  - Not initializing counters and totals
  - Placing semicolon immediately after `for` header

```
for ( init; condition; increment )
    {
       action(s);
    }
```