CODE SAMPLES

The following code and algorithm samples are used throughout the IB Computer Science course. Using these code examples, or a combination of them will help you during problem-solving activities, when answering past papers, and when creating your solution for the IA.

Note: This document is a work in progress and will be updated/edited as the year progresses.

More study materials can be found at www.ibcompsci.net

Table of Contents

COMMENTS

It is always good practice to include comments in your program. This acts as internal documentation. Not only does it help you understand you own code better, it also helps other people understand your code. Comment statement are ignored by your interpreter/compiler, so you can use them as much as you like. Comments are useful for commenting out code you don't need for a while, instead of deleting it.

Further reading: http://bit.ly/2L0Akmn

Single-line comments can be used throughout your program; however, they are really useful when used at the end of statements.

A single like comment uses two forward slashes.

```
// This is a single line comment
```

Multi-line comments are often used at the beginning of your program and before method declarations.

A multi-line comment uses a combination of forward slashes and asterisks.

```
/*  This is a multi-line comment that allows you to say what you have
    to say on multiple lines. You could use single-line comments on
    multiple lines if you so wish. */
```

Working example

```
/**
 *  This program adds two numbers together and outputs the total.
 *
 *  @author: Maurice Moss
 *  @version: 1.0
 */
public static void main(String[] args) {

        int num1 = 35;
        int num2 = 15;
        int total = num1 + num2;                    // add num1 and num2 together

        System.out.println("The total is " + total);        // output the total
}
```

Variables

Variable provide a location to store items of data during the running of a program. Think of them as named boxes in memory you can place items into of a certain type. These items can then be manipulated and used at any time.

The variable types are String, integer, char, Boolean, long and float.

Further reading: http://bit.ly/2Jf2YeC

```
String firstName = "Maurice";                  // variables of type String
String secondName = “Moss”;
String telNumber = “0123456789”;
String jobTitle = "Programmer";

char firstInitial = 'M';                       // variables of type char
char secondInitial = ‘M’;

int yearBorn = 1998;                           // variables of type integer
int age = 20;

boolean employed = true;                       // variable of type Boolean

double decimalNumber = 3.14;                   // variable of type double
long longNumber = 37493729L;                   // variable of type long
float floatNumber = 1.3f;                      // variable of type float
```

Working example

```
public static void main(String[] args) {

        String firstName = "Maurice";
        String secondName = “Moss";
        int age = 20;

        System.out.println(firstName + " " + secondName + “ is ” + age + “ years old.”);

}
```

Output

```
Maurice Moss is 20 years old.
```

Branching

To execute a block of code based on a certain condition, you will need to use an if-statement. Based on a condition your program will change its course of action.

Analogy: You are driving along the road and come to a fork in the road. In order to make a decision as to which fork to take, you need to evaluate your options based on a condition, such as traffic congestion, for example.

Further reading: http://bit.ly/2N7w8yB

Skeleton code

```
if (condition) {
        do something
else
        do something else
```

```
if (left fork is quiet) {
        take the left fork
else
        take the right fork
```

Working example

Note: A Boolean variable can only be in one of two states, true or false.

```
public static void main(String[] args) {

        Boolean leftFork = false;          // false denotes a quiet road

        if (leftFork == false) {
                System.out.println("Take the left fork in the road");
        } else {
                System.out.println("Take the right fork in the road");
}
```

Sometimes you may need to test more than one thing.

Skeleton code

```
if ( condition ) {
        do something
}
else if ( condition ) {
        do something
}
else {
        do something
}
```

```
if ( light == red ) {
        stop
}
else if ( light == green ) {
        go
}
else {
        wait
}
```

Working example

Note: When comparing strings in Java use .equals() instead of ==

```
public static void main(String[] args) {

        String light = "Green";

        if (light.equals("Red") {
                System.out.println("Stop");
        }
        else if (light.equals("Green") {
                System.out.println("Go");
        }
        else {
                System.out.println("Wait");
        }
}
```

Repetition

In order to execute a block of code a certain number of time, or until a condition is true, you will need to use a loop structure. Where an if statement only executes a block of code once, a loop will execute a block multiple times.

Further reading: http://bit.ly/2ukEkov

The three loops are:

- While
- For
- Do-while

While

Skeleton code (While loop)

```
while ( condition ) {
        do something
}
```

```
while ( number is less than 10 ) {
        number = number + 1
}
```

Working example

```
public static void main(String[] args) {

        number = 0;

        while (number < 10) {
                number++;                    // short for number = number + 1
                system.out.println("number " + number);
        }
}
```

Output

```
Number 0
Number 1
Number 2
Number 3
…
```

For

Skeleton code

```
for (initialise ; condition ; increment) {
        do something
}
```

```
for (int i = 0; i < 10; i++) {
        do something
}
```

Working example

```
public static void main(String[] args) {

        for (int i = 0; i < 10; i++) {
                system.out.println(i);
        }
}
```

Output

```
0
1
2
3
…
```

Further explanation

Enhanced for loop

The enhanced for loop is a more compact version of the traditional for loop. It useful for iterating through an array.

Skeleton code

```
for (declaration : expression) {
        do something
}
```

```
for (String name : names) {
        do something
}
```

Working example

```
public static void main (String args []) {

        String [] names = {"John", "William", "David"};        // see arrays further down

        for (String name : names) {
                System.out.print(name);
                System.out.print(",");
        }
}
```

Output

John, William, David

Do-while

The while loop and for loop always execute their tasks after checking their specified condition. If you would like to execute a certain piece of code before checking the condition, use a do-while loop.

Skeleton code

```
do {
        do something
}
while (condition);
```

```
do {
        number = number + 1
}
while (number is less than 10);
```

Working example

```
public static void main (String[] args) {

number = 0;

do {
        number++;
        system.out.println("number " + number);
}
while (number < 10);
```

Output

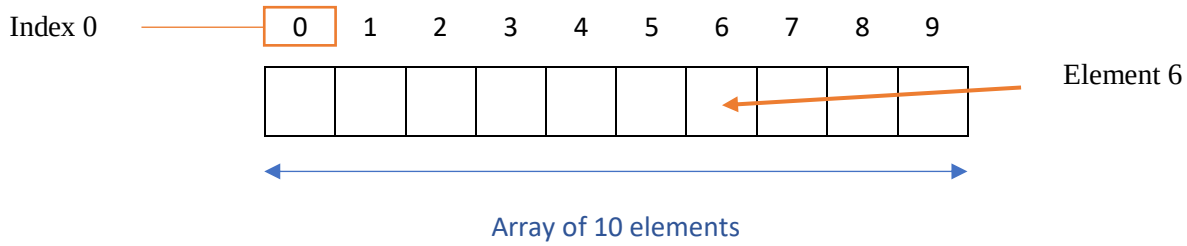
```
Number 1
Number 2
Number 3
…
```

ARRAYS

Single arrays

Sometimes you will want to store a number of items of the same type. Instead of using multiple variables to store the items, you can use an array container.

Further reading: http://bit.ly/2Oo1Nx7



Each box or item in the array is called and element, and each element is assigned a number or index. Computers always start counting at 0, so the first element has the index 0.

Declaration and initialization

To declare an array that accepts only integers, you would write the following code:

```
int[] myArray;          // the array can be named anything that is descriptive
```

To say how many elements you would like allocated in memory:

```
myArray = new int[10];
```

Both of the above statements can also be combined on one line:

```
int[] myArray = new int [10];
```

Once you have declared and initialized an array, you can then assign items to each element:

```
myArray[0] = 23;
myArray[1] = 4;
myArray[2] = 54;
myArray[3] = 17;
myArray[4] = 155;
myArray[5] = 261;
myArray[6] = 87;
myArray[7] = 12;
myArray[8] = 79;
myArray[9] = 37;
```

Because computers always start counting from zero, the last element has an index of 9.

In diagrammatical form, the array would look something like this once all the above values had been assigned.

0	1	2	3	4	5	6	7	8	9
23	4	54	17	155	261	87	12	79	37

The contents of the array are now stored for further use by the program.

Another method of declaring, initializing and assigning value to an array is using this shorter method:

```
int[] myArray = { 23, 4, 54, 17, 155, 261, 87, 12, 79, 37 };
```

Using a for loop, the contents of the array can be outputted (See also enhanced for loop example previously)

Working example(s)

```
public static void main(String args[]) {

        int[] numbers = new int [5];            // declare and initialize array with 5 elements

        numbers[0] = 23;                        // assign integers to each element
        numbers[1] = 4;
        numbers[2] = 54;
        numbers[3] = 17;
        numbers[4] = 155;

        System.out.println(numbers[0]);         // output contents of element 0
        System.out.println(numbers[1]);         // output contents of element 1
        System.out.println(numbers[2]);         // output contents of element 2
        System.out.println(numbers[3]);         // output contents of element 3
        System.out.println(numbers[4]);         // output contents of element 4

}
```

A quicker way to output the contents of an array is by using a for loop.

```
public static void main(String args[]) {

        int[] numbers = new int [5];            // declare and initialize array with 5 elements

        numbers[0] = 23;                        // assign integers to each element
        numbers[1] = 4;
```

```
        numbers[2] = 54;
        numbers[3] = 17;
        numbers[4] = 155;

        for (int i = 0; i <= numbers.length; i++) {          // loop through each element of the array
                System.out.println(numbers[i]);              // and print contents of each element
        }
}
```

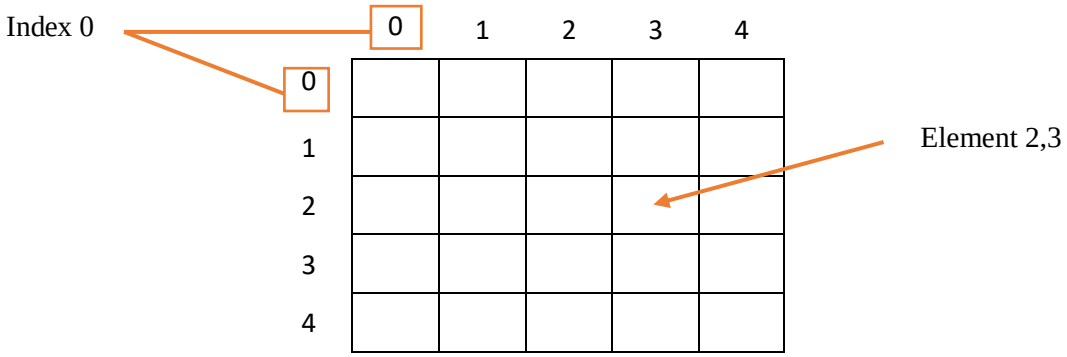
Output

Both of the above example would have the same output.

```
23
4
54
27
155
```

Multidimensional arrays

Whereas normal arrays have one set of square brackets, two-dimensional, or multidimensional arrays have two sets of square brackets. Pictorially, a multidimensional array would look more like a grid.



Array of 25 elements

Declaring and initializing a multidimensional array is just like a normal array, but with double brackets. In the example below, an array is declared and initialized with 5 rows and 5 columns.

```
int[][] myArray = new int[5][5]
```

In a similar way to a normal array, values are assigned like so.

```
myArray[0][0] = 23;          // assign value to element on row[0], column[0]
myArray[0][1] = 4;           // assign value to element on row[0], column[1]
myArray[0][2] = 54;          // assign value to element on row[0], column[2]
…

myArray[1][0] = 17;          // assign value to element on row[1], column[0]
myArray[1][1] = 155;         // assign value to element on row[1], column[1]
myArray[1][2] = 261;         // assign value to element on row[1], column[2]
…

myArray[2][0] = 87;          // assign value to element on row[2], column[0]
myArray[2][1] = 12;          // assign value to element on row[2], column[1]
myArray[2][2] = 79;          // assign value to element on row[2], column[2]
…
```

Using the shorter method as shown before, you can declare, initialise and assign like so.

```
int[][] myArray = {
        {23, 4, 54},
        {17, 155, 261},
        {87, 12, 79}
};
```

Outputting the contents of a multidimensional array is the same when using standard output statements.

Working example(s)

```
public static void main(String args[]) {

        int[][] numbers = new int[5][5];  // declare and initialize array with 5 elements

        numbers[0][0] = 23;
        numbers[0][1] = 4;
        numbers[0][2] = 54;
        …

        System.out.println(numbers[0][0]);          // output contents of element 0,0
        System.out.println(numbers[0][1]);          // output contents of element 0,1
        System.out.println(numbers[0][2]);          // output contents of element 0,2
        …

}
```

When output the contents of a multidimensional array using a for loop, a nested loop is needed. The example below will print the contents of the array in a grid format.

```
for (int i = 0; i < 5; i++) {                       // visit each row one at a time
        for (int j = 0; j < 5; j++) {               // loop through each column until the end
                System.out.print(numbers[i][j]);// output each element
        }
        System.out.println();
}
```

PSEUDOCODE

Pseudocode is a simplified programming language and an alternative way to represent a program in many different languages.

Video: http://bit.ly/2uen2t5

Typical example

```
TOTAL = 0
NUMBERS[49]

loop N from 0 to 50
        if NUMBERS[N] > 0 then
                TOTAL = TOTAL + NUMBERS[N]
        end if
end loop
output "The total is ", TOTAL
```

IB supplies two documents that offer further reading and examples, one of which is supplied during the exams as a guide. You can download both documents on this website.

Approved notation for developing pseudocode (This document is allowed in exams)

Pseudocode in Examinations (This document is not allowed in exams)

- Standard Data Structures
- Examples of Pseudocode

FLOWCHARTS

Flowcharts are another way to represent algorithms in a pictorial/diagrammatical way instead of plain text, such code or pseudocode. It can often make it much easier to quickly see how an algorithm operates.

Further reading: http://bit.ly/2NIrisE