

O-TECH

a way to learning and professionalism

2nd Edition
March 2010

Brought to you by
The Origin

ECONOMICAL IMPORTANCE OF SOFTWARE

PAGE 26

EVOLUTION OF DATABASE MODELS

PAGE 19

CHECK OUT NEW SECTION

CONCEPTS

PAGE 16 & 17

EDITORIAL

The Origin motto is “A divine guidance for the IT people of Pakistan”; keeping it in mind, I decided to design a complete IT book for the people who are keen to gain maximum knowledge in minimum time. There is a saying that there is no shortcut to success. So why make shortcuts if we want to succeed in our lives? Why not put some hard work for ourselves? That is the basic concept of this book.

The book contains a jackpot package including all the material needed to become an IT professional. IT field demands a lot of knowledge, research and practical experience. To help our fellows in Information Technology, we are going to give concepts of programming languages, databases, web development and software engineering.

As “The Origin” Team aims to work for its people; so we tried our best to include those topics which were suggested to us in your feedback.

This project also comprises of Online Classes that will be available on our online website (Direct Link: www.TheOrigin.Webnode.com). You will be facilitated with live discussions at any time. You can also post and share things on the website with other people. You are free to discuss any topic related to online classes. Come, join our live discussions and give us your feedback at team_theorigin@yahoo.com.

I hope you would like our efforts for the online book and for further detail visit our online classes on “The Origin” website. We need your feedback and support as we are working really hard to make things possible for you.

For any suggestion, comment or query; contact me at team_theorigin@yahoo.com.

Hafiz Fahad Hassan

O-TECH Team

Editor

Hafiz Fahad Hassan

Graphic Designer

Sara Muhammad

Contributors

Bilal Hafeez

Muhammad Umair

Zeeshan Safdar

Umer Asif

Hafiz Fahad Hassan

Madiha Qureshi

Want to contribute?

Mail us at

editor_theorigin@yahoo.com

About O-Tech

O-Tech is a product of “The Origin” which aims to provide a divine guidance for the IT people of Pakistan. O-Tech is a unique sort of idea comprises of a comprehensive guide for the people who are related to the field of Information Technology in any way, i.e. from the students of IT to the IT professionals. This is for all, who are keen to gain practical knowledge.

Along with this book, The Origin has also arranged Online Classes on the online services website of The Origin. In those classes you will be able to share and discuss your own problems 24/7 live with your online friends.

The Origin is strengthening its roots rapidly all around due to you people. We admire your support and affection towards us. Thanks.

CONTENTS

.....▶ **pROGRAMING**

<i>Java</i>	4
<i>C++</i>	6
<i>C</i>	8
<i>Visual Basic</i>	12

.....▶ **cONCEPTS**

<i>Preprocessors Directives</i>	16
<i>Header Files</i>	17

.....▶ **dATABASES**

<i>Databases Article</i>	19
--------------------------	-----------

.....▶ **WEB dEVELOPMENT**

<i>HTML</i>	23
-------------	-----------

.....▶ **sOFTWARE eNGINEERING**

<i>Article</i>	26
----------------	-----------

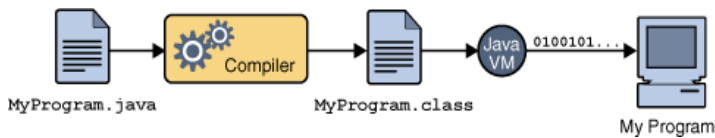
The Java Programming Language

By
Bilal Hafeez

The Java programming language is a high-level language that can be characterized by all of the following buzzwords:

- ☐ Simple
- ☐ Object oriented
- ☐ Distributed
- ☐ Multithreaded
- ☐ Dynamic
- ☐ Architecture neutral
- ☐ Portable
- ☐ High performance
- ☐ Robust
- ☐ Secure

In the Java programming language, all source code is first written in plain text files ending with the .java extension. Those source files are then compiled into .class files by the javac compiler. A .class file does not contain code that is native to your processor; it instead contains *bytecodes* — the machine language of the Java Virtual Machine¹ (Java VM). The java launcher tool then runs your application with an instance of the Java Virtual Machine.

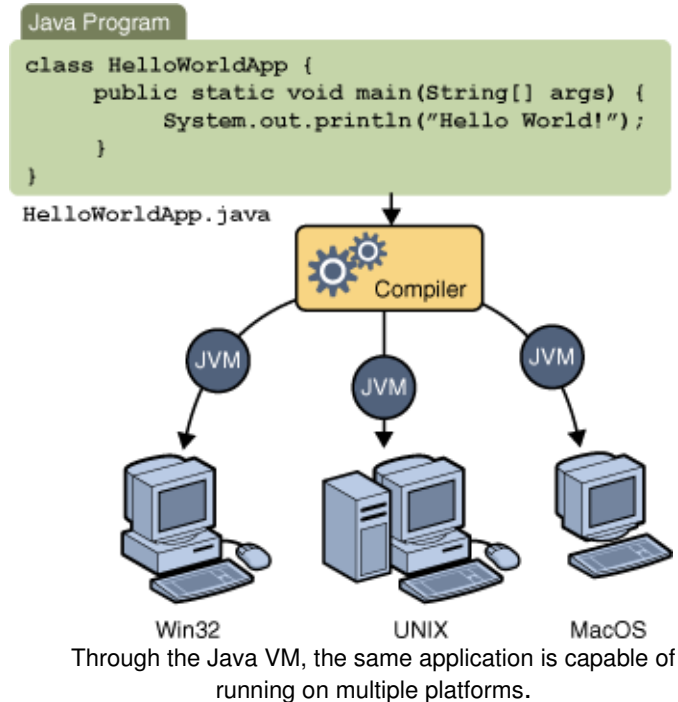


Because the Java VM is available on many different operating systems, the same .class files are capable of running on Microsoft Windows, the SolarisTM Operating System (Solaris OS), Linux, or Mac OS. This includes various tasks such as finding performance bottlenecks and recompiling (to native code) frequently used sections of code.

The Java Platform

A *platform* is the hardware or software environment in which a program runs. I've already mentioned some of the most popular platforms like Microsoft Windows, Linux, Solaris OS, and Mac OS. Most platforms can be described as a combination of the operating system and underlying hardware. The Java platform differs from most other platforms in

that it's a software-only platform that runs on top of other hardware-based platforms.

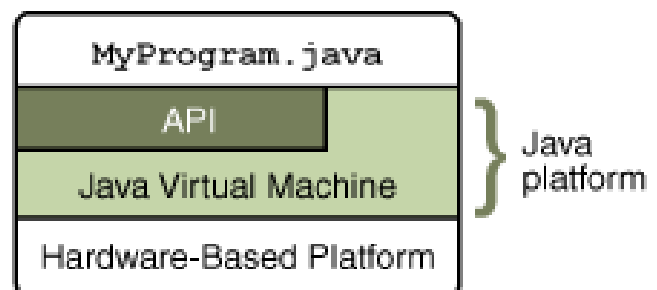


The Java platform has two components:

- The *Java Virtual Machine*
- The *Java Application Programming Interface (API)*

You've already been introduced to the Java Virtual Machine; it's the base for the Java platform and is ported onto various hardware-based platforms.

The API is a large collection of ready-made software components that provide many useful capabilities. It is grouped into libraries of related classes and interfaces; these libraries are known as *packages*.



The API and Java Virtual Machine insulate the program from the underlying hardware.

As a platform-independent environment, the Java platform can be a bit slower than native code. However, advances in compiler and virtual machine technologies are bringing performance close to that of native code without threatening portability.

What Can Java Technology Do?

The general-purpose, high-level Java programming language is a powerful software platform. Every full implementation of the Java platform gives you the following features:

- **Development Tools:** The development tools provide everything you'll need for compiling, running, monitoring, debugging, and documenting your applications. As a new developer, the main tools you'll be using are the javac compiler, the java launcher, and the javadoc documentation tool.
- **Application Programming Interface (API):** The API provides the core functionality of the Java programming language. It offers a wide array of useful classes ready for use in your own applications. It spans everything from basic objects, to networking and security, to XML generation and database access, and more. The core API is very large.
- **Deployment Technologies:** The JDK software provides standard mechanisms such as the Java Web Start software and Java Plug-In software for deploying your applications to end users.
- **User Interface Toolkits:** The Swing and Java 2D toolkits make it possible to create sophisticated Graphical User Interfaces (GUIs).
- **Integration Libraries:** Integration libraries enable database access and manipulation of remote objects.

INSHALLAH Next time I will tell you to write your first application. Those instructions will be for

users of the NetBeans IDE. The NetBeans IDE runs on the Java platform, which means that you can use it with any operating system for which there is a JDK 6 available. These operating systems include Microsoft Windows, Solaris OS, Linux, and Mac OS X.

To write your first program, you'll need:

The Java SE Development Kit 6 (JDK 6)

For Microsoft Windows, Solaris OS, and Linux:

<http://java.sun.com/javase/6/download.jsp>

For Mac OS X: <http://connect.apple.com>

The NetBeans IDE

For all platforms:

<http://www.netbeans.info/downloads/index.php>

**Join Us
on facebook**

O-TECH
get into learning

Join the fanpage and stay connected to
get updates from the O-TECH

Building Concepts C++

By

Muhammad Umair

Article 02

Hi Guys!! Hope you are fine and enjoying learning with me. 😊

In the previous article, we explored a little bit about C++ and then we started with C++ *structs*.

We learnt that how to:

- Declare a C++ struct
- Calculate struct size
- Declare struct objects
- Initialize struct objects
- Access and modify struct fields using the dot operator (.)

In this article, we'll explore C++ structs in a bit more detail.

So, here we go ...

Pointers to Structures & Arrow Operator (->)

Like any other type, structures can be pointed by pointers of its own type:

```
struct Person{
    char name[30];
    int age;
    float height;
};

int main()
{
    Person P;           // Declares an
object of type Person
    Person * PPtr;      // Declares a
pointer to point to objects of
                        // structure
type Person
    return 0;
}
```

In the above code, **P** is an object of structure type Person and **PPtr** is a pointer to point to objects of structure type Person. So, the following code would also be valid:

```
PPtr = &P;
```

Here, **PPtr** would be assigned a reference to the object **P** (its memory address).

We'll now continue with another sample program that includes pointers, which will serve to introduce a new operator: the arrow operator (->):

```
#include <iostream>

using namespace :: std;

struct Person{

    char name[30];
    int age;
    float height;
};

int main ()
{
    Person P;
    Person * PPtr;
    PPtr = &P;

    cout << "Enter Name: ";
    cin.getline (P.name, 30);
    cout << "Enter Age: ";
    cin >> P.age;
    cout << "Enter Height: ";
    cin >> P.height;

    cout << "\nYou Have Entered:\n";
    cout << PPtr -> name << " " <<
PPtr -> age << " " << PPtr ->
height;

    /* Above line will print the
name, age and height of object P
that is pointed by PPtr */

    return 0;
}
```

The previous code includes an important introduction: the arrow operator (->). This is a dereference operator that is used exclusively with pointers to objects with members. This operator serves to access a member of an object to which we have a reference. In the example we used:

```
PPtr -> name
```

Which is equivalent to:

```
(*PPtr).name
```

Both expressions `PPtr -> name` and `(*PPtr).name` are valid and both mean that we are evaluating the member title of the data structure **pointed by** a pointer called **PPtr**. It must be clearly differentiated from:

```
*PPtr.name
```

Which is equivalent to:

```
*(PPtr.name)
```

And that would try to access the value pointed by a hypothetical pointer member called *name* of the structure **object** *PPtr* (which in this case would not be a pointer and the program will generate an error!).

Nested Structures

Structures can also be nested so that a member of a structure can also be another structure itself.

```
struct Date{
    int date;
    int month;
    int year;
};

struct Person{
    char name[30];
    int age;
    float height;
```

```
    Date dob; // Date of Birth
} P1, P2; // Define objects of type
Person

Person * PPtr = &P1;
```

After the above declarations, we can use any of the following expressions:

```
P1.name
P2.dob.month
P1.dob.year
PPtr -> dob.year
```

(Where, by the way, the last two expressions refer to the same member)

Using Structures with Functions

There are two ways to pass the information in structures to functions. You can either pass the entire structure or pass the individual members of a structure.

By default, structures are passed by value.

Structures and their members can also be passed by reference by passing either references or pointers. To pass a structure by reference, pass the address of the structure object or a reference to the structure object.

Let's consider another sample program:

```
#include <iostream>

using namespace :: std;

struct Person{
    char name[30];
    int age;
    float height;
};

void Print1(Person P)
{
    cout << "\nYou Have Entered:\n";
    cout << P . name << " " << P .
age << " " << P . height;
}
```



```

void Print2(Person *P)
{
    cout << "\nYou Have Entered:\n";
    cout << P -> name << " " << P ->
age << " " << P -> height;
}

void Print3(Person &P)
{
    cout << "\nYou Have Entered:\n";
    cout << P . name << " " << P .
age << " " << P . height;
}

int main ()
{
    Person P;

    cout << "Enter Name: ";
    cin.getline (P.name, 30);
    cout << "Enter Age: ";
    cin >> P.age;
    cout << "Enter Height: ";
    cin >> P.height;

    Print1(P);          // Pass by value
    Print2(&P);          // Pass by
reference
    Print3(P);          // ??? Check
Definition of Print3 and then decide
                        // yourself that what
will happen in this case?
    return 0;
}

```

Important

Passing structures (and especially large structures) by reference is more efficient than passing them by value (which requires the entire structure to be copied).

That's All for Now!!

We are done with C++ *structs*. Do practice all the things that you have learned in this article. Code all the above sample programs in your compiler, compile/execute them and observe the results. When you are done, make minor changes in your

programs and then execute them. This will surely be helpful to you in boosting your learning!!

Do send me your feedback if you have any problems or suggestions!

Next time we will begin with *classes*.

Until then, Take Care & Bye ...

By

Zeeshan Safdar

Building Concepts C

In the previous article we have seen some basic functionalities of C. In this article we are going to elaborate *stdio.h* library functions:

1. printf()
2. scanf()
3. gets()
4. puts()
5. getch()
6. getche()
7. clrscr() and
8. gotoxy()

By

Zeeshan Safdar

The Printf() Function

The *printf()* function is used to print values and text on the output device in a specific format. It is also called the formatted output function. Its general syntax is:

printf (Control String [, List of Arguments]);

Control String

It consists of text, the format specifier and the escape sequence. It is written within double quotes.

- The text specifies the message that is to be printed along with the values.
- A format specifier specifies the format according to which a value is to be printed.

List of Arguments

It consists of a list of variables, constants or arithmetic expressions. All separated by commas. It is an optional part if you just want to print some text on the screen not going to print a value of a variable.

Some Examples

1. printf ("C is a powerful tool");
2. printf (" Height h = %d", h);
3. printf (" Height h = %f", h);

4. `printf (" Height h = %d\n Width w = %d", h, w);`

What will be the shown on screen if $h = 20.5$ and $w = 15$?

1. C is a powerful tool
2. Height $h = 20$
3. Height $h = 20.5$
4. Height $h = 20$

Width $w = 15$

Try yourself I may be wrong! How this output is possible? Where is “%d” and “\n” in the output?

These are the format specifier and escape sequence respectively. Now the next question is, if I want to print more than one variable then how would I specify them in arguments list? Example-4 is elaborating this confusion. First variable is for first format specifier and second for second specifier.

Another question why example-2 and example-3 differ in results? If still finding the reason, then take another look to the format specifiers of both examples. In example-2, it is “%d” and in example-3 it is “%f”. Thus it is clear format specifier is not only use to tell compiler to print corresponding variable but it also instruct compiler how the output must look like.

Next we are going to discuss *Format Specifier* and *Escape Sequence* in detail.

Format Specifier

A format specifier specifies the *data type*, *field width* and *format* of a value. Its general syntax is like, without any spaces:

% flags field-width precisions h/l/L con-character

% **sign** indicates the beginning of the specifier

Flags the flag characters **+**, **-**, **#** and **blank** are used to effect the output as

+ sign A plus or a minus sign is shown if value is signed

- sign To justify output to left. By default it is right justified

sign The octal values are preceded by 0, hexadecimal are 0x or 0X and the floating point values contain a decimal point.

Field-Width

It specifies the minimum numbers of columns reserved for the output value.

Precision

Used with floating point values and *.n* precision specifies that *n* decimal places are to be output by rounding off.

h/l/L

if we are using short, long or long double data type *e.g long int, long float*, then h/l/L are to indicate them respectively.

Con-Character

Conversion character, convert value and then display on screen. As we all known for computer everything is 0 or 1. A variable name refers to a memory location. Is it complete definition for variable name? No some main point is missing. Undoubtedly a variable name refers to a memory location but starting address of the memory location. We are all familiar with size of data types. Thus con-character just helps compiler to tell it how many max bytes it need to need from starting location. If you are reading intentionally that you might have notice I have used a word max bytes. As if you are printing a string of 100 characters but output will only be till the NULL or ‘\0’ character.

Commonly Used Conversion Characters

- d for integer type values
- f for float types values
- c for character types and
- s for strings

There is many more con-character that I have discus. When you need search and use them. For need I mean if you want to show large numbers as exponential form, then use con-character *g* or *G*. you can find all this easily whenever you need them.

Examples

1. `printf ("A = %+5, B = %7.1f", a, b);`
2. `printf ("A = %-5d, B = %+7.1f", a, b);`

Write these instructions into your compiler, run it and match with my results. I am using $a = 65$ and $b = 511.13$. first row of the tables show your screens columns and second is your expected output.

1

0	1	2	3	4	5	6	7	8	9	1	1	1	1	1	1	1	1	1	2	2	
										0	1	2	3	4	5	6	7	8	9	0	1
A	=						+	6	5	,	B	=			5	1	1	.	1	3	

2

0	1	2	3	4	5	6	7	8	9	1	1	1	1	1	1	1	1	1	2	2	
										0	1	2	3	4	5	6	7	8	9	0	1
A	=	6	5							,	B	=		+	5	1	1	.	1	3	

Escape Sequence

Non-printable characters used to control printing on output device. It is a combination of a backslash '`\`' *control character* and *code character*.

If you have do practice on printf() function. You may find difficult to align your output. You want to print something on next line, want to give tab. That will be done through escape sequence.

Escape Sequence	Functionality
<code>\a</code>	Alert or Alarm. It causes beep sound in computer internal speakers.
<code>\b</code>	Backspace. Cause your cursor to move back one location but will not remove the previous character.
<code>\t</code>	Cause cursor to move a tab forward
<code>\n</code>	Cause cursor to move on the next line
<code>\f</code>	Feed that cause one blank sheet out to attached printer
<code>\r</code>	Carriage Return. Moves the cursor to the beginning of the current line
<code>\\</code>	To print ' <code>\</code> ' on output screen
<code>\"</code>	To print ' <code>"</code> ' on output screen
<code>\'</code>	To print ' <code>'</code> ' on output screen

Before moving on I like to suggest to play with printf() function a hour or so. As now I going to explain scanf() function that is totally different from what we have just learnt. scanf() is used to take input from keyboard.

The scanf() Function

it is used to get values into the variables from keyboard during execution of the program. Values are inputted in specified format. It general syntax is:

```
scanf ( Control String, List of Variables );
```

Control String

Written within double quotes and specifies the format to which a value is to be entered. Unless printf() control string we can only specify type of variable here not strings.

List of Variables

Contains the list of variables into to which value are to be entered. scanf() function not input the value into variables but rather into the memory location. We have to give memory address where we want to save our input. Do not think you have to give and remember memory address explicitly, but you can. What we need to do just append '&' sign, address operator, before variable name. but do not append '&' before string name. Reason will be discus in coming articles.

Examples

Let take some int a, float b and char[20] str.

1. `scanf ("%d", &a);`
2. `scanf ("%s", str);`
3. `scanf ("%s%d %f", str, &a, &b);`

As you can see to take input into int in example-1 I use '&' sign before variable 'a' but have not used in example-2 with string 'str'. scanf() will continue to take the input will you press return key. What about example-3? You can give input by return or space separated.

Question arise, if scanf() consider new input after space. Then how to input spaces into a string. Think and try. you will find the solution later in this article.

Using a Scanset

The scanf() function supports a general-purpose format specifier called a scanset. A scanset defines a set of characters. When scanf() processes a scanset, it will input characters as long as those characters are part of the set defined by the scanset. The characters read will be assigned to the character array that is pointed to by the scanset's corresponding argument.

Examples

When you use a scanset, scanf() continues to read characters, putting them into the corresponding character array until it encounters a character that is not in the scanset. Upon return from scanf(), this array will contain a null-terminated string that

consists of the characters that have been read. To see how this works, try this program:

```
main( )
{
    int a;
    char str[80], str2[80];
    scanf("%d%[abcdefg]%s", &a, str, str2);
    printf("%d %s %s", a, str, str2);
}
```

Enter 123abcdtye followed by. The program will then display 123 abcd tye. ENTER Because the "t" is not part of the scanset, scanf() stops reading characters into str when it encounters the "t." The remaining characters are put into str2.

You can specify an inverted set if the first character in the set is a ^. The ^ instructs scanf() to accept any character that is not defined by the scanset.

In most implementations you can specify a range using a hyphen. For example, this tells scanf() to accept the characters A through Z: %[A-Z]

One important point to remember is that the scanset is case sensitive. If you want to scan for both upper- and lowercase letters, you must specify them individually.

The gets() function

Answer of my earlier question. How to input spaces in a string? gets() facilitates you to input any kind of character, except '\n' as it stop inputing only press return key. Its general syntax is:

```
gets( string );
```

where 'string' refers to a string type variable.

The puts() function

It is used to print a string on the computer screen. A new line is inserted after printing the string. its syntax is:

```
puts( string )
```

The string may be a string constant or a string type variable.

In case of a string constant, it is enclosed in double quotes.

In case of a sting variable, it is written without using quotes.

The getch() and getche() Function

Both are used to get a single character from keyboard but difference is that getche() displays that character whereas as getch() does not display the inputted character. Their general syntax is:

```
[ var = ] getch();
```

```
[ var = ] getche();
```

Input character will store on var and control shift to the next statement as you press a key from your keyboard. Storing part is optional. Compiler will not force you to save inputted value into a variable. The normal use of getch() is to hold the screen.

The clrscr() Function

This function will clear the contents of your output screen and cursor shifts to upper left corner. It general syntax is:

```
clrscr();
```

The gotoxy() Funtion

This function shifts the cursor on the screen to a specified location. Its general syntax is:

```
gotoxy( x, y );
```

x and y specifies the x co-ordinate and y co-ordinate of the screen respectively.

I hope, I have successfully explained and elaborated a very important and commonly used C library "stdio.h". In next section we try to learn art and functions used for decision making in C.

By

Hafiz Fahad Hassan & Madiha Qureshi

VISUAL BASIC

Your Very First Visual Basic Program

Visual Basic lets you build a complete and functional Windows application by dropping a bunch of controls on a form and writing some code that executes when something happens to those controls or to the form itself.

This programming paradigm is also known as *event-driven programming* because your application is made up of several event procedures executed in an order that's dependent on what happens at run time. The order of execution can't, in general, be foreseen when the program is under construction.

This section offers a quick review of the event-driven model and uses a sample application as a context for introducing Visual Basic's intrinsic controls, with their properties, methods, and events. This sample application, a very simple one, queries the user for the lengths of the two sides of a rectangle, evaluates its perimeter and area, and displays the results to the user.

Adding Controls to a Form

We're ready to get practical. Launch the Visual Basic IDE, and select a Standard EXE project. You should have a blank form near the center of the work area. More accurately, you have a *form designer*, which you use to define the appearance of the main window of your application. You can also create other forms, if you need them, and you can create other objects as well, using different designers

One of the greatest strengths of the Visual Basic language is that programmers can design an

application and then test it without leaving the environment. But you should be aware that designing and testing a program are two completely different tasks.

- At *design time*, you create your forms and other visible objects, set their properties, and write code in their event procedures.
- At *run time* you monitor the effects of your programming efforts: What you see on your screen is, more or less, what your end users will see. At run time, you can't invoke the form designer, and you have only a limited ability to modify the code you have written at design time.



For instance, you can modify existing statements and add new ones, but you can't add new procedures, forms, or controls. On the other hand, at run time you can use some diagnostic tools that aren't available at design time because they would make no sense in that context

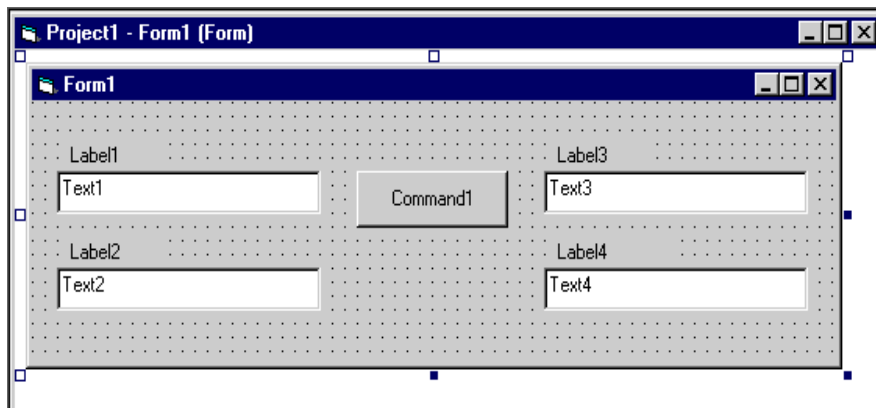


To create one or more controls on a form's surface, you select the control type that you want from the Toolbox window, click on the form, and drag the mouse cursor until the control has the size and shape you want. Alternatively, you can place a control on the form's surface by double-clicking its icon in the Toolbox: this action creates a control in the center of the form. Regardless of the method you follow, you can then move and resize the control on the form using the mouse.

To complete our Rectangle sample application, we need four TextBox controls—two for entering the rectangle's width and height and two for showing the resulting perimeter and area. Even if they aren't strictly required from an operational point of view, we also need four Label controls for clarifying the purpose of each TextBox control. Finally we add a

CommandButton control named *Evaluate* that starts the computation and shows the results.

Place these controls on the form, and then move and resize them as depicted in Figure 1-8. Don't worry too much if the controls aren't perfectly aligned because you can later move and resize them using the mouse or using the commands in the Format menu.



Setting Properties of Controls

Each control is characterized by a set of properties that define its behavior and appearance. For instance,

- Label controls expose a *Caption* property that corresponds to the character string displayed on the control itself, and a *BorderStyle* property that affects the appearance of a border around the label.
- The TextBox control's most important property is *Text*, which corresponds to the string of characters that appears within the control itself and that can be edited by the user.



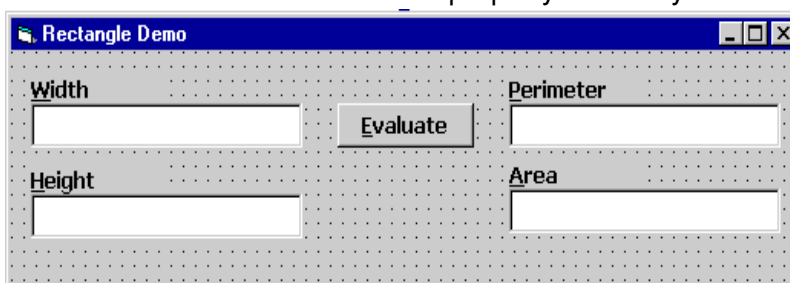
In all cases, you can modify one or more properties of a control by selecting the control in the form designer and then pressing F4 to show the Properties window. You can scroll through the contents of the Properties window until the property you're interested in becomes visible. You can then select it and enter a new value.

Using this procedure, you can modify the *Caption* property of all four Label controls to *&Width*,

&Height, *&Perimeter*, and *&Area*, respectively. You will note that the ampersand character doesn't appear on the control and that its effect is to underline the character that follows it. This operation actually creates a *hot key* and associates it with the control. When a control is associated with a hot key, the user can quickly move the focus to the control by pressing an Alt+x key combination, as you normally do within most Windows applications. Notice that only controls exposing a *Caption* property can be associated with a hot key. Such controls include the Label, Frame, CommandButton, OptionButton, and CheckBox.

Notice that once you have selected the *Caption* property for the first Label control, it stays selected when you then click on other controls. You can take advantage of this mechanism to change the *Caption* property of the CommandButton control to *&Evaluate* and the *Caption* property of the Form itself to *Rectangle Demo*, without having to select the *Caption* item in the Properties window each time. Note that ampersand characters within a form's caption don't have any special meaning.

As an exercise, let's change the font attributes used for the controls, which you do through the *Font* property. While you can perform this action on a



control-by-control basis, it's much easier to select the group of controls that you want to affect and then modify their properties in a single operation. To select multiple controls, you can click on each one of them while you press either the Shift

or the Ctrl key, or you can drag an imaginary rectangle around them.

When you select a group of controls and then press the F4 key, the Properties window displays only the properties that are common to all the selected controls. The only properties that are exposed by any control are *Left*, *Top*, *Width*, and *Height*. If you select a group of controls that display a string of characters, such as the TextBox, Label, and CommandButton controls in our Rectangle example, the *Font* property is also available and can therefore be selected. When you double-click on the *Font* item in the Properties window, a Font dialog box appears. Let's select a Tahoma font and set its size to 11 points.

Finally we must clear the *Text* property of each of the four TextBox controls so that the end user will find them empty when the program begins its execution. Oddly, when you select two or more TextBox controls, the *Text* property doesn't appear in the Properties window. Therefore, you must set the *Text* property to an empty string for each individual TextBox control on the form. To be honest, I don't know why this property is an exception to the rule stated earlier.

Naming Controls

One property that every control has and that's very important to Visual Basic programmers is the *Name* property. This is the string of characters that identifies the control in code. This property can't be an empty string, and you can't have two or more controls on a form with the same name. The special nature of this property is indirectly confirmed by the fact that it appears as (Name) in the Properties window, where the initial parenthesis serves to move it to the beginning of the property list.



When you create a control, Visual Basic assigns it a default name. For example, the first TextBox control that you place on the form is named *Text1*, the second one is named *Text2*, and so forth. Similarly,

the first Label control is named *Label1*, and the first CommandButton control is named *Command1*. This default naming scheme frees you from having to invent a new, unique name each time you create a control. Notice that the *Caption* property of Label and CommandButton controls, as well as the *Text* property of TextBox controls, initially reflect the control's *Name* property, but the two properties are independent of each other. In fact, you have just modified the *Caption* and *Text* properties of the controls in the Rectangle Demo form without affecting their *Name* properties.

Because the *Name* property identifies the control in code, it's a good habit to modify it so that it conveys the meaning of the control itself. This is as important as selecting meaningful names for your variables. In a sense, most controls on a form are special variables whose contents are entered directly by the user.

Microsoft suggests that you always use the same three-letter prefix for all the controls of a given class. The control classes and their recommended prefixes are shown in Table

Control Class	Prefix	Control Class	Prefix
CommandButton	cmd	Data	dat
TextBox	txt	HScrollBar	hsb
Label	lbl	VScrollBar	vsb
PictureBox	pic	DriveListBox	drv
OptionButton	opt	DirListBox	dir
CheckBox	chk	FileListBox	fil
ComboBox	cbo	Line	lin
ListBox	lst	Shape	shp
Timer	tmr	OLE	ole
Frame	fra	Form	frm

Adding Code

Up to this point, you have created and refined the user interface of your program and created an application that in principle can be run. But you don't have a useful application yet. To turn your pretty but useless program into your first working application, you need to add some code. More precisely, you have to add some code in the *Click* event of the *cmdEvaluate* control. This event fires when the user clicks on the Evaluate button.

To write code within the *Click* event, you just select the *cmdEvaluate* control and then press the F7 key, or right-click on it and then invoke the View Code command from the pop-up menu. Or you simply double-click on the control using the left mouse button. In all cases, the code editor window appears, with the flashing cursor located between the following two lines of code:

```
Private Sub cmdEvaluate_Click()
```

```
End Sub
```

Visual Basic has prepared the template of the *Click* event procedure for you, and you have to add one or more lines of code between the *Sub* and *End Sub* statements. In this simple program, you need to extract the values stored in the *txtWidth* and *txtHeight* controls, use them to compute the rectangle's perimeter and area, and assign the results to the *txtPerimeter* and *txtArea* controls respectively:

```
Private Sub cmdEvaluate_Click()  
    ' Declare two floating point variables.  
    Dim reWidth As Double, reHeight As Double  
    ' Extract values from input TextBox  
    controls.  
    reWidth = CDbI(txtWidth.Text)  
    reHeight = CDbI(txtHeight.Text)  
    ' Evaluate results and assign to output text  
    boxes.  
    txtPerimeter.Text = CStr((reWidth +
```

```
    reHeight) * 2)  
    txtArea.Text = CStr(reWidth * reHeight)  
End Sub
```

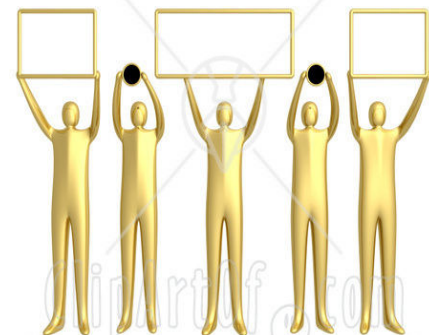
Note that you should always use the *Dim* statement to declare the variables you are going to use so that you can specify for them the most suitable data type. If you don't do that, Visual Basic will default them to the Variant data type. While this would be OK for this sample program, for most occasions you can make better and faster applications if you use variables of a more specific type. Moreover, you should add an *Option Explicit* statement at the very beginning of the code module so that Visual Basic will automatically trap any attempt to use a variable that isn't declared anywhere in the program code.

Running and Debugging the Program

You're finally ready to run this sample program. You can start its execution in several ways:

- By invoking the Start command from the Run menu.
- By clicking the corresponding icon on the toolbar.
- Or by pressing the F5 key.

In all cases, you'll see the form designer disappear and be replaced by the real form. You can enter any value in the leftmost TextBox controls and then click on the Evaluate to see the calculated perimeter and area in the rightmost controls. When you're finished, end the program by closing its main form.



Preprocessor Directives

By

Hafiz Fahad Hassan

What is Preprocessor?

The C preprocessor, often known as *cpp*, is a *macro processor* that is used automatically by the C compiler to transform your program before compilation. It is called a macro processor because it allows you to define *macros*, which are brief abbreviations for longer constructs.

The C preprocessor is intended to be used only with C, C++, and Objective-C source code. In the past, it has been abused as a general text processor. It will choke on input which does not obey C's lexical rules. For example, apostrophes will be interpreted as the beginning of character constants, and cause errors. Also, you cannot rely on it preserving characteristics of the input which are not significant to C-family languages. If a Makefile is preprocessed, all the hard tabs will be removed, and the Makefile will not work.

Ref:

<http://gcc.gnu.org/onlinedocs/cpp/Overview.html#Overview>

Another view of preprocessor according to Wikipedia is:

In computer science, a preprocessor is a program that processes its input data to produce output that is used as input to another program. The output is said to be a preprocessed form of the input data, which is often used by some subsequent programs like compilers. The amount and kind of processing done depends on the nature of the preprocessor; some preprocessors are only capable of performing relatively simple textual substitutions and macro expansions, while others have the power of fully-fledged programming languages.

A common example from computer programming is the processing performed on source code before the next step of compilation. In some computer languages (e.g., C) there is a phase of translation known as preprocessing.

Ref: <http://en.wikipedia.org/wiki/Preprocessor>

Wiki.Answers says:

As the name indicates the C PreProcessor is the preprocessor for C programming language and is used by the compiler automatically to transform the program before actual compilation. It processes the preprocessor directives (file inclusion, macro definition and conditional compilation) in the source code. It is also known as a macro processor.

Ref:

http://wiki.answers.com/Q/What_is_preprocessor_in_C

Another comprehensive definition says:

The preprocessor is used to modify your program according to the preprocessor directives in your source code. Preprocessor directives (such as *#define*) give the preprocessor specific instructions on how to modify your source code. The preprocessor reads in all of your include files and the source code you are compiling and creates a preprocessed version of your source code. This preprocessed version has all of its macros and constant symbols replaced by their corresponding code and value assignments. If your source code contains any conditional preprocessor directives (such as *#if*), the preprocessor evaluates the condition and modifies your source code accordingly.

The preprocessor contains many features that are powerful to use, such as creating macros, performing conditional compilation, inserting predefined environment variables into your code, and turning compiler features on and off. For the professional programmer, in-depth knowledge of the features of the preprocessor can be one of the keys to creating fast, efficient programs.

Ref:

http://www.geekinterview.com/question_details/3370

Now at this stage we are ready to learn what are preprocessor directives?

The preprocessor is a program that is invoked by the compiler to process code before compilation. Commands for that program, known as directives, are lines of the source file beginning with the character *#*, which distinguishes them from lines of source program text. The effect of each preprocessor directive is a change to the text of the source code, and the result is a new source code file, which does not contain the directives. The preprocessed source code, an intermediate file, must be a valid C or C++ program, because it becomes the input to the compiler.

Preprocessor directives consist of the following:

- **Macro definition directives**, which replace tokens in the current file with specified replacement tokens
- **File inclusion directives**, which imbed files within the current file
- **Conditional compilation directives**, which conditionally compile sections of the current file
- **Message generation directives**, which control the generation of diagnostic messages
- **IBM extension Assertion directives**, which specify attributes of the system the program is to run on
- **The null directive (#)**, which performs no action
- **Pragma directives**, which apply compiler-specific rules to specified sections of code

Preprocessor directives begin with the # token followed by a preprocessor keyword. The # token must appear as the first character that is not white space on a line. The # is not part of the directive name and can be separated from the name with white spaces.

A preprocessor directive ends at the new-line character unless the last character of the line is the \ (backslash) character. If the \ character appears as the last character in the preprocessor line, the preprocessor interprets the \ and the new-line character as a continuation marker. The preprocessor deletes the \ (and the following new-line character) and splices the physical source lines into continuous logical lines. White space is allowed between backslash and the end of line character or the physical end of record. However, this white space is usually not visible during editing.

Except for some #pragma directives, preprocessor directives can appear anywhere in a program.

To study preprocessor directives in depth, follow the link: <http://gcc.gnu.org/onlinedocs/cpp/>

Header Files

In computing, header files are a feature of some programming languages (most famously C and C++) that allows programmers to separate certain elements of a program's source code into reusable files. Header files commonly contain forward declarations of classes, subroutines, variables, and other identifiers. Programmers who wish to declare standardized identifiers in more than one source file can place such identifiers in a single header file, which other code can then include whenever the header contents are required. The C standard library and C++ standard library traditionally declare their standard functions in header files.

Newer compiled languages (such as Java, C#) do not use forward declarations; identifiers are recognized automatically from source files and read directly from dynamic library symbols. This means header files are not needed.

Ref: http://en.wikipedia.org/wiki/Header_file

Information that is needed by several different files or functions is collected into a header file. A header file contains C-language definitions and structures. Centralizing information into a header file facilitates the creation and update of programs. Because #include statements are used to insert header files into a C-language program, header files are often referred to as include files.

Header files define the following functions:

- Structures of certain files and subroutines
- Type definition (typedef) synonyms for data types
- System parameters or implementation characteristics
- Constants and macros that are substituted during the C language preprocessing phase.

By convention, the names of header files end with .h (dot h). The .h suffix is used by header files that are provided with the operating system; however, the suffix is not required for user-generated header files. Note: Several of the header files provided with the operating system end with .inc (include file).

Ref:

http://publib.boulder.ibm.com/infocenter/aix/v6r1/index.jsp?topic=/com.ibm.aix.files/doc/aixfiles/C_hapter_4.htm

Note: There is a term used above, i.e. **forward declaration**. It means a **declaration of an identifier** (denoting an entity such as a type, a variable, or a function) **for which the programmer has not yet given a complete definition**.

Another term, i.e. **forward reference** is sometimes used as a synonym of forward declaration. However, more often it is taken to refer to the **actual use of an entity before any declaration**.

An example of (valid) forward reference in C++:

```
class C {
public:
    void mutator(int x) { myValue = x; }
    int accessor() { return myValue; }
private:
    int myValue;
};
```

In this example, there are two references to `myValue` before it is declared. C++ generally prohibits forward references, but they are allowed in the special case of class members. Since the member function `accessor` cannot be compiled until the compiler knows the type of the member variable `myValue`, it is the compiler's responsibility to remember the definition of `accessor` until it sees `myValue`'s declaration.

Warning: Permitting forward references can greatly increase the complexity and memory requirements of a compiler, and generally prevents the compiler from being implemented in one pass.

Ref:

http://en.wikipedia.org/wiki/Forward_declaration

Header files serve two purposes.

- System header files declare the interfaces to parts of the operating system. You include them in your program to supply the definitions and declarations you need to invoke system calls and libraries.
- Your own header files contain declarations for interfaces between the source files of your program. Each time you have a group of related declarations and macro definitions all or most of which are needed in several different source files, it is a good idea to create a header file for them.

Including a header file produces the same results as copying the header file into each source file that

needs it. Such copying would be time-consuming and error-prone. With a header file, the related declarations appear in only one place. If they need to be changed, they can be changed in one place, and programs that include the header file will automatically use the new version when next recompiled. The header file eliminates the labor of finding and changing all the copies as well as the risk that a failure to find one copy will result in inconsistencies within a program.

In C, the usual convention is to give header files names that end with `.h`. It is most portable to use only letters, digits, dashes, and underscores in header file names, and at most one dot.

To know further about the following terms related to header files;

- Include Syntax
- Include Operation
- Search Path
- Once-Only Headers
- Alternatives to Wrapper `#ifndef`
- Computed Includes
- System Headers

Visit the following reference link.

Ref: <http://gcc.gnu.org/onlinedocs/cpp/Header-Files.html>

To study about all the standard C++ header files, visit the following link:

Ref: <http://msdn.microsoft.com/en-us/library/a7tkse1h%28VS.80%29.aspx>

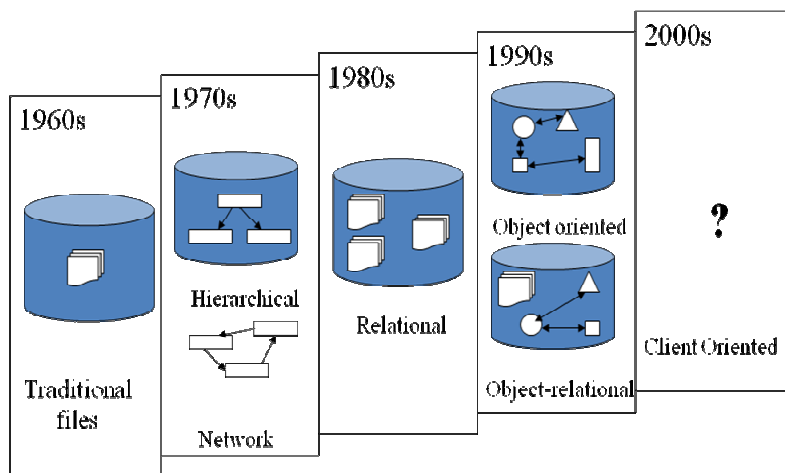
By

Hafiz Fahad Hassan & Madiha Qureshi

DATABASES

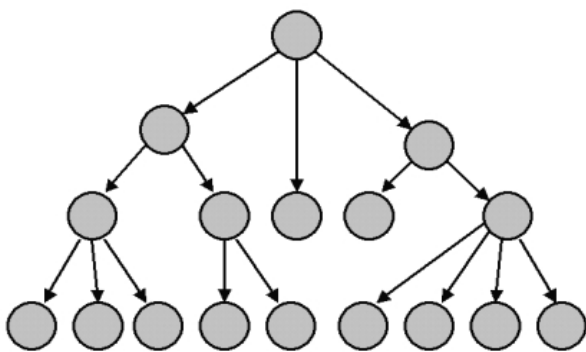
Evolution of Database Models:

In this article I will present the most common approaches on DBMSs and give a quick introduction to each of them, while trying to put them into the correct context of history. Then I will explain the relational database model in detail. First of all let's have a look on the history of database.



Hierarchical Database Model:

A hierarchical data model is a data model in which the data is organized into a tree-like structure. The structure allows repeating information using parent/child relationships: each parent can have many children but each child only has one parent. All attributes of a specific record are listed under an entity type.



This simple model is commonly known as the adjacency list model, and was introduced by Dr. Edgar F. Codd after initial criticisms surfaced that the relational model could not model hierarchical data.

Advantages of Hierarchical model:

- Adding and deleting records is easy.
- Fast data retrieval through higher level records.
- Multiple associations with like records in different files.
- Hierarchical model is simple to construct and operate on.
- Corresponds to a number of natural hierarchically organized domains- eg; assemblies in manufacturing, personnel organization in companies.
- Language is simple; uses constructs like GET, GET UNIQUE, GET NEXT, GET NEXT WITHIN PARENT etc.

Disadvantages of Hierarchical Model:

- Pointer path restrict access
- Each association requires repetitive data in other records.
- Pointers require large amount of computer storage.
- Navigational and procedural nature of processing.
- Database is visualized as a linear arrangement of records.
- Little scope for "query optimization".

Network Model :

The network model organizes data using two fundamental constructs, called **records and sets**.

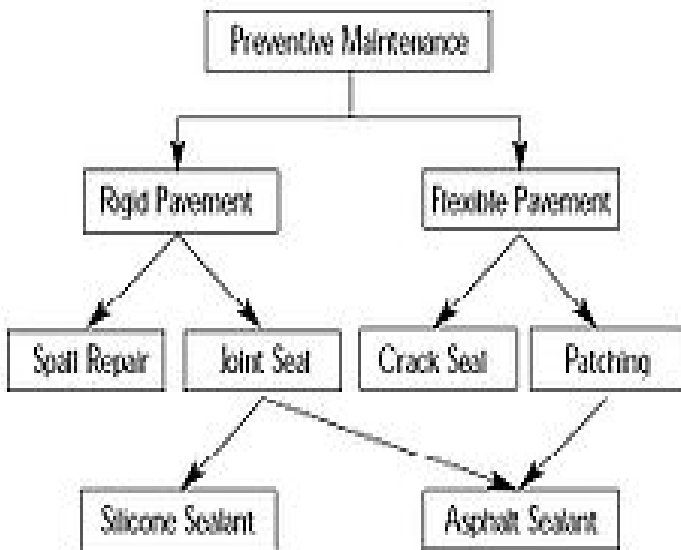
- Records contain fields.
- Sets define one-to-many relationships between records: one owner, many members.

A record may be an owner in any number of sets, and a member in any number of sets.

The network model is a variation on the hierarchical model, to the extent that it is built on the concept of multiple branches (lower-level structures) emanating from one or more nodes (higher-level structures), while the model differs from the hierarchical model in that branches can be connected to multiple

nodes. The network model is able to represent redundancy in data more efficiently than in the hierarchical model.

The operations of the network model are navigational in style: a program maintains a current position, and navigates from one record to another by following the relationships in which the record participates. Records can also be located by supplying key values.



Advantages:

- Provide very efficient "High-speed" retrieval.
- **Simplicity**
- The network model is conceptually simple and easy to design.
- **Ability to handle more relationship types**
- The network model can handle the one-to-many and many-to-many relationships.
- **Ease of data access**
- In the network database terminology, a relationship is a set. Each set comprises of two types of records.- an owner record and a member record, In a network model an application can access an owner record and all the member records within a set.
- **Data Integrity**
- In a network model, no member can exist without an owner. A user must therefore first define the owner record and then the member record. This ensures the integrity.

Data Independence

- The network model draws a clear line of demarcation between programs and the complex physical storage details. The application programs work independently of the data. Any changes made in the data characteristics do not affect the application program.



Disadvantages:

System complexity:

In a network model, data are accessed one record at a time. This makes it essential for the database designers, administrators, and programmers to be familiar with the internal data structures to gain access to the data. Therefore, a user friendly database management system cannot be created using the network model.

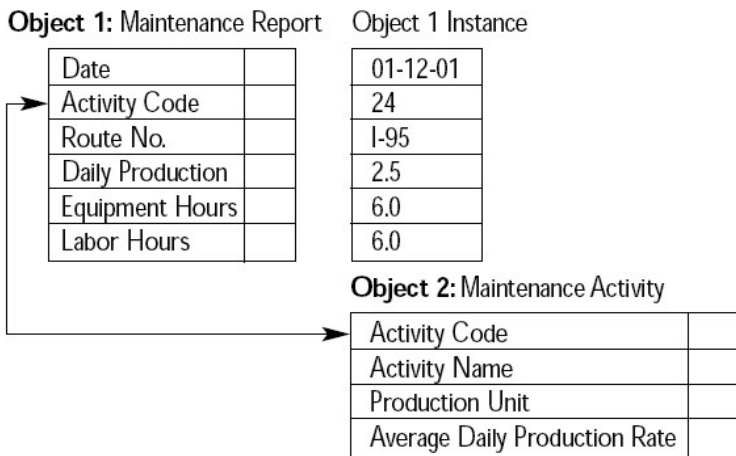
Lack of Structural independence:

Making structural modifications to the database is very difficult in the network database model as the data access method is navigational. Any changes made to the database structure require the application programs to be modified before they can access data. Though the network model achieves data independence, it still fails to achieve structural independence.

Object Oriented Database Model:

In recent years, the object-oriented paradigm has been applied to database technology, creating a new programming model known as object databases. These databases attempt to bring the database world and the application programming world closer together, in particular by ensuring that the database uses the same type system as the application program. This aims to avoid the overhead of converting information between its representation in the database (for example as rows in tables) and its representation in the application program (typically as objects). At the same time, object databases attempt to introduce the key ideas of object programming, such

as encapsulation and polymorphism, into the world of databases.



Advantages:

These are some of the major advantages of OOP.

- **Simplicity:**
- Software objects model real world objects, so the complexity is reduced and the program structure is very clear.
- **Modularity:**
- Each object forms a separate entity whose internal workings are decoupled from other parts of the system.
- **Modifiability:**
- It is easy to make minor changes in the data representation or the procedures in an OO program. Changes inside a class do not affect any other part of a program, since the only public interface that the external world has to a class is through the use of methods.
- **Extensibility:**
- Adding new features or responding to changing operating environments can be solved by introducing a few new objects and modifying some existing ones.
- **Maintainability:**
- Objects can be maintained separately, making locating and fixing problems easier.
- **Re-usability:**
- Objects can be reused in different programs.



Disadvantages:

- **Schema Changes:**
- In an OODBMS based application modifying the schema by creating, updating or modifying a persistent class typically means that changes have to be made to the other classes in the application that interact with instances of that class. This typically means that all schema changes in an OODBMS will involve a system wide recompile. Also updating all the instance objects within the database can take an extended period of time depending on the size of the database.
- **Language Dependence:**
- An OODBMS is typically tied to a specific language via a specific API. This means that data in an OODBMS is typically only accessible from a specific language using a specific API.
- **Lack of Ad-Hoc Queries**
- it is currently not possible to duplicate the semantics of joining two tables by "joining" two classes then there is a loss of flexibility with an OODBMS. Thus the queries that can be performed on the data in an OODBMS is highly dependent on the design of the system.



Relational Model:

The relational model was introduced by E.F. Codd in 1970 as a way to make database management systems more independent of any particular application. It is a mathematical model defined in terms of predicate logic and set theory. The products that are generally referred to as relational databases in fact implement a model that is only an approximation to the mathematical model.

Advantages:

- Structural independence
- Data access path is irrelevant to database design; change structure will not affect the database.
- Improved conceptual simplicity.
- Easier database design, implementation, management, and use.
- Ad hoc query capability with SQL (4GL is added).
- Powerful database management system.

- It allows 1:1, 1:M, M:N relationships.

Disadvantages:

- Substantial hardware and system software overhead.
- Poor design and implementation is made easy.
- May promote “islands of information” problems.

Explanation:

In relational database model designer focuses on logical representation rather than physical.

Basic Terms:

An understanding of relational databases requires an understanding of some of the basic terms.

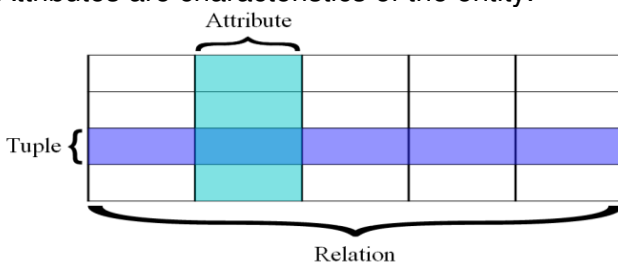
The table below summarizes some of the most important relational database terms and their SQL database equivalents.

Relational term	SQL equivalent
<u>relation</u> , base <u>relvar</u>	table
derived relvar	view, query result, result set
tuple	row
attribute	column

Entities and Attributes:

Entity is a person, place, event, or thing about which data is collected.

Attributes are characteristics of the entity.



Tables:

- Holds related entities or entity set.
- Also called relations.
- Comprised of rows and columns.

Table Characteristics:

- Two-dimensional structure with rows and columns.
- Rows (tuples) represent single entity.
- Columns represent attributes.

- Row/column intersection represents single value.
- Tables must have an attribute to uniquely identify each row.
- Column values all have same data format.

Data types:

- Number.
- Character.
- Date.
- Logical.

Attribute domain:

- Each column has range of values called attribute domain.

Order of the rows and columns is immaterial to the DBMS.

Relational Keys:

It is a set of one or more columns whose combined values are unique among all occurrences in a given table. A key is the relational means of specifying uniqueness.

Super Key:

An attribute (or combination of attributes) that uniquely identifies each entity in a table.

Candidate Key:

A minimal Super key. A Super key that does not contain subset of attributes that itself is a super key.

Primary Key:

A candidate key selected to uniquely identify all other attribute value in a given row. Cannot contain null entities.

Foreign Key:

An attribute (or combination of attribute) in one table whose values must either match with the primary key in another table or be null.

Secondary Key:

An attribute (or combination of attribute) that is strictly used for data retrieval purposes.
That's all for now!

By
Umer Asif

html

In this tutorial we will learn more about Basic formatting of the text in html pages. We will use different tags to make our text look better. So here are some common tags we can use.

Bold

To make the text **bold** we can use `..` tag. For example

Typing this code:

```
<b> This text is bold. </b>
```

Result is:

This text is bold.

Italics

We use `<i>..</i>` tag to write *italic* text. For example

Typing this code:

```
<i> This text is in Italics.</i>
```

Result is:

This text is in Italics.

Underlining

To underline the text we use `<u>..</u>` tag. For example

Typing this code:

```
<u>This code is underlined.</u>
```

Result is:

This code is underlined.

You can write tags in CAPITAL or small letters e.g `` or `` both mean the same.

You can also mix up different tags and combine them. You just have to surround the text with those sets of tags e.g. `<i>` this text is bold and Italic.`</i>`. But you have to be careful with the order of tags i.e the tag that starts first will end last and tag that starts last ends first. So it works on LAST IN FIRST OUT.

Line break

To insert a line break we use `
`. This is known as an “empty element” a tag that does not have an end tag. So where you need to start from new line insert it there.

Typing this code:

```
This is my text.<br>
```

This is new line.

Result is:

This is my text.

This is new line.

Heading

There is a special tag for specifying headings in HTML. There are 6 levels of headings in HTML ranging from h1 for the most important, to h6 for the least important. For example

Typing this code:

```
<h1>Heading 1</h1>
```

```
<h2>Heading 2</h2>
```

```
<h3>Heading 3</h3>
```

```
<h4>Heading 4</h4>
```

```
<h5>Heading 5</h5>
```

```
<h6>Heading 6</h6>
```

Result is:

Heading 1

Heading 2

Heading 3

Heading 4

Heading 5

Heading 6

Attributes

A tag on its own does something, but then you can add attributes to further define what the tag does. You will see many tags having attributes, they are very important part of HTML.

Basic structure of an attribute is as follows:

```
< tag attribute = "value"> text affected </tag>
```

A tag can have many attributes at a time but usually few attributes work for a certain tag. Note that in the end tag we did not write any attributes, so you just need to write simple end tag, attributes are already taken in it.

Lines

To insert a line across the page we can use `<hr>` tag which stands for 'horizontal rule'.

Just put that any where on your page, you don't need to add end tag and the text will stop and a big line will appear. Your rest of the text will begin after this line.

This tag can be manipulated through attributes too. For example it can have two attributes related to size of the line.

`<hr width = "100">` would result this

The value in the double quotes represents the width of the line in pixels here Or you can use percentage like for example

`<hr width = "80%">`

Above attribute will create a line that is 80% as wide as the available screen width.

By default a line drawn using `<hr>` will be equal to width of the screen.

To make the line strong you can use `<hr size = "4" noshade>` which will produce following line:

'noshade' is a special attribute that have no value and it is specific to 'hr'. it stops the line from having two shades of gray color in it.

Comments

Once your documents start getting filled with confusing tags and sections you're going to need to know which part is which. You use HTML comments to **add notes to your code** so that you can read it easily the next time you come back to edit it. The

code for a comment is a little different than for other tags:

```
<!-- Navigation starts here -->
```

Anything you put between the `<!-- -->` lines will be completely skipped by your browser. You can add in some hash marks (#) to make your comments stand out. When scrolling through the HTML code of a page you want your structure comments to jump out. Comments will be used later on in HTML to hide things from old browsers. They're very useful — use them and help yourself out.

Spaced Out

How you lay out your source code is a matter of taste in most regards. You can indent some tags off the left margin so they're easy to see, and skip lines after each paragraph. It doesn't matter much to your browser, which usually disregards spaces, tabs, blank lines and other 'white space' characters when it is displaying your pages. I should warn you that sometimes however, extra spacing characters will mess up something on you. It's not serious, but a line like the one below should be fixed:

```
<p><u>Some underlined text. </u></p>
```

The space character before the closing `</u>` tag will make the underline effect run on for longer than the sentence, which looks sloppy.

Links

As we have made our first page already, so to create link we will just need another page. It does not have to be a special page, any basic page will work or even you can use the same page by saving its copy with a different name. you just have to make sure that they are in the same folder.

Note: for convenience always use lowercase letters when naming your html files, images and folders because most web servers are usually case sensitive.

Link Structure

Like all tags link also follows a structure, and have a start tag and an end tag. For example:

```
<a href="theotherpage.html">The Link</a>
```

in above example 'a' stands for Anchor, which means a link; this is the tag that makes a link.

Href: means Hypertext Reference, this part is an attribute that has location of other page to which we are creating a link. In case other page is not located in the same folder as of first page then you will have to provide complete path of that page as its value, otherwise only name of the other page will work.

Whatever you put inside the link tags will become a link, coloured blue and underlined. When you rest your mouse on it your cursor will turn into a hand and the **URL** of the page will appear in your browser's status bar (at the bottom of the window).

Linking to Email Addresses

if you want to let people to email you by clicking this link you can use this code:

```
<a href="mailto:xyz@example.com"> mail me </a>
```

clicking the "mail me" will open the user's email program with your address in the To: box.

Linking to pictures

Linking to a picture file is practically the same as to a html file. Just include the name of the file, and do not forget to write the correct suffix — the format of your picture file e-g jpg or gif.

Linking to Files

You link to a file just like a picture. The only difference is that it won't open in a browser, but instead will download onto a specified place on the reader's hard drive. An example:

```
<a href="song.mp3">download the song  
(2.6MB mp3)</a>.
```

Here again your song should be in same folder as of your web page otherwise you will have to write complete path/address of it in ' href ' part.

The reason why we have to write full address will be discussed in coming lessons.



Link:

<http://www.facebook.com/pages/O-Tech/343926296775>

Economic Importance of Software:



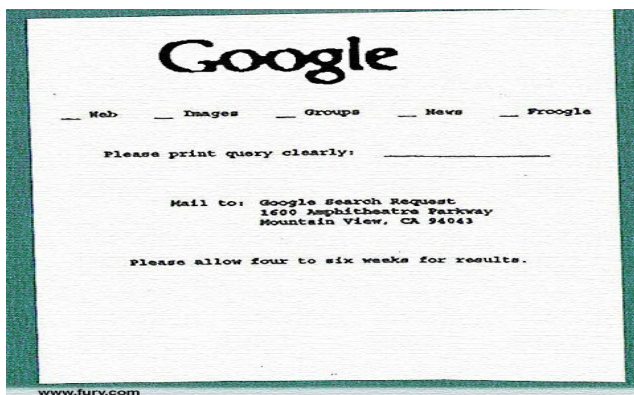
About this Article:

This article summarizes the history of software and importance of economics related to it, current state of the art and recent trends of economics in software engineering. It provides an overview of economic applicability to software engineering.

Advancement of Software w.r.t Economics

Earliest Software

The software industries started in the early 1960s, almost immediately after computers ('mainframes') were first sold in a more or less standardized way.



Universities and businesses began to use these computers and to seek out programs to do certain computing tasks. Many of these programs were

written in-house by full-time staff programmers. Some were distributed freely between users of a particular machine for no charge. But others were done on a commercial basis, and the very first standalone software firms started in the United States in 1959-1960. This can be considered as the first step when experts understood the importance of economics in the field of Software.

Introduction of Operating Systems

Pretty soon in early 1970s, the computer-makers started bundling operating systems software and programming environments with their machines. IBM, which delivered most of the computers at the time, became a household name in corporate businesses worldwide.



Enhancement of Technology

When Digital Equipment Corporation brought a relatively low-priced micro-computer to market, it brought computing within reach of many more companies and universities worldwide, and it spawned great innovation in terms of new, powerful programming languages and methodologies. New software was built for micro-computers, and others, including IBM, followed DEC's example quickly, resulting in the IBM AS400 amongst others.

Personal Computers

The industry expanded greatly with the rise of the personal computer in the mid-1970s, which brought computing to the desktop of the office worker. In subsequent years, it also created a growing market for games, applications, and utilities. DOS, Microsoft's first product, was the dominant operating system at the time.

Software Industry of 21st century

In the early years of the 21st century, another successful business model has arisen for hosted software, called software as a service, or SaaS; this was at least the third time this model had been

attempted. SaaS reduces the concerns about software piracy, since it can only be accessed through the Web, and by definition no client software is loaded onto the end user's PC, so this service has drastically reduced the worries of piracy which has eventually help to strengthen the economy related to the field of software.

Software Industry and its trend

There are several types of businesses in the software industry. Infrastructure software, including operating systems, middleware and databases, is made by companies such as Microsoft, IBM, Sybase, EMC, Oracle and VMWare. Enterprise software, the software that automates business processes in finance, production, logistics, sales and marketing, is made by Oracle, SAP AG, Sage and Infor. Security software is made by the likes of Symantec, Trend Micro and Kaspersky. Several industry-specific software makers are also among the largest software companies in the world: SunGard, making software for banks, BlackBoard making software for



schools, and companies like Qualcomm or CyberVision making software for telecom companies. Other companies do contract programming to develop unique software for one particular client company, or focus on configuring and customizing suites from large vendors such as SAP or Oracle.

Research by DataMonitor

According to market researcher DataMonitor, the size of the worldwide software industry in 2008 was

US\$ 303.8 billion, an increase of 6.5% compared to 2007. Americas account for 42.6% of the global software market's value. DataMonitor forecasts that in 2013, the global software market will have a value of US\$ 457 billion, an increase of 50.5% since 2008.

Join Us on facebook

The Origin

get inspired for the IT World

Join the group on facebook and stay connected with the origin team and fans.

Disclaimer: All the stuff is not self-created in this book. O-Tech contributors take help from any source, i.e. books, Internet or any other relevant person to make authenticated research to maximum extent.

Thank you for reading the **O-Tech** book. If you have any query, post it on The Origin website in the relevant class.

The Origin Official Website: www.TheOrigin.Co.Nr