CA212
OO Design & Implementation

Lecturer

Brian Stone

Brian.Stone@comppapp.dcu.ie

Office: L244

Course Structure

- Follows structure of text book "*C++ HOW TO PROGRAMME*" by Deitel & Deitel.
- Uses Microsoft's Visual C++ compiler.
- Based on lectures, **programming workshops** and UML notation.
- Handouts and Slides augment the textbook
- Assessments: lab exams; project, written exam.

Objectives

- To be able to read and write programs in C
- To be able to read and write programs in C++
- To learn structured programming techniques in C and C++.
- To understand and apply basic UML (and SSADM) diagramming techniques.

Syllabus (provisionally)

- Variable Types. Simple C/C++ instructions, expressions and Looping structures. Functions.
- Pointers. Dynamic memory and the heap: Memory allocation. Parameter passing. File I/O
- Complex Data structures. C++ Extensions - reference passing and the const keyword.

- Name Mangling and function and operator overloading. Inline Functions.
- Introduction to C++ classes. Designing a class. Constructors and Destructors. Derived classes.
- Copy constructor and assignment operator. Virtual Functions.
- Multiple Inheritance. Exception Handling.
- Templates
- UML Notation throughout

W1: What is a Program?

- Programs are like recipes or a series of instructions which must be followed precisely
- Programs may be constructed of sub programs
  - Chocolate cake recipe does not define how to make chocolate!
  - Chocolate recipe does not define how to grow cocoa beans!

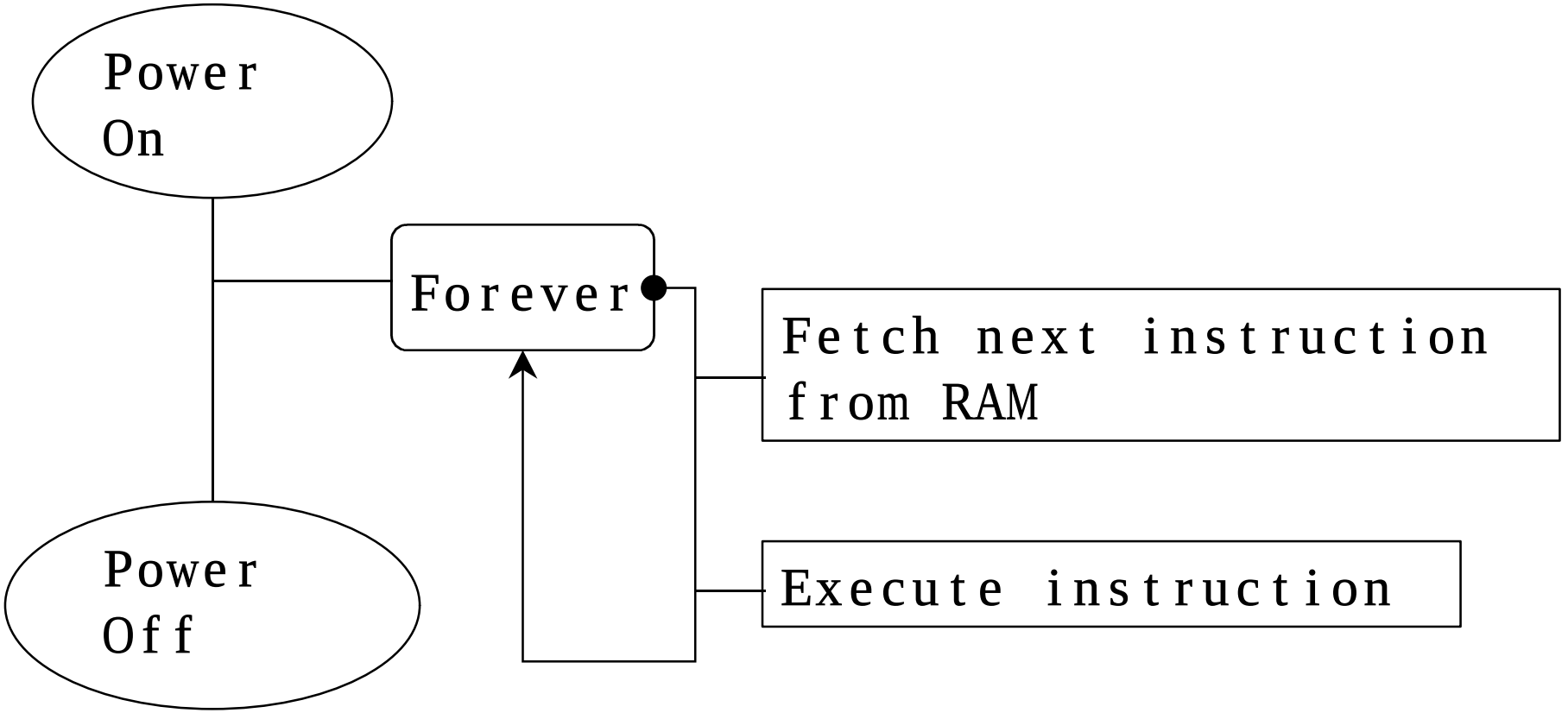
What is an Operating System?

- Operating system responsible for giving your programs to the hardware (CPU).
- OS may be capable of controlling several programs, passing them one-at-a-time to the CPU
- It is a piece of software, NT and Windows 95, 98 developed by Microsoft, Solaris developed by SUN

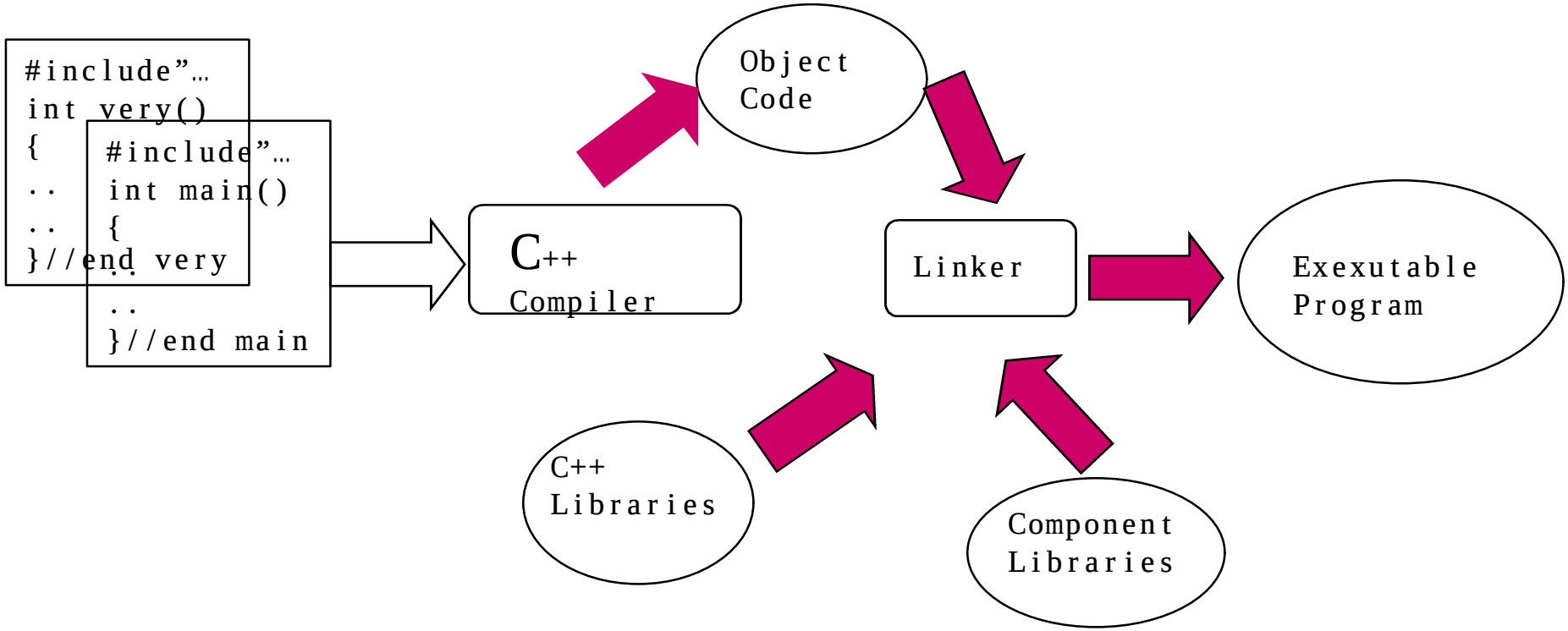
How do programs run?

- User selects which program to run
- OS loads executable into RAM from disk
- OS reads the program file in RAM and passes instructions to the H\W
- H\W executes instruction, OS fetches next instruction
- OS may decide to schedule each program for a little while, in a round-robin style

CPU Execution Cycle



Making C & C++ Executables



Good Programs

- Accuracy
  - Does exactly what it says in the spec. !
- Reliability
  - Keeps on doing it
- Robustness
  - Handles exceptional data sensibly

Good Programs (cont.)

- Efficiency
  - Acceptable performance (speed\code-size etc..)
- Usability
  - Easy to use (other programmers may be users !)
- Maintainability
  - Source code facilitates easy modification and extensibility
- Portability
  - May move code to another operating environment.

Lets look at C First

- Evolved from B and CCPL languages
- Developed by Ritchie (the R in K&R) in 1972
- KnR C was developed in late 70's and became a de-facto standard.
- There is now an ANSI standard for C and C++
- Bjarne Stroustrup developed C++ in the early 80's

Hello …..

```
//comment: first C program

#include<stdio.h>

void main()
{
   printf("First program output\n");
   return;
}
```

```
//comment: first C++ program

#include<iostream.h>

void main()
{
   cout<<"First program output\n";
   return 0;
}
```

Does this look OO ?

Adding two Integers in C

```
#include <stdio.h>
main()
{
   int integer1, integer2, sum;      /* declaration */
   printf("Enter first integer\n");  /* prompt */
   scanf("%d", &integer1);           /* read an integer */
   printf("Enter second integer\n"); /* prompt */
   scanf("%d", &integer2);           /* read an integer */
   sum = integer1 + integer2;        /* assignment of sum */
   printf("Sum is %d\n", sum);       /* print sum */
   return 0;  /* indicate that program ended successfully */
}
```

Adding two integers in C++

```
#include <iostream.h>
main()
{
        int integer1, integer2, sum;      // declaration
        cout << "Enter first integer\n";  // prompt
        cin >> integer1;                  // read an integer
        cout << "Enter second integer\n"; // prompt
        cin >> integer2;                  // read an integer
        sum = integer1 + integer2;        // assignment of sum
        cout << "Sum is " << sum << endl; // print sum
  return 0;   // indicate that program ended successfully
}
```

Does this look OO ?

C++ Vs C

- Some of the new features of C++ have nothing whatsoever to do with Object Oriented programming.
  - Parameter passing, debugging, exceptions.
- Don’t feel bad because you’re not using a Multiple Inheritance Class Hierarchy to implement “Hello World”.

C++ Vs C (cont.)

- Some of the new features supplant and make obsolete some of the functionality of C.
- That is, they do it better and more simply, and with a more elegant syntax.
- One example is in keyboard/screen Input and Output
  - Like the two examples just seen! We will continue to borrow nice bits from C++!!!

Programming in C

- Many of the basic types that exist in Java exist here also.
- There is no such thing as Class in C, there is in C++.
- Types may be declared ANYWHERE in a program and used so long as they are in scope.
- Functions may be declared and called at will.

Some Types

- **char** 8 bits usually used to store ASCII codes, but may be used for small integers.
- **int** size dictated by word length of computer, 32 bits on most modern computer, may be set by compiler. Problems with porting *int*.
- **float** Up to 7 decimal places of floating-point accuracy in 32 bits. Not recommended.

Some Types (cont.)

- **double** Up to 17 decimal places of floating point accuracy in 64 bits. Typically 52 bit mantissa and 12 bit exponent.
- **void** Has no value. For example the main program function is often declared as type void, as it returns no value.
- Qualifier **unsigned** may be used.
- **short** and **long** and **long double** (128 bit f.p.)

Type Conversions

- You should never assign an instance of one type to an instance of another.
- If you are sure of what you are doing, use a *cast.*

```
int x;
long y;
y=x; /* compiler warning! */
y=(long)x; // this is Ok
```

Exercise / Tutorial

- Code and run the examples given so far
- Annotate your code with pencil, find the following and mark them
  - identifiers
  - declarations
  - assignments
  - statements
  - expressions
  - compound statement
  - compound statement start delimiter
  - compound statement end delimiter

W2: Taking for Granted...

- You know about
  - if\else and case\switch statements
  - looping constructs for, whole, do\while
  - functions with parameters

Introducing Pseudocode & Flowcharts

Learn to design a simple algorithm using an *if / else* selection structure with pseudocode and debug the resulting code

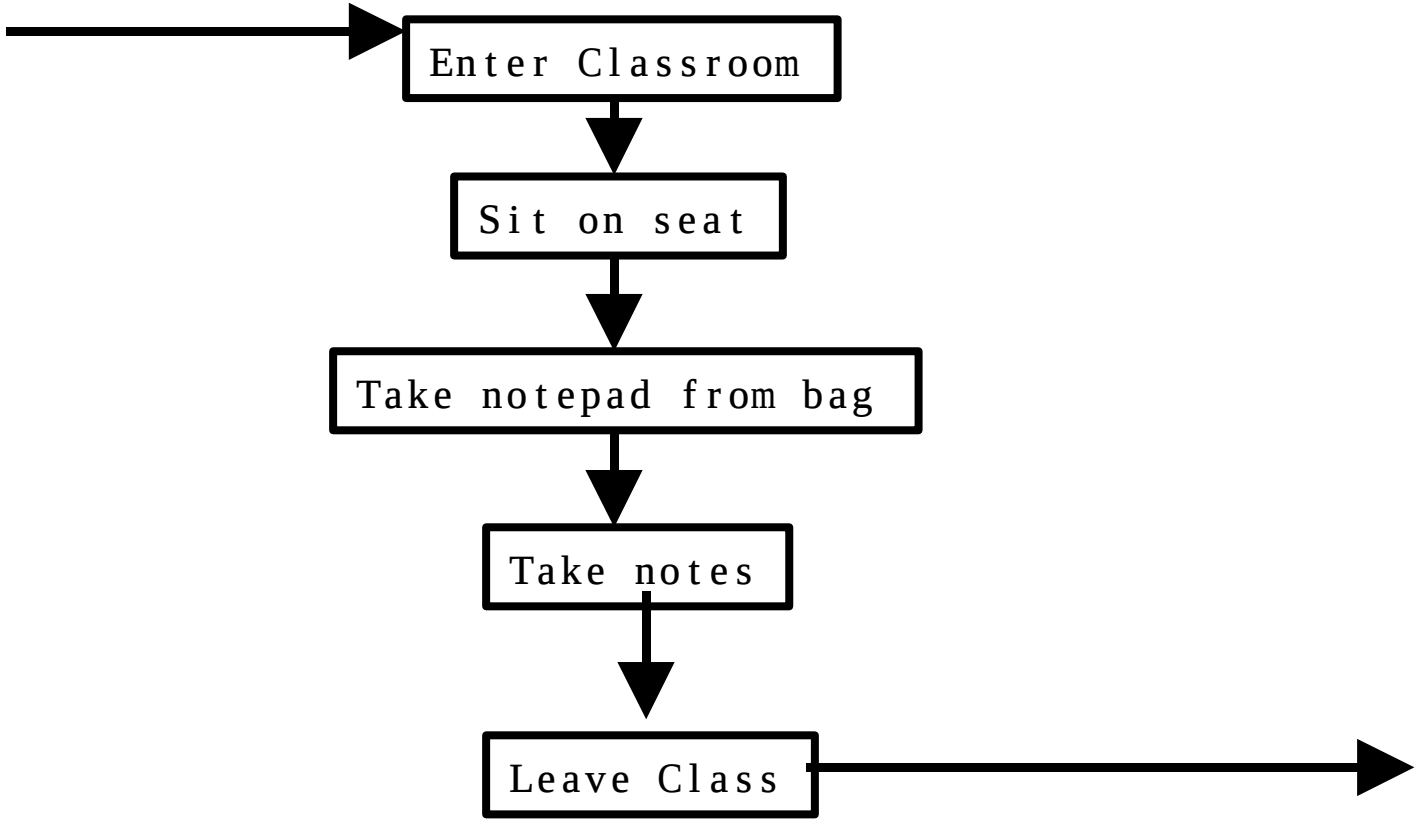
Pseudocode

- Informal language, English-like representation that helps programmer *think out* the problem
- No need for declararions, “”only executable statements

```
If student’s grade is greater than or equal to 70
     then print “ Passed ”
```

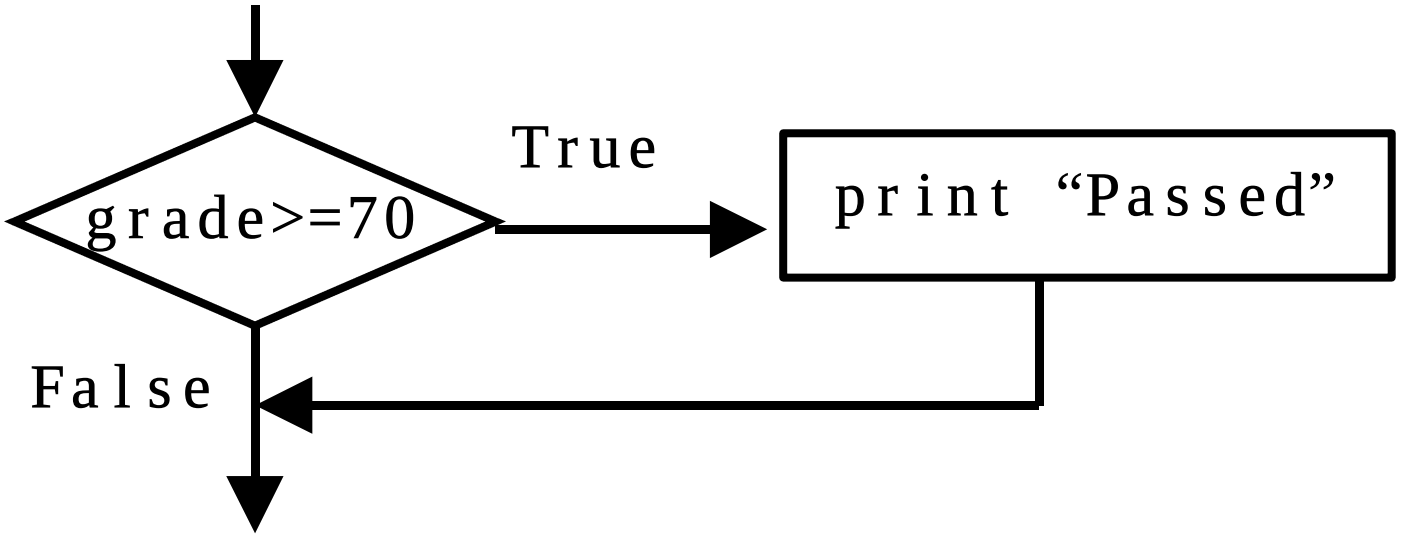
```
if (grade >= 60)
   cout<< “Passed ”;
```

Flowchart Sequence



Flowchart Decisions

- Diamond denotes a decision, contains an **expression** such as a **condition** which will be evaluated True (not 0) or False (0)



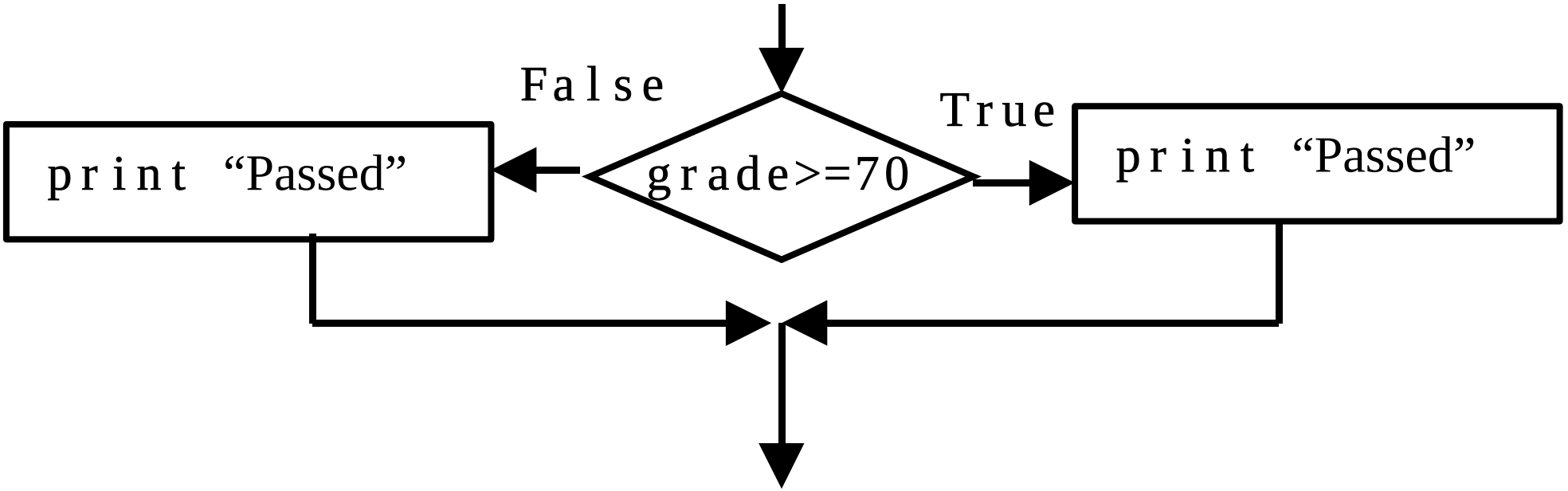
Conditional *?:* Operator

- Closely related to *if* / *else* structure
- Ternary expression, three operands
  - First operand is condition, evaluates True or False
  - Second operand is value for expression if first operand is true
  - Third operand is value for expression if conditional is false

Syntax: logical-OR-expr ? expr : conditional-expr

```
cout<<(grade>=70 ? “Passed” : “Failed”);
```

Double Selection... *if* / *else* and *?:*



Example Pseudocode / C

```
if student grade is greater than or equel to 90
  print “A”
else
  if student grade is greater than or equel to 80
    print “B”
  else
    if student grade is greater than or equel to 70
      print “C”
    else
      if student grade is greater than or equel to 60
        print “D”
```

Now the C

```
if (grade >= 90)
   cout<< “A”;
else
   if (grade >= 80)
      cout<< “B”;
   else
      if (grade >= 70)
         cout<< “C”;
      else
         if (grade >= 60)
            cout<< “D”;
         else
            cout<<“F”;
```

- Not very elegant looking
- Difficult to read this code, too much indentation
- Perhaps we can do better...

Indentation

- Indentation of code is **extremely important** as it makes your programs human readable
- Recommended to use **three spaces** for statements under control of *if* and *else*

```
if (grade >= 60)
   cout<<“Passed”;
else
   cout<<“Failed”;
```

Improved Code

```
if (grade >= 90)
   cout<< “A”;
else if (grade >= 80)
   cout<< “B”;
else if (grade >= 70)
   cout<< “C”;
else if (grade >= 60)
   cout<< “D”;
else
   cout<<“F”;
```

- More elegant looking
- Both forms equivalent
- Easier for humans to read this
- We can write the pseudocode in this form also

Selecting from a list...

- Just take some time to look at an advanced form of selection structure
- Saves on many nested else statements with an if
- Called switch statement

switch and **case** and **break**

```
/* Counting letter grades */
#include <stdio.h>

main()
{
  int grade;
  int aCount = 0, bCount = 0, cCount = 0,
     dCount = 0, fCount = 0;

  printf("Enter the letter grades.\n");
  printf("Enter the EOF character to end input.\n");

  while ( ( grade = getchar() ) != EOF) {

    switch (grade) {    /* switch nested in while */

      case 'A': case 'a':  /* grade was uppercase A */
        ++acount;        /* or lowercase a */
        break;

      case 'B': case 'b':  /* grade was uppercase B */
        ++bcount;        /* or lowercase b */
        break;

      case 'C': case 'c':  /* grade was uppercase C */
        ++ccount;        /* or lowercase c */
        break;

      case 'D': case 'd':  /* grade was uppercase D */
        ++dcount;        /* or lowercase d */
        break;

      case 'F': case 'f':  /* grade was uppercase F */
        ++fcount;        /* or lowercase f */
        break;

      case '\n': case' ':  /* ignore these in input */
        break;

      default:       /* catch all other characters */
        cout<<"Incorrect letter grade entered.";
        cout" Enter a new grade.\n";
        break;
    }
  }
  cout"\nTotals for each letter grade are:\n";
  cout<<"A: “<< aCount<<endl;
  cout<<"B:”<< bCount<<endl;
  cout<<"C:”<< cCount<<endl;
  cout<<"D: ”<< dCount<<endl;
  cout<<"F: ”<< fCount<<endl;
  return 0;
}
```

Revising Basic Constructs

- Expression evaluations
- Control Structures
  - Sequence, Selection and iteration

Arithmetic Operators

C++ operation	Arithmetic operation	Algebraic expression	C++ Expression
Addition	+	f + 7	f + 7
Subtraction	-	p - c	p - c
Multiplication	*	bm	b * m
Division	/	x / y	x / y
Modulus	%	r mod s	r % s

Precedence of Operators

- My Dear Aunt Sally
- (* / %) (+ -)
- Parenthesis *( )* always comes first
- If there are several operators, evaluation is left to right
- what is the algebraic equivalent of the following
    - z = p * r % q + w / x - y
- Rewrite it using appropriate parenthesis!

Constants and Literals

- Integer constants - 1234, 0345 (Octal), 0x1234 (Hex)
- Long integer constant - 12345678L
- Floating-point constant - 47.324, 3.2e13
- Character constant - ‘p’, ‘\n’ (new line)
- String constant - “Hello there” (but there is no string type?… later folks)

Relational Operators

- = = is equal to
- != is not equal to
- > is greater than
- < is less than
- >= is greater than or equal to
- <= is less than or equal to

Expressions

- Parenthesis give order of evaluation.
- Syntax *x*+=*y* is the same as *x*=*x*+*y*
- Syntax *x*++ adds one to **x** (*inc ax* in assembler).
- Similarly *x*- - subtracts one from **x**. The syntax **x=y=0** is valid, and does as you would expect.
- Logical comparisons evaluate to 0 (False) or 1 (True). Any non-zero result is taken as True.

Control Structures

- Structures
  - *if/else*
  - *while*
  - *for*
  - *switch*
- These are much the same in Java !

Control Structures

- Programs exhibit three behaviours
  - sequence
    - execute statements sequentially, we have seen this with the debugger
  - selection
    - choose between different execution paths, may decide to execute compound statements, or not depending on conditions
  - iteration
    - may have to do the same thing many times over, repetition is something computers do very well

Selection

- The *if* statement.

Syntax:

if (<condition>) <statement1>;

if (<condition>) <statement1>;
else <statement2>;

```
#include <stdio.h>
main(){
  int num1, num2;
  printf("Enter two integers, and I will tell you\n");
  printf("the relationships they satisfy: ");
  scanf("%d%d", &num1, &num2);   /* read two integers */
  if (num1 == num2)
    printf("%d is equal to %d\n", num1, num2);
  if (num1 != num2)
    printf("%d is not equal to %d\n", num1, num2);
  if (num1 < num2)
    printf("%d is less than %d\n", num1, num2);
  if (num1 > num2)
    printf("%d is greater than %d\n", num1, num2);
  if (num1 <= num2)
    printf("%d is less than or equal to %d\n",
          num1, num2);
  if (num1 >= num2)
    printf("%d is greater than or equal to %d\n",
          num1, num2);
  return 0;   /* indicate program ended successfully */
}
```

Code this up, use debugger to trace \ step.

Careful !

- A statement like:- ***if (x=3) {},*** probably a mistake (surely **if (x==3) {}** was intended?), will not generate a compiler error. It will assign 3 to **x**, which evaluates as True.
- A neat idea is to write it as **if (3==x)**. This will generate an error if you accidentally type it in as **if (3=x)**.

Selection with Alternatives

- If … else

```
If (grade >= 60)
   cout<<“Passed \n”;
else
   cout“Failed \n”;
```

Exercise: Write a program to assign exam grades to percentages.The user inputs the percentage from the keyboard, the program outputs the percentage to the screen

Exercise / Tutorial

- Do the following questions from chapter 1 and keep the program listings in your workshop folder.
- Also keep screen dumps of program runs.
- This **may be assessed** later.
  - 2.14 (coding errors)
  - 2.17, 2.18, 2.19, 2.26, 2.27

Keeping Your CA212 Exercises

- Keep a special folder for CA212 exercises
- Grades will be affected by your exercises
- Tutors will award grades for up to date work on a week to week basis
- This forms a part of your continuous assessment.