

ગુજરાત રાજ્યના શિક્ષણવિભાગના પત્ર-કમાંડક
મશબ/1213/989/છ, તા.29-01-2014-થી મંજૂર

COMPUTER STUDIES

Standard 10



PLEDGE

India is my country.
All Indians are my brothers and sisters.
I love my country and I am proud of its rich and varied heritage.
I shall always strive to be worthy of it.
I shall respect my parents, teachers and all my elders and treat everyone with courtesy.
I pledge my devotion to my country and its people.
My happiness lies in their well-being and prosperity.

Price : ₹ 120.00



Gujarat State Board of School Textbooks
'Vidyayan', Sector 10-A, Gandhinagar-382010

© Gujarat State Board of School Textbooks, Gandhinagar
Copyright of this book is reserved by Gujarat State Board of School Textbooks.
No reproduction of this book in whole or in part, in any form is permitted without
written permission of the Director, Gujarat State Board of School Textbooks.

Subject Adviser

Prof. R. P. Soni

Authors

Dr Harshal Arolkar (Convener)
Dr Sanjay M. Shah
Dr Priti S. Sajja
Dr Kuntal P. Patel

Reviewers

Shri Girish S. Brahmbhatt
Shri Saket A. Dave
Shri Bimal K. Raval
Shri Mayuri I. Shah
Shri Ramaniklal L. Gilatar
Shri Rajanikant A. Pandya
Shri Pankaj R. Shukla
Shri Rajshree N. Padia
Shri Balkrishna Patel
Shri Sejal Trivedi

Co-ordinator

Shri Ashish H. Borisagar
(Subject Co-ordinator : Mathematics)

Preparation and Planning

Shri C. D. Pandya
(Dy. Director : Academic)

Lay-out and Planning

Shri Haresh S. Limbachiya
(Dy. Director : Production)

PREFACE

The Gujarat State Secondary and Higher Secondary Education Board has prepared new syllabi based on the open source operating system and compatible open source software tools for various topics of Computer Studies. These syllabi are sanctioned by the Government of Gujarat.

It is a matter of pleasure for the Gujarat State Board of School Textbooks to place this textbook of **Computer Studies** prepared according to the new syllabus before the students of **Standard 10**.

Before publishing the textbook, its manuscript has been fully reviewed by experts and teachers teaching at this level. Carrying out suggestions given by teachers and experts, we have made necessary changes in the manuscript and then have published the textbook.

The board has taken special care to ensure that this textbook is interesting, useful and free from errors. However, we welcome suggestions to enhance the quality of the textbook.

Dr Bharat Pandit

Director

Date : 1-12-2013

Dr Nitin Pethani

Executive President

Gandhinagar

First Edition : 2014

Published by : Bharat Pandit, Director, on behalf of Gujarat State Board of School Textbooks,
'Vidyayan', Sector 10-A, Gandhinagar

Printed by :

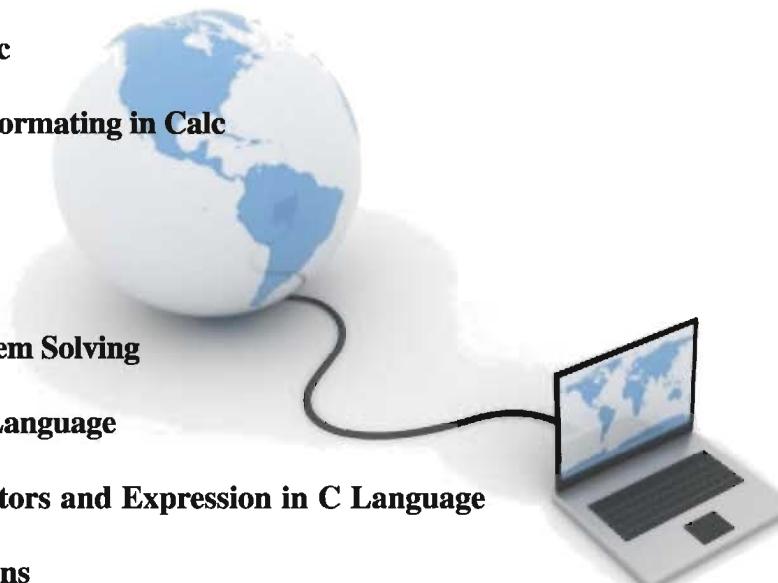
FUNDAMENTAL DUTIES

It shall be the duty of every citizen of India :

- (a) to abide by the Constitution and respect its ideals and institutions, the National Flag and the National Anthem;
 - (b) to cherish and follow the noble ideals which inspired our national struggle for freedom;
 - (c) to uphold and protect the sovereignty, unity and integrity of India;
 - (d) to defend the country and render national service when called upon to do so;
 - (e) to promote harmony and the spirit of common brotherhood amongst all the people of India transcending religious, linguistic and regional or sectional diversities; to renounce practices derogatory to the dignity of women;
 - (f) to value and preserve the rich heritage of our composite culture;
 - (g) to protect and improve the natural environment including forests, lakes, rivers and wild life, and to have compassion for living creatures;
 - (h) to develop the scientific temper, humanism and the spirit of inquiry and reform;
 - (i) to safeguard public property and to abjure violence;
 - (j) to strive towards excellence in all spheres of individual and collective activity so that the nation constantly rises to higher levels of endeavour and achievement.
-

CONTENTS

1. Introduction to HTML	1
2. Head and Body Sections	18
3. Handling Images in HTML	37
4. List and Table Handling in HTML	55
5. Introduction to Calc	81
6. Data Editing and Formating in Calc	97
7. Functions in Calc	127
8. Charts in Calc	153
9. Problem and Problem Solving	175
10. Introduction to C Language	192
11. Data Types, Operators and Expression in C Language	216
12. Using I/O Operations	239
13. Decision Structures	255
14. Loop Control Structures	270
15. Arrays	284
16. Function	296
● Appendix - I	309
● Appendix - II	310
● Appendix - III	311
● Appendix - IV	312



About This Textbook...

Dear Teachers,

With a mission to spread computer literacy on a fast track, the Gujarat Government has provided latest computer equipment to more than 6000 aided schools under the ICT@School program. As a new policy initiative, all the schools are given the Ubuntu (a variant of Linux) Operating System and other Open Source software packages so that schools can freely use and exchange the software without bothering about the licensing issues. Since earlier textbooks were largely based on proprietary software, there was a need to rewrite the textbooks based on new syllabus. This was also necessary in view of the fact that the 8th standard has been transferred to primary section. Therefore, new content has been provided for 9th to 12th standard in a phased manner based on the open source Operating System and compatible open source software tools for various topics of computer studies.

This textbook for 10th standard is the second in series for the subject of 'Computer Studies'. Students have already learnt Open Source Operating System-Ubuntu, Open Office Word Processor Writer, Presentation tool Impress and some concepts of Internet surfing, searching, email, file downloading etc. as well as security in 9th standard. In this textbook of 10th standard, they will be initially introduced to basic markup language HTML for presentation of web pages and contents. The Spreadsheet component of Open Office Calc is also discussed in detail. Later they will be learning the problem solving method and the C programming language. For easy comprehension of the language, numerous examples have been given along with explanation. We believe, learning programming will help students to develop logical thinking capabilities.

We hope this coverage will be useful to the students in 10th standard and you will enjoy teaching and conducting practicals using open source Ubuntu operating system and tools.

Dear Students,

Since you now are quite familiar with various definitions around computers, open source operating system Ubuntu as well as other operating systems available in public domain, Open Office components of word processing and presentation tools, Internet and its uses. It is now time to move further and learn advanced topics such as HTML, Spreadsheet Calc and C programming language.

In this textbook the essentials of markup language HTML are covered in chapters 1 to 4. In these chapters, full explanation is given for evolution of HTML, Structure of HTML document, necessary tags for creating documents with examples. Head and Body elements along with their attributes are again explained with relevant examples. Finally how to handle images, lists and tables within HTML code is presented and explained through various illustrations. Chapters 5 to 8 cover the Open Office Spreadsheet component Calc, Coverage of Calc includes Data Editing and Formatting, Calc Functions and generating Charts. Since elementary knowledge of problem solving and programming is very essential, Introduction to C Programming covering Data types, Operators, Expressions, I/O operations, Decision Structures, Arrays and Loop Control Structures and C Functions are covered in chapters 9 to 16. Numerous examples covering the described features have been presented at appropriate place with explanation.

It is expected that if you carefully study the text and practice the laboratory exercises, you will develop reasonable confidence in working with HTML, Calc and C Programming which are although elementary but quite essential topics in understanding computer applications.



Introduction to HTML

Working of the Internet and HTML

Computers are widely used in variety of applications. When computers are connected with each other they can share resources. Such group of connected computers is known as a computer network. The Internet is collection of such multiple computer networks, hence known as network of networks. On the platform of the Internet, a distributed information system exists, which is called World Wide Web, WWW or Web in short. The notion of the Web was conceived in 1991 by Tim Berners-Lee (figure 1.1), while consulting at CERN (a European Organization for Nuclear Research, <http://cern.web.cern.ch/CERN/>) in Switzerland. The Web is a repository of multimedia information on the internet platform. The web content in form of web pages is explored using browsers (special applications to retrieve and view web information). On these web pages, links are placed pointing towards different locations. These links are known as hyperlinks. Clicking on such hyperlink, one can redirect himself to an intended location. This operation is known as following the hyperlink. Any content such as text, picture, graphics, etc. can be embedded with such hyperlink.

The content and hyperlinks cannot be directly expressed on the Web. Hyper Text Markup Language (HTML), is needed to describe how a web page should be displayed by a web browser. Thus the HTML is considered as a language for describing web pages. The HTML is a documentation language to mark content of web pages such as heading, title, table, image, etc. It is machine independent and all Internet browsers accept the content written using HTML code.



Figure 1.1 : Tim Berners-Lee

HTML is a kind of markup language. A markup language is a set of tags that enables additional information (besides the content) on how to present the web content. HTML files are text files that contain additional formatting markup information in form of tags along with its content. The HTML is the most popular markup language; and it offers fixed set of tags. HTML is derived from of SGML (Standardized General Markup Language), which was developed by the International Organization for Standards (ISO) in 1986 to facilitate the sharing of machine-readable documents.

An HTML code is thus a combination of content to be displayed on a web page using browser and tags that helps in guiding the presentation of the content. Without such building block codes, it is impossible to display content on web pages. This makes HTML coding compulsory utility for web page creation, interpretation and presentation.

A Simple HTML Document

Let us create a simple web page that discusses about rainbow using HTML. The contents of the web page are shown in table 1.1.

RAINBOW

Rainbow consists of seven colours. These colours are Violet, Indigo, Blue, Green, Yellow, Orange and Red. They are also acronymed as VIBGYOR.

Rainbow is caused by reflection of light in water droplets in the Earth's atmosphere, resulting in a spectrum of light appearing in the sky. It takes the form of a multi coloured arc.

Table 1.1 : Text to be displayed on web page using HTML

HTML code to display the contents shown in table 1.1 is given in code listing 1.1.

```
<html>
  <head>
    <title> About Rainbow
    </title>
  </head>
  <body>
    <h1> RAINBOW </h1>
    <p> Rainbow consists of seven colours. These colours are
        Violet, Indigo, Blue, Green, Yellow, Orange and Red.
        They are also acronymed as VIBGYOR.
    </p>
    <p> Rainbow is caused by reflection of light in water
        droplets in the Earth's atmosphere, resulting in a
        spectrum of light appearing in the sky. It takes the
        form of a multi coloured arc.
    </p>
  </body>
</html>
```

Code Listing 1.1 : Sample HTML Code

Observe that besides the content about rainbow, code listing 1.1 also displays several sets of angular brackets with words or letters within them. These brackets and words inside them are known as tags.

A tag is made up of letters, words and numbers enclosed between a left and right angular bracket.

A combination of opening and closing tag along with some content between the two tags forms an element. An HTML element may be empty or can have some attributes to specify the additional formatting and publishing instructions. Figure 1.2 illustrates structure of tags and elements with an example.

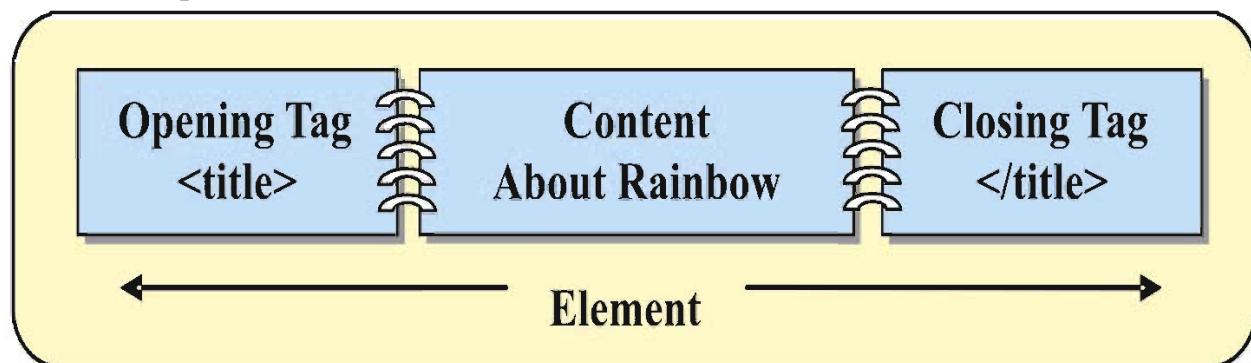


Figure 1.2 : An HTML element

Figure 1.2 indicates that **<title> About Rainbow </title>** forms an element of the HTML code. Observe that a closing tag contents are preceded with a forward slash (**</title>**). It marks the end of an element. Together, the pair of tags and the content within them forms an HTML element. The title tag here defines title of the web page generated by the HTML code.

Another example of such element is as follows :

<h1> RAINBOW </h1>

Here h1 tag refers to heading.

As stated above, content available between **<h1>** and **</h1>** is identified as heading and presented as heading. Similarly, content available between **<p>** and **</p>** is identified as paragraph and presented as paragraph text. The whole document is embedded between opening **<html>** and closing **</html>** tags.

To view how this page will look in a browser, follow the given steps :

Step 1 : Open gedit editor using Applications → Accessories → gedit. The gedit is a general purpose text editor for the GNOME (part of a project called GNU, free software by MIT) desktop environment, Mac OS X and Microsoft Windows. Alternatively you may use a shortcut available for the gedit editor at the header row of the screen.

Step 2 : Type the HTML contents of code listing given in table 1.2 in the empty gedit Window. Figure 1.3 show the look of the gedit editor after you have typed the code. Save the code as "p1.html", by selecting save option shown at the header row of the editor. Note that the HTML file can be saved with html or htm extensions. Figure 1.4 shows method to save the code.

The screenshot shows the gedit text editor interface. The title bar reads '*Unsaved Document 1 - gedit'. The menu bar includes File, Edit, View, Search, Tools, Documents, and Help. The toolbar contains icons for Open, Save, Undo, and various file operations. The main window displays the following HTML code:

```
<html>
  <head>
    <title> About Rainbow
    </title>
  </head>
  <body>
    <h1> RAINBOW
    </h1>
    <p> Rainbow consists of seven colours. These colours are Violet, Indigo, Blue, Green, Yellow, Orange and Red. They are also acronymed as VIBGYOR.
    </p>
    <p> Rainbow is caused by reflection of light in water droplets in the Earth's atmosphere, resulting in a spectrum of light appearing in the sky. It takes the form of a multi coloured arc.
    </p>
  </body>
</html>
```

At the bottom right of the editor window, there are buttons for Plain Text, Tab Width: 8, Ln 19, Col 1, and INS.

Figure 1.3 : The HTML code written in gedit editor

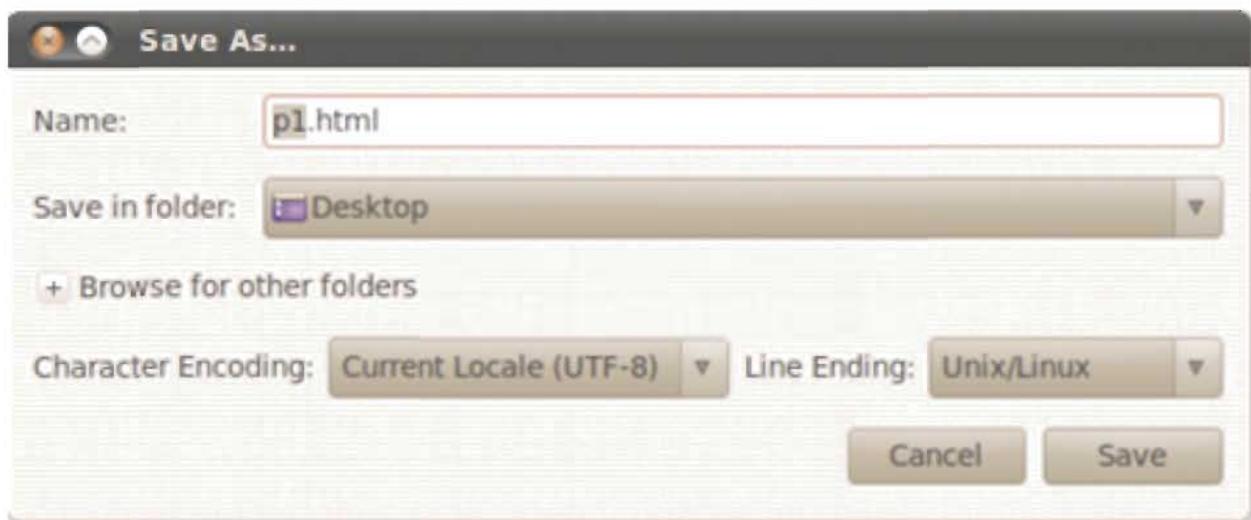


Figure 1.4 : Save As... dialog box

Step 3 : Open a browser such as Mozilla Firefox or any other browser that is installed on your computer. Select File → Open File, you will see an open file dialog box. Select the file that you want to open and click on Open button. Alternatively, you may double click on the file. Figure 1.5 shows the look and feel of the file when opened in Mozilla Firefox.



Figure 1.5 : The web page generated through the HTML code in a browser

It is to be noted that the tags used in HTML code are not case sensitive. The tags may be written in uppercase letters, lower case letters or mixture of upper and lower case letters. Further, multiple elements can be written in a single line. However, it is advisable to write each element in a new line with proper indentations for the sake of readability.

Structure of an HTML Document

HTML document is structured into two major parts. The first part is head section and second part is body section. They are also known as head element and body element. The head section contains information about the page such as title and description of the page. All these information should be embedded within the <head> and </head> tags.

The body element is embedded within the <body> and </body>. This is the content which can be seen within the browser. Both the head and body elements are embedded within the <html> and </html> tags.

HTML Title

The title of a web page is specified by the TITLE element, which should be placed in the head section of the document. It is to be noted that a document should have only one title element. It is used to identify the document content in a general way. Further, the content of title is not a part of the document text. Because of this, it should be simple text and cannot contain special commands such as hyperlinks. The title appears as a label of the window displaying the text. The title also holds a place in a browser's history or bookmark list. It is therefore recommended that title should be short. In the example HTML code given in listing of table 1.2, the title is "About Rainbow". It appears at the top of the windows displayed as shown in figure 1.5.

HTML Heading Style

HTML document generally begins with heading. In the example shown in code listing 1.1 heading style 1 (h1) tag is used. Observe the heading RAINBOW shown in the figure 1.5 to visualize how the heading style 1 looks. There are five more heading styles available in HTML. Heading can be

created in total six inbuilt sizes named as h1, h2, h3, h4, h5 and h6. These six levels of headings are described in the HTML code given in figure 1.6.

```
<html>
  <head>
    <title> About Rainbow
    </title>
  </head>
  <body>
    <h1> Rainbow </h1>
    <h2> Rainbow </h2>
    <h3> Rainbow </h3>
    <h4> Rainbow </h4>
    <h5> Rainbow </h5>
    <h6> Rainbow </h6>
  </body>
</html>
```

Figure 1.6 : HTML code for different levels of heading in HTML

The code is entered using the gedit editor. Save the code as "p2.html". When we see the code in a browser, it presents a view as shown in figure 1.7.

Most browsers display the contents of the <h1>, <h2>, and <h3> elements larger than the default size of text in the document. The content of the <h4> element is similar to the default content size. However, one can always redefine the sizes of these headings.

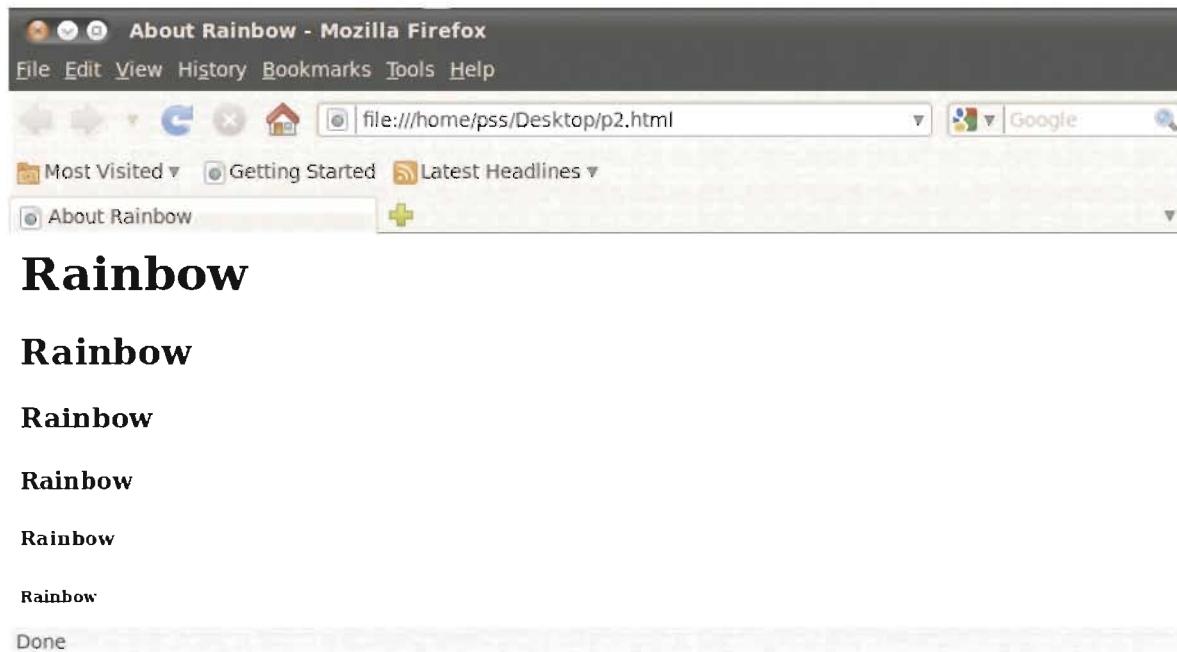


Figure 1.7 : Output of different levels of heading in HTML

Basic Text Formatting Tags

Besides the heading styles such as h1 to h6, there are more basic formatting tags available in HTML, which can be used in body section of an HTML code. Some of the useful tags are as follows :

Paragraph tags : <p> and </p>

The <p> tag structures the content into a paragraph. Each paragraph of text should go in between an opening <p> and closing </p> tag. Following are three valid examples of the same :

<p> This is first paragraph. </p>

<p> The second paragraph is here. This is about multiple colours of rainbows.

Though it is a temporary event it leaves a great impact on human mind. </p>

<p> Best of Luck! </p>

Consider the second example from the above listed examples. You may notice the multiple blanks and carriage returns (where enter key is pressed and text appears on a new line) in the second paragraph. It is to be noted that such white space will be considered as a single space. When an HTML code is displayed into a browser, the text will automatically take up the full width of the screen. If you resize the browser window, the browser will wrap the text onto new lines. Such white space management allows developer of HTML code to indent the code and add extra spaces to maintain readability of the code.

Line Break:
 or

Unlike all other tags, the
 element does not have an opening and closing tags.
 is an abbreviated form of break. Such tags are known as empty tags. Advanced versions like XHTML use
 tag. Note that there should be a space between 'br' and '/'.

The
 just pushes the coming text into next line. In case you need multiple lines, simply use multiple
 tags. Examples demonstrating use of the
 are as follows:

**First example

**

Second example

Using
 in paragraph forces compulsory break into the content and disturbs text wrapping while presentation and resizing the browser window.

Preformatted Text

Many a times we want to display text with multiple white spaces and in multiple lines without wanting to be changed it by the browser. For that, we may embed the content into pre-formatted tag set using <pre> and </pre>. Any text between the opening <pre> tag and the closing </pre> tag will preserve the formatting of the given content. Example is as follows :

```
<pre>
```

This is first line.

This is second line.

This is third line.

```
</pre>
```

In a browser the text appears along with given indentations. Also try following examples.

**Bold : and **

This tag is used to display given content into bold letters. The example can be given as follows :

```
<p> This is the <b> first </b> paragraph. </p>
```

Here the "first" word is displayed in bold letters.

Underline : <u> and </u>

This tag is used to display given content with underlined letters. The example can be given as follows :

```
<p> This is the <u> first </u> paragraph. </p>
```

Here the "first" word is displayed in underlined manner.

Italics : <i> and </i>

This tag is used to display given content into italics letters. The example can be given as follows :

```
<p> This is the <i> first </i> paragraph. </p>
```

Here the "first" word is displayed in italics letters.

Strike Through : <s> and </s>

The content of an <s> or <strike> element is displayed with a strikethrough a thin line through the text. Here the 's' is an abbreviated form of 'strike'. Example of presentation of strikethrough content is given as follows.

```
<p> This is the <s> cancelled </s> paragraph. </p>
```

Here the "cancelled" word is displayed in strikethrough manner.

Type writer font: <tt> and </tt>

The content of a <tt> element is written in typewriter type of fonts, which is also identified as mono-spaced font (like that of a teletype machine). Example is as follows :

```
<p> This is the <tt> first </tt> paragraph. </p>
```

Here the "first" word is displayed in mono-spaced fonts.

There are some other elements which are described in table 1.2.

Elements	Description
<small> and </small>	The content is displayed one font size smaller than the rest of the text surrounding it.
<big> and </big>	The content is displayed one font size bigger than the rest of the text surrounding it.
^{and}	The content is displayed in superscript.
_{and}	The content is displayed in subscript.
<acronym> and </acronym>	It defines the content as an acronym.
<dfn> and </dfn>	It defines a special term.
<q> and </q>	It defines a quote.

Table 1.2 : Some other formatting tags

Anchor Tag

When text is displayed within an HTML document, besides the content and format specification, some extra information or reference to other entity is needed. Many times further explanation is also required. Set of such words or text that appears in different colour (generally blue and underlined) are called hyperlink. A hyperlink is created using an [element, where the 'a' stands for an anchor. Let us modify the file "p1.html" as shown in code listing 1.2.](#)

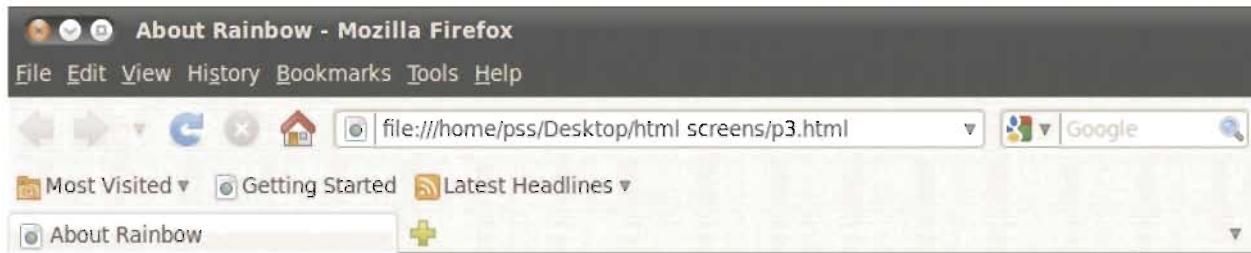
```

<html>
  <head>
    <title> About Rainbow
  </title>
  </head>
  <body>
    <h1> RAINBOW </h1>
    <p> Rainbow consists of seven colours. These colours are Violet, Indigo, Blue, Green, Yellow, Orange and Red. They are also acronymed as VIBGYOR.
    </p>
    <p> Rainbow is caused by reflection of light in water droplets in the Earth's atmosphere, resulting in a spectrum of light appearing in the sky. It takes the form of a multi coloured arc.
    </p>
    <p>
      <a href= "p4.html" > Click here to visit Theory of Rainbow. </a>
    </p>
  </body>
</html>

```

Code Listing 1.2 : HTML code for showing use of hyperlink

Save the file as "p3.html". Figure 1.8 shows the output of this code when viewed in browser.



RAINBOW

Rainbow consists of seven colours. These colours are Violet, Indigo, Blue, Green, Yellow, Orange and Red. They are also acronymed as VIBGYOR.

Rainbow is caused by reflection of light in water droplets in the Earth's atmosphere, resulting in a spectrum of light appearing in the sky. It takes the form of a multi coloured arc.

[Click here to visit Theory of Rainbow.](#)

Done

Figure 1.8 : Output of p3.html

When user clicks on the hyperlink, an intended file describing the 'Theory of Rainbow' must be opened. Let us create an HTML code for the file referred by the hyperlink as shown in code listing 1.3.

```
<html>
  <head>
    <title> Theory of Rainbow
    </title>
  </head>
  <body>
    <h1> How Rainbow Developed </h1>
    <p> Rainbow is caused by reflection of light in water droplets
        in the Earth's atmosphere, resulting in a spectrum of light
        appearing in the sky. It takes the form of multi coloured arc.
    </p>
  </body>
</html>
```

Code Listing 1.3 : HTML code linked to hyperlink

Save the code shown in code listing 1.3 as "p4.html". When you click on the linked (anchor) text (Click here to visit Theory of Rainbow.) shown in figure 1.8. The contents of p4.html will be loaded in the browser, see figure 1.9.



Figure 1.9 : Contents of p4.html

The hyperlink is a way to link two HTML documents by creating a hyper text in a document and giving reference of the other document to it. A website, which is a collection of many web pages, manages links through such hyperlink management. Here it is to be noted that, the web pages we have created are just presenting given content in a formatted way as we have specified them into the HTML code. Such web page contains only static (fixed) information, hence known as static web pages.

Absolute or Relative Address

Observe the line Click here to visit Theory of Rainbow. shown in code listing 1.2.

Instead of giving a full address such as <http://www.somedomain.com/p4.html>, we have given only the file name "p4.html". Giving just a file name will work only when you have the calling file (p3.html, also called parent file) and called file (p4.html, also called referred file) in the same directory. The location of the called file is relative to the calling file. Hence it is known as a relative address. While the complete address is known as an absolute address. If the referred file is located one directory above, we may prefix .. to the filename.

Note :

If no path is provided, the browser will understand that the referred file is located in the same directory where the parent file is stored.

Attributes to the Tags

To specify more information along with tags, additional attribute accompany the tags. In other words, attributes tell more about the elements. Attributes always appear on the opening tags of the elements that carry them. An attribute is made up of two parts. The first part is a name and the second part is a value.

The name of an attribute indicates the property to be set. In case of <a> tag demonstrated in table 1.5 has name href. The value is a value to be set to the property. In case of the href, the value is p4.html (the reference to the link). The values should be in double quotation marks. Between the name and the value there should be an equal (=) sign. See next section for example showing how to define a tag with an attribute.

Align Attribute

The align attribute indicates whether the heading appears to the left, center, or right of the page. By default, the content is aligned to the left of the page. It can take three values as follows.

Left : The content is aligned to the left of the page.

Right : The content is aligned to the right of the page.

Center : The content is aligned to the center of the page.

Following are some examples demonstrating use of the align attributes :

<p align="right"> This content will be displayed in right aligned form </p>

<p align="center"> This content will be displayed in center position of the page </p>

There are some attributes which can appear along with every tag. Such attributes are called universal attributes. Align is such universal attribute. Being a universal attribute, the align attribute can also go with heading as shown below.

<h1 align="center"> Centered Heading </h1>

When content given in a paragraph is aligned, some spaces are automatically added for adjustment. The spaces which are inserted automatically are known as soft spaces. If users himself (manually) inserts some spaces, such hard spaces will be automatically deleted unless the content is written using <pre> and </pre> tag pairs.

Working with other Editor - SciTE

SciTE is a text editor based on a free source code editing component called Scintilla [<http://www.scintilla.org>]. It comes with complete source code and a license that permits use in any free project or commercial product. The interface of the SciTE is shown in figure 1.10.

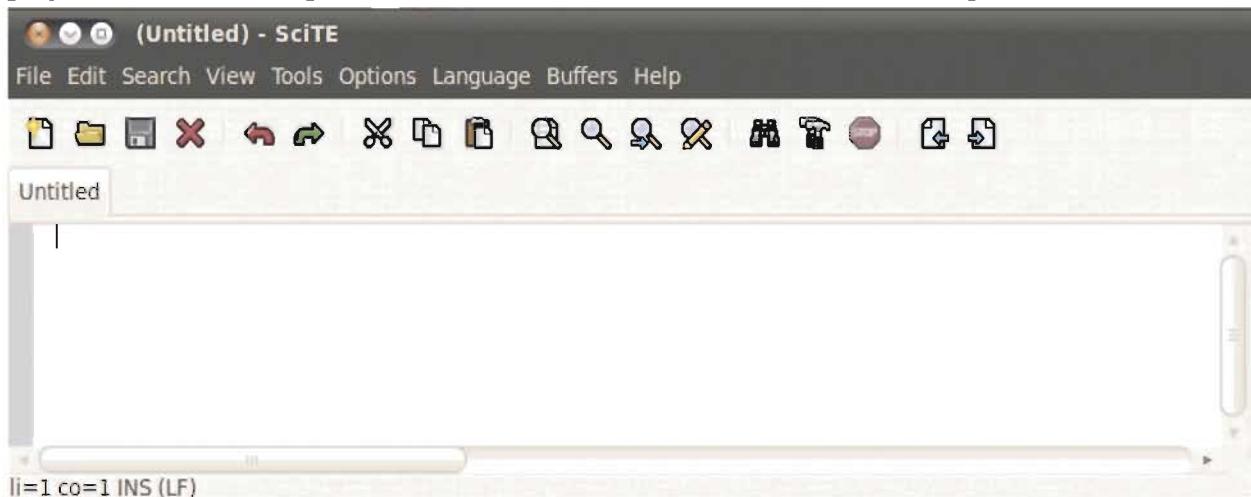
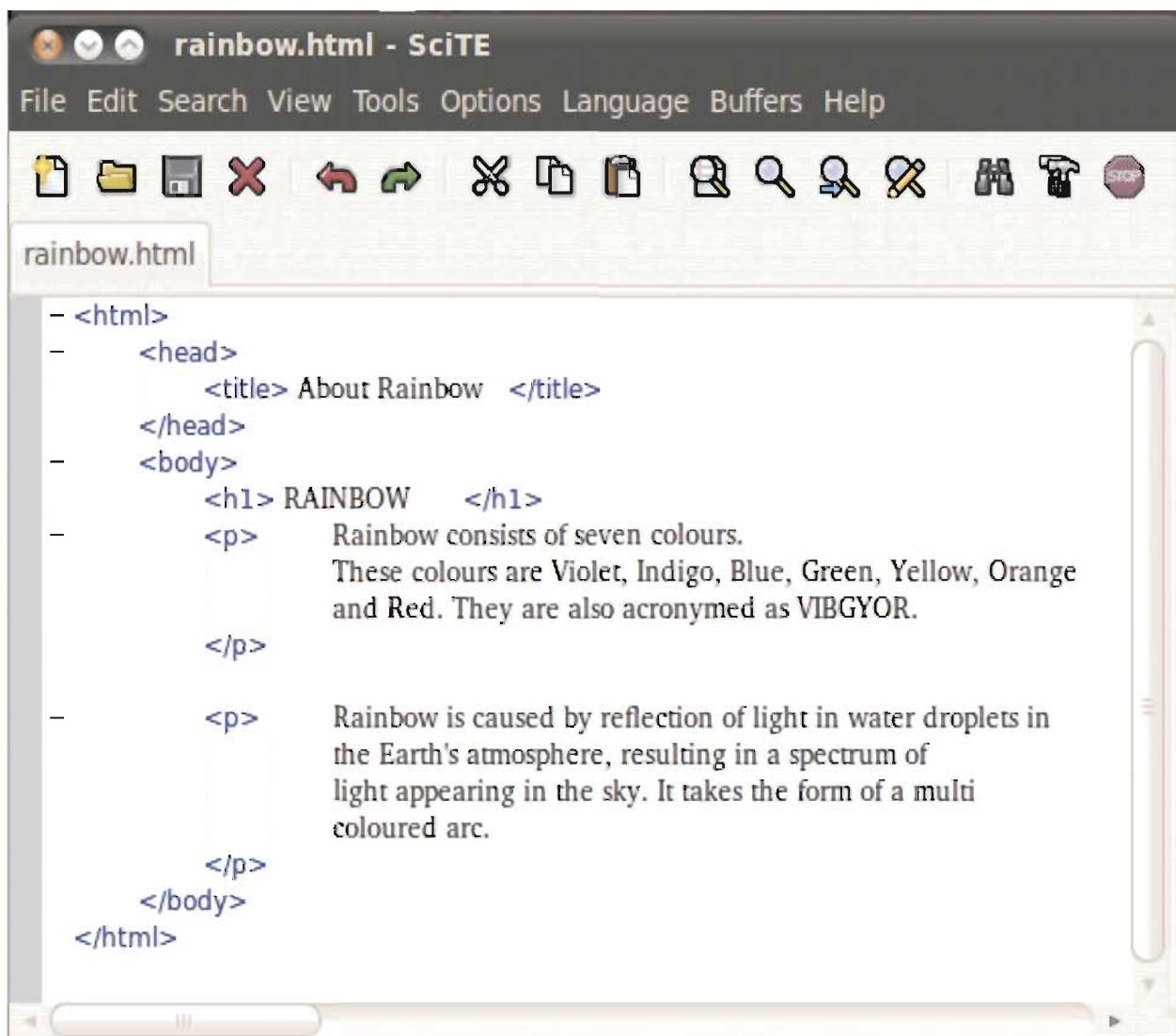


Figure 1.10 : Interface of the SciTE editor

The HTML code written above that prints information about rainbow (See example in figure 1.3) can be written in the SciTE editor as shown in figure 1.11.



The screenshot shows the SciTE editor window with the title bar "rainbow.html - SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, and Help. Below the menu is a toolbar with various icons for file operations like new, open, save, cut, copy, paste, find, and search. The main area displays the following HTML code:

```
- <html>
-   <head>
-     <title> About Rainbow </title>
-   </head>
-   <body>
-     <h1> RAINBOW </h1>
-     <p> Rainbow consists of seven colours.  
These colours are Violet, Indigo, Blue, Green, Yellow, Orange  
and Red. They are also acronymed as VIBGYOR.  
</p>
-
-     <p> Rainbow is caused by reflection of light in water droplets in  
the Earth's atmosphere, resulting in a spectrum of  
light appearing in the sky. It takes the form of a multi  
coloured arc.  
</p>
-   </body>
</html>
```

Figure 1.11 : HTML code in SciTE editor

Notice the menu bar items at the top of the screen shown in the figure 1.11. The menu items include basic facilities for file operations, editing facilities, searching, viewing, and other tools, options, etc. Next line beneath the menu items represents some icons for basic operations such as new, open, save, edit, search, etc. You may open any existing code in the SciTE editor as follows.

Step 1 : Locate the file you want to open.

Step 2 : Right click on it and choose Open With. You will see interface as shown in figure 1.12.

Step 3 : Select SciTE Text Editor. It will open the file in the SciTE editor. You may note the indicators for indentations and colour of tags shown by the SciTE. Having such indentations and tags in different colour separates the content from the tags and increases ease of reading the code.

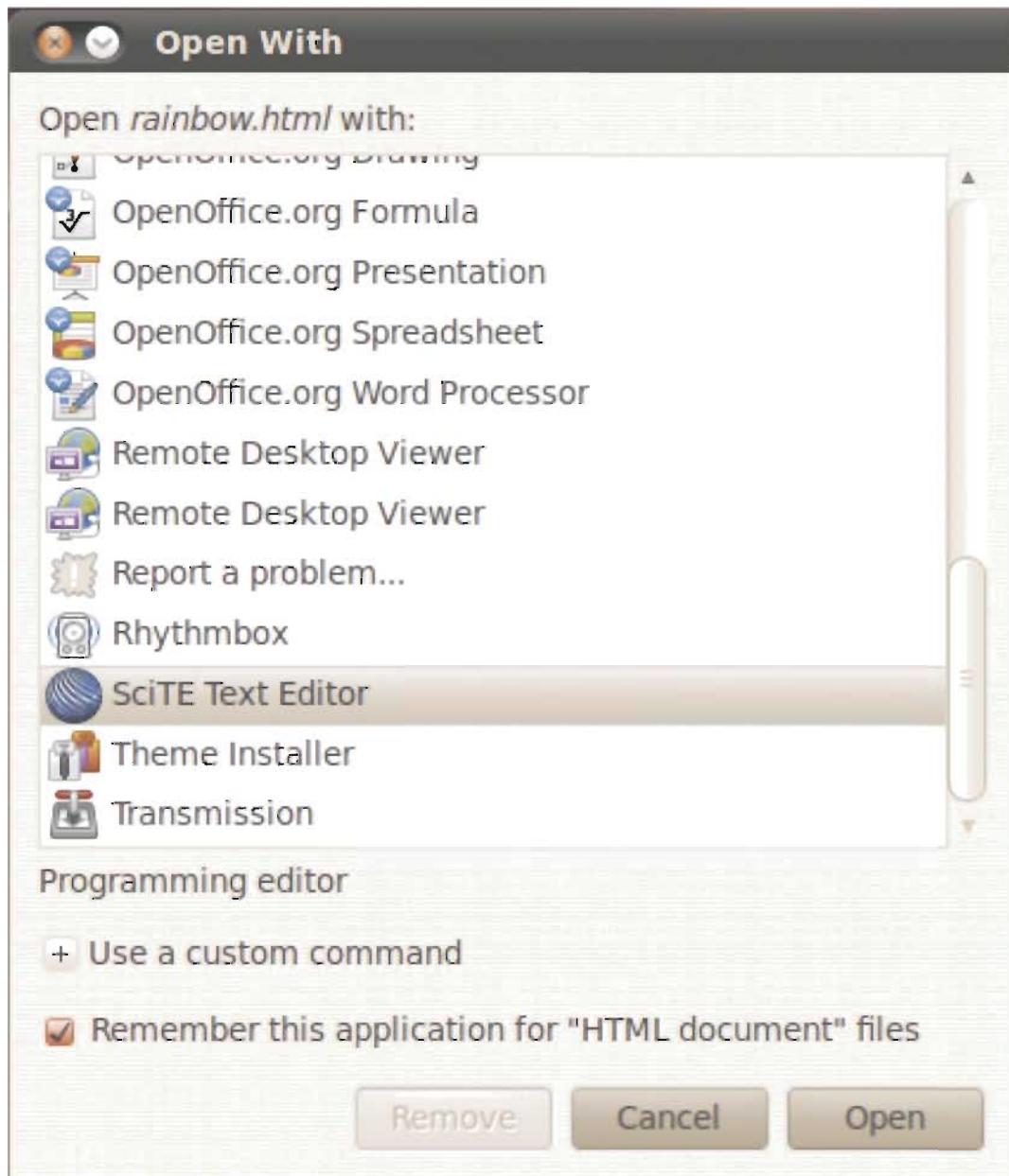


Figure 1.12 : The Open With dialog box

Step 4 : To view the html output in browser select Tools → Go or press F5.

Current Version of HTML

During the process of its evolution the HTML kept its focus on its purpose so that the composing and publishing content remains effective. With the progress of time, more and more functionalities were added to the HTML so that it can be useful and compatible with new browsers, new technologies and ever increasing developer demands. Latest version of the HTML is called HTML 5.0 and is becoming popular now.

Tags Covered in this Chapter

In this chapter we have discussed the tags mentioned in table 1.3.

Tag	Description
<a href> ... 	Anchors parent file to the referred file through hot text (link)
 ..	Displays text in bold fonts.
<body>...</body>	Defines body of the HTML document. Appears within the <html> tag pair.
 or 	Defines line break. It is an empty singular tag.
<h1>...</h1>	Defines a first level heading.
<h2>...</h2>	Defines a second level heading.
<h3>...</h3>	Defines a third level heading.
<h4>...</h4>	Defines a fourth level heading.
<h5>...</h5>	Defines a fifth level heading.
<h6>...</h6>	Defines a sixth level heading.
<head>...</head>	Defines the head section of an HTML document. Appears within <html> tag pair.
<html>...</html>	Covers the entire HTML document.
<i> ..</i>	Displays text in italics fonts.
<p>...</p>	Defines a paragraph
<pre>...</pre>	Displays preformatted text.
<s> ..</s>	Displays text in strikethrough manner.
<title>...</title>	Defines title of the document. Appears within the <head> tag pair.
<tt> ..</tt>	Displays text in typewriter fonts.
<u> ..</u>	Displays text in underlined fonts.

Table 1.3 : HTML Tags Covered in Chapter 1

Summary

In this chapter we learnt how to design a web page using HTML. HTML is a fundamental utility that describes how the web content is composed, published and retrieved using web browsers. In this chapter, two important components of the structure of an HTML document such as head and body are described with some basic tags. Using the information you can create simple web pages. You may also link multiple web pages created using the anchor tag discussed in this chapter. Besides providing fundamental concepts of the HTML and history of the HTML; the chapter also provides information about editors such as gedit and SciTE to build HTML documents.

EXERCISE

LABORATORY EXERCISE

1. Develop a web page that provides introductory information about your school. Give heading of this page as 'My School'. Use necessary formatting and presenting tags.
 2. Develop a web page that introduces your class. Include information such as your class teacher, other course teachers and subjects you are learning. Give heading of this page as 'My Class'. Use necessary formatting and presenting tags.
 3. Modify the web page created in question 1 of this exercise; to create a reference to another web page you have created in question 2 of this exercise. Set hot text in such a way that, when it is clicked, from the 'My School' page it will jump to the 'My Class' page.





Head and Body Sections

Head Section

An HTML document is divided into two sections called Head and Body. The head section of the HTML document provides necessary information about the HTML document. The content of the head section is included within `<head>` and `</head>` tags. Combination of these head tags and content between these is known as a head element.

The first obvious thing that can be included into the Head section is the title. It is to be noted that the title will not be displayed as a content of the web page. It is displayed as a title of the browser window that shows the web page.

It is to be noted that if a title is not specified, most browsers display the URL path or file name. Further, the title tag must have its closing pair. Failing to use the closing title tag, the whole content will be considered as title and it may be possible that the HTML document may not be loaded into the browser window.

The head section also contains additional information about the content and the HTML document. The tag that provides additional information is known as meta-tag. Meta-tags are used to store information usually relevant to browsers and search engines. Addition of appropriate meta-tags describes nature of the web page precisely and makes it easy for a search engine to search the web page efficiently.

Description

Most search engines will display the description when they put the results of a search to the users. In absence of such description, the search engine will display only first few words. An example of the description attribute of a meta-tag is as follows :

```
<meta name="DESCRIPTION" content="About rainbow and its colours">
```

Keywords

Keywords provided in this tag will be used by the search engine. Names of important characteristics, objective of the web page and important topics may be enlisted as keywords. Example of the meta-tag attribute that describes keywords is as follows.

```
<meta name="KEYWORDS" content="Rainbow, VIBGYOR">
```

Author

The following meta-tag attribute provides information about the author of the web page.

```
<meta name="AUTHOR" content="M K Gandhi">
```

Comments

Comments allow you to provide additional information in the HTML code. The comments are not meant to be displayed. That is no one can see the comments unless they look at the HTML source code. To add a comment we use '<!--' and '-->' tags. The '<!--' tag represents beginning of comment while the '-->' tag represents end of comment. Comments are also referred as prologue. One example of comment is <!-- This is a comment -->.

Other meta-tags

There are some meta-tags that tell the web page to load a specific URL after some seconds or tags telling it that a certain page should not be cached. The following example refreshes the web page (by reloading it) after every 5 seconds :

```
<meta http-equiv="REFRESH" content="5">
```

Following example refreshes the content of the given URL, <http://test.com/> every 5 seconds :

```
<meta http-equiv="refresh" content="5; URL='http://test.com/'>
```

HTML base

The <base> tag specifies the base URL/target for all relative URLs in a page.

```
<head>
<base href="http://test.com/ " >
</head>
```

Table 2.1 summarizes important HTML head elements.

Tag	Description
<head>	Defines information about the document.
<title>	Defines the title of a document.
<base>	Defines a default address or a default target for all links on a page.
<link>	Defines the relationship between a document and an external resource.
<meta>	Defines metadata about an HTML document.
<script>	Defines a client side script.
<style>	Defines style information for a document.

Table 2.1 : Head elements of HTML document

Consider the HTML code as shown in figure 2.1 to experiment with the head elements.

The screenshot shows a code editor window with a dark header bar containing 'File', 'Edit', 'Search', 'View', 'Tools', 'Options', 'Language', 'Buffers', and 'Help'. Below the header is a toolbar with various icons. The main area displays an HTML file named '2_2.html'. The code is as follows:

```
- <html>
-   <head>
-     <meta name="description" content="About rainbow and its colours">
-     <meta name="keywords" content="Rainbow, VIBGYOR">
-     <meta name="author" content="M K Gandhi">
-     <!-- This is a comment -->
-   </head>
- 
-   <body>
-     <p> <h1>Nothing will be printed but this line... </p>
-   </body>
- 
</html>
```

In the status bar at the bottom left, it says 'li=22 co=1 INS (CR+LF)'.

Figure 2.1 : Experimenting head elements of HTML

Implement the above code and see its output in a browser. It will look similar to figure 2.2.



Figure 2.2 : Output of the HTML code shown in figure 2.1

Body Section

The body section of an HTML document is written between `<body>` and `</body>` tags. The HTML body element acts as a container of the items which have to be displayed within a browser.

In previous chapter we have seen some tags related to heading, paragraph and formatting text that can appear in a body section. These tags usually require some content between them. There are some tags, which do not require any content; and hence are called empty tags. Example of such empty tag is `
`. Let us see some more tags and attributes used in the body section of HTML document.

Background Image

So far, we have seen HTML code that presents content on simple white background. To present the content on decorative and colourful background, we can use the background attribute with body tag. For the text background, we may use any image available with us in our computer. The set a background of HTML page we use the following tag:

```
<body background = "rainbow.jpg">
```

Here the term background is an attribute. The attribute has to be provided with a value. This value is a name of a file (normally an image file) that we want to display as background. In our case it is rainbow.jpg. We can provide any valid image formats like jpg, bmp, png, and tiff.

Figure 2.3 : HTML code to add a background image

Let us write HTML code as shown in figure 2.3 that sets background with an image called rainbow.jpg. Note that you may replace this file name with an image file name of your choice.

When we use a file within the HTML code, we have to be sure that the filename (along with the right path) and file extension are specified correctly. In case there is mistake in specifying the file name nothing will be visible in the background of the page.

Write the code as shown in figure 2.3 and save it with your desired file name (here it has been saved as e7.html). View the code in browser. Figure 2.4 illustrates the result of adding background image.



Figure 2.4 : Adding an image as a background

Many times, the background image is much more attractive than its content. We must take care that the background image must not take reader's focus from the content. The background image must be sober and adding an appeal to the content, instead of overriding it. Further, if the background image is too big, it may require lot of time to load the page in browser. Note that if the image used for the background is smaller in size than the screen size, the image may be replicated until it fills the entire screen. Generally the background image will scroll when you scroll down the page, unless you have set it to be fixed as follows:

```
<body background="rainbow.gif" bgproperties="fixed">
```

Background Colour

We can make the web page attractive by using background colour instead of using a background image. Consider if we want a background with yellow colour; following tag will work.

```
<body bgcolor="#FFFF00">
```

Here bgcolor is an attribute defining background colour. Note that we can also set both background image and colour together as shown:

```
<body background="rainbow.jpg" bgcolor="#FFFF00">
```

In such cases the background colour will be displayed till image is completely loaded in the browser. This effect will be generally visible on slow computers; on fast computers we may not be able to see any effect.

Any colour in an electronic media (such as television and computer screens) is considered as a combination of three basic colours namely red, green and blue. They are also acronymed as RGB. Recall, in your childhood, you were mixing two or more colours to get a new colour! Often you may have mixed blue and red colours to get purple colour !

Colours in computer are coded as degree from 00 to FF in hexadecimal. Hence, to represent a colour we need to build a code representing some red, some green and some blue (RGB) colour making it six digit code. That is, two digits for each colour. The string that defines the red colour is "FF0000". Here the red colour is assigned value FF (hex equivalent of decimal value 255), green is assigned a value 00, and blue is assigned a value 00. All these three sets of digits have ranges from 00 to FF (that is from 0 to up to 255; making total of 256 values). The combination of Red, Green, and Blue values from 0 to 255, gives more than 16 million different colors (256 x 256 x 256). By mentioning colour strings in hexadecimal format, we can specify various colours. Some examples of colours are shown in figure 2.5 :

FFFFFF →	White colour	→ 
FF0000 →	Red colour	→ 
FFFF00 →	Yellow colour	→ 
000000 →	Black colour	→ 
FF00FF →	Pink/Magenta colour	→ 
0000FF →	Blue colour	→ 
00FF00 →	Green colour	→ 

Figure 2.5 : Some examples of colour

Alternatively you may use the colour names also as shown below.

```
<body background="rainbow.jpg" bgcolor="Green">
<body background="rainbow.jpg" bgcolor="Chocolate">
```

You may produce various colourful backgrounds by changing just the colour string. Try the HTML code shown in code listing 2.1.

```
<html>
  <body bgcolor="#FF00FF">
    <h1>Everything is pink here, if you call this pink .....</h1>
  </body>
</html>
```

Code Listing 2.1 : HTML code for adding background

When viewed in the browser it will look similar to figure 2.6.

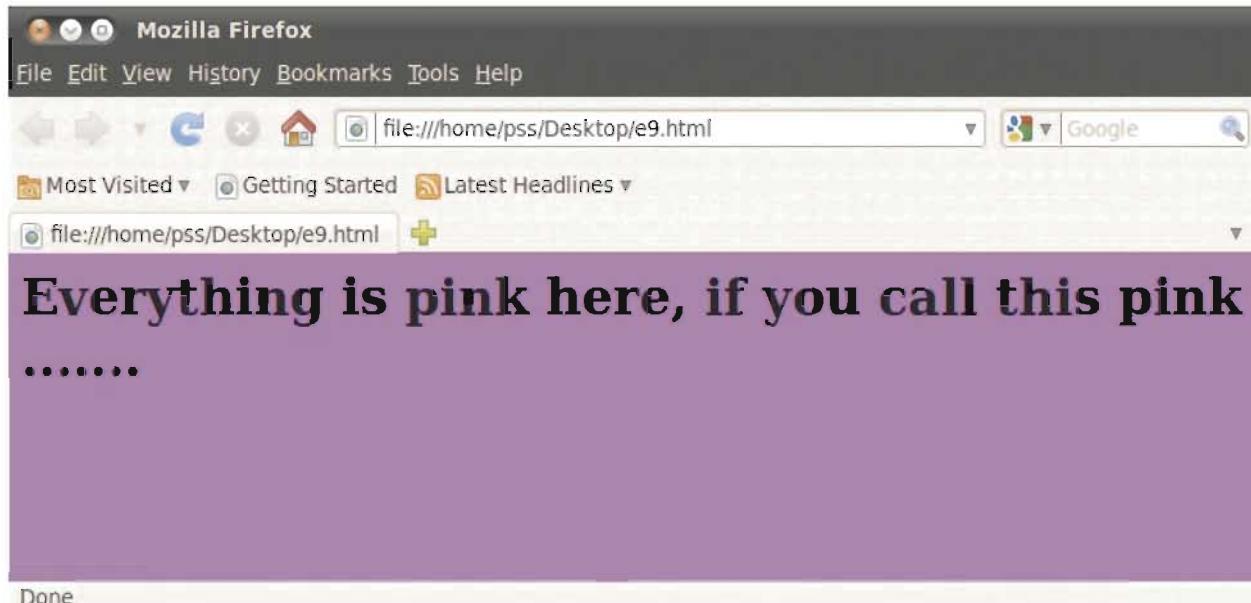


Figure 2.6 : Use of background colour

Text Colour

As we have defined background colour and images using attributes along with the body tag, we can also define text colour with attribute text along with the body tag. Assume that we write the following example.

```
<body text="FF0000" >
```

The use of text attribute with value "FF0000" will set the text colour as red. Suppose, you want to set background colour as pink and text colour to be yellow, you can do this in single instruction as follows.

```
<body bgcolor="FF00FF" text="FFFF00">
```

Link Colour

Generally you might have seen a blue coloured text pointing to different locations/pages within a web page being displayed. When such a text is clicked, it takes you to another page/resource. When you come back to the original page, you might observe that the colour of this text changes. As you have visited the link, it is identified by the web page as a 'visited link'. Such a visited link is identified as vlink. While the link that is being visited is called an active link. Active link is identified as alink. To distinguish visited links and active links, different colours are used. See the example below:

```
<body alink="#00FF00">  
<body vlink="#FF0000">
```

Alternatively, you may write

```
<body vlink="pink">
```

The alink attribute used in the above example sets the colour for active links within the document. Active link represent the state of a link as it is being clicked. The vlink attribute used in the above example sets the colour for hyperlinks within the document that have already been visited.

Horizontal Line

The `<hr />` element displays a horizontal line across the page. It is an empty element like `
` element. This is used to separate content by breaking it into sections. It is also called horizontal rule. Consider example shown in figure 2.7.



The screenshot shows the SciTE editor interface with the title bar "e10.html - SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, and Help. A single buffer tab labeled "1 e10.html" is open. The code in the buffer is:

```
- <html>
- <head>
    <title>About Rainbow</title>
</head>
- <body>
    <hr/>
        This line
    <hr/> <hr/>
        is broken in
    <hr/>
        three parts.
</body>
</html>
```

Figure 2.7 : Illustrating a horizontal rule

The output of code written in figure 2.7 is shown in figure 2.8.

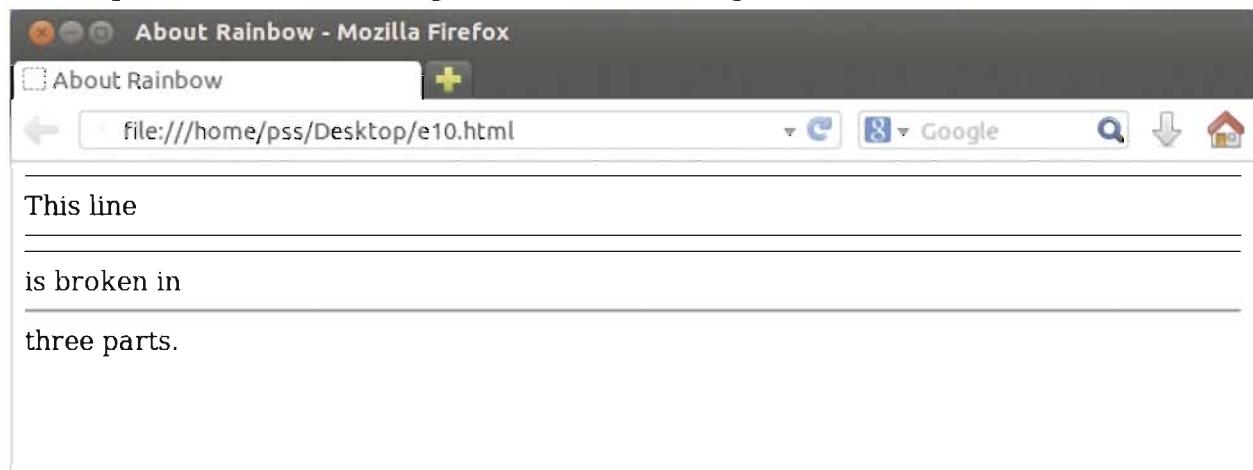


Figure 2.8 : How browser displays the `<hr>` command

There are five attributes which can be used with the horizontal line. These are given in the table 2.2.

Attributes	Description
size	Defines the thickness of the line. You may use pixel size 3, 5 or even 10.
color	Defines the line colour in explorer such as the Mozilla Firefox. The line colour must be defined in hexadecimal.
width	Defines the length of the line. The length can also be defined as in percentage of the width of the browser's window. The default value of the width is 100%.
align	Defines alignment of line in RIGHT, LEFT, or CENTER manner.
noshade	Displays a solid plain line instead of shaded line.

Table 2.2 : Attributes of a horizontal rule

Assume that the user wants to display a horizontal rule of size 2 that uses only 25% of the screen size and is aligned to the right side of the screen then following tag can be used.

`<hr size=2 width="25%" align="right" >`

Note that in HTML closing of `<hr>` tag is not compulsory. In addition to the above arguments, we can also fill the horizontal line with some images. Embed this line into a valid HTML code and view it in a browser.

Example of Preformatted Text

Consider a scenario where we would like to display a notice about a forthcoming event. The content of the notice is as shown in table 2.3.

Practicing Rainbow Colours

Attention Please !

The rainbow is made up of seven colours. When we mix all seven colours they will become a white colour. To demonstrate this practically, we have arranged a laboratory session. The detail of the laboratory session is as follows:

Date	:	6 August
Time	:	9 : 30 am
Place	:	First Floor, Lab-1
Instructor	:	I M Patel

Principal

Table 2.3 : Contents of notice

Let us first make a few observations from the information given to us in form of table 2.3.

1. The text shown in table 2.3 must be included in body section of the HTML document.
2. The first line ('Practicing Rainbow Colours') is heading line and appears in center. We may use `<h1>` tag with attribute `align="center"` to display it appropriately.
3. The second line ('Attention Please!') is normal text. We may use `<p>` tag with attribute `align="center"` to display it appropriately.
4. There is a normal text informing us about the experiment going to take place in a laboratory. We may display the content using simple paragraph tag.
5. The next four lines that give information about date, time, venue and instructor are presented in different way. This is an example of preformatted text. We can show it exactly in the same way by using tag pair `<pre>` and `</pre>`.

The HTML code that publishes content shown in table 2.3 is given in code listing 2.2.

```
<html>
<head>
<title> Notice </title>
</head>
<body>
<h1 align= "center "> Practicing Rainbow Colours </h1>
<p align =center> Attention Please! </p>
<p> The rainbow is made up of seven colours. When we mix all seven colours
    they will become a white colour. To demonstrate the experiment,
    we have arranged a laboratory session. The detail of the laboratory
    session is as follows:
</p>
<pre>
    Date      : 6 August
    Time     : 9:30 am
    Place    : First Floor, Lab-1
    Instructor : I M Patel
                Principal
</pre>
</body>
</html>
```

Code Listing 2.2 : HTML code for setting tabs

The output of the code is shown in figure 2.9.

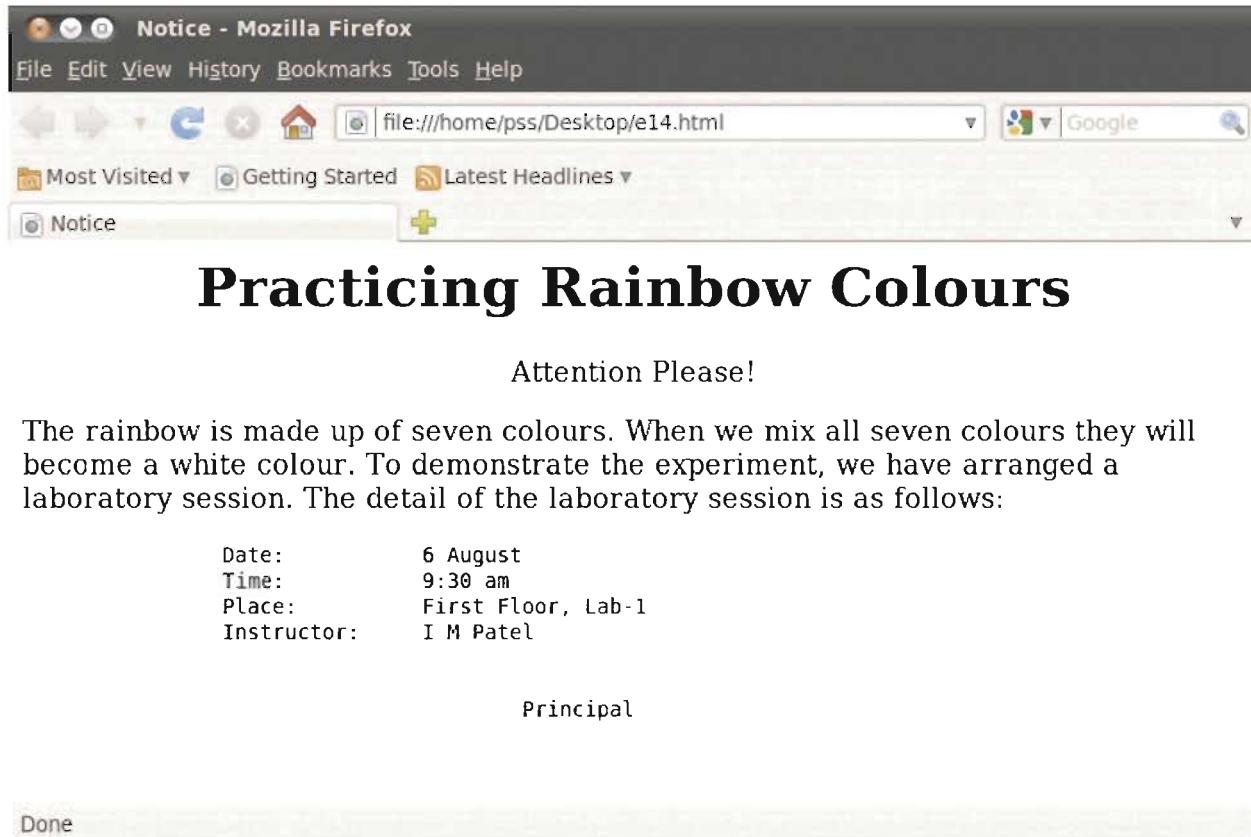


Figure 2.9 : Preformatted text in HTML

Formatting Characters

In the previous chapter we have seen that how text can be made bold, italics, and underlined. When we physically indicate the formatting style, browser will follow the instructions strictly and publishes the content in the said manner. Such tags are known as physical style tags. Table 2.4 summarizes the physical style tags.

Tag	Description
 ... 	Displays the content in bold fonts.
<i> ... </i>	Displays the content in italics.
<strike> ... </strike>	Displays the content with a line strike through it.
_{...}	Displays the content as a subscript.
^{...}	Displays the content as a superscript.
<tt> ... </tt>	Displays the content in a fixed typewriter-like font.
<u> ... </u>	Displays the content as underlined text.

Table 2.4 : Physical style tags

There is another way to format text. Instead of physically mentioning the formatting instructions in the browser, we just have to tell the browser what we want. Tags that manage such formatting are known as logical style tags. Following are example of some logical style tags.

** and **

The content is displayed in emphasized manner using this tag pair. Important things such as "Must be done" and "Important" can be written using this tag. Example of this tag is as follows.

What a beautiful rainbow...!

** and **

The content is displayed in much emphasized manner using this tag pair. Example of this tag is as follows.

** What a beautiful rainbow...!**

Observe the difference between the emphasized and strong text content in the above examples. Some other tags that manage the logical formatting are summarized at table 2.5.

Tag	Description
<dfn>	To publish the text content in defined fashion.
	To give emphasis on specific text.
<cite>	For citing important text such as titles of books, films, etc.
<code>	To demonstrate computer programming code segments. The text is displayed in a fixed-width font.
	This tag prints the content in strong emphasis manner. The content is typically displayed in bold.

Table 2.5 : Logical style tags

Font Tag

The font tag is used to set a specific font and size. Some examples are shown below:

```
<p><font size="3" color="red">This is some text!</font></p>
<p><font size="2" color="blue">This is some text!</font></p>
<p><font face="verdana" color="green">This is some text!</font></p>
```

Here you may notice that instead of a hexadecimal number, we have used colour names such as green, blue and red. The above examples have also used attributes along with the font tags. These attributes are size and colour; which specify size of the content and colour of the content respectively.

At the beginning of the HTML code, it is possible to decide the default font and its attributes for the page. This can be done as follows.

```
<basefont face="Arial" size="16">
```

The above line can be tested by writing the full HTML code as shown in code listing 2.3.

```
<html>

<body>

<basefont face = Arial size=16>

<p><font color="red">This is some text!</font></p>

<p><font color="blue">This is some text!</font></p>

<p><font color="green">This is some text!</font></p>

</body>

</html>
```

Code Listing 2.3 : Sample HTML code for testing font tag

Try to write and view this code in a browser. The output will look similar to the one shown in figure 2.10.

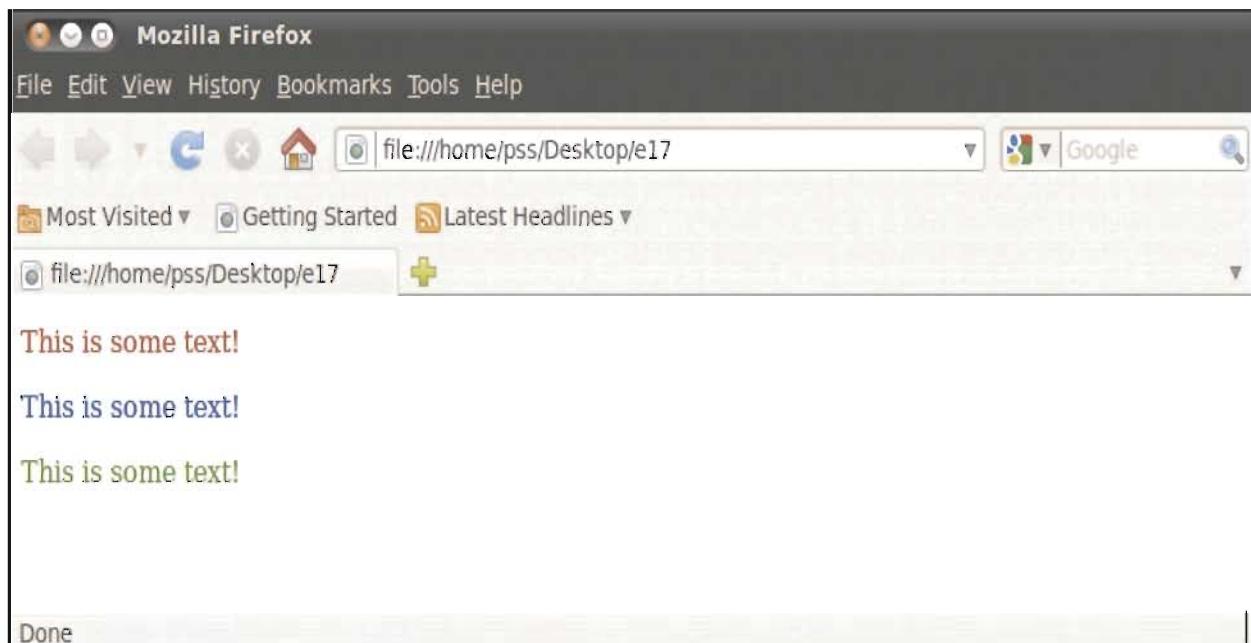


Figure 2.10 : Base font and font colour

Note :

Observe that the file name visible in figure 2.10 is e17 instead of e17.html. Linux generally does not use the file extensions. When an extension is used it indicates the content or usage of that file. For example in our case the HTML extension tells us that it is a HTML file. To view the desired output user on Linux does not need a file extension but needs to make sure that a proper program is used to open the file.

Special Characters

Special characters such as <, &, ©, etc. can be included on the web page along with & (ampersand) as a prefix of the mnemonic code (symbol) of the letter. Instead of such symbolic code, we can also give the ASCII value for the symbol. This is required because the characters like '<' and '>' cannot be used directly in an HTML document, as they can be mixed with tags. Use the special character shown in table 2.6.

Symbol	Description	Mnemonic code	ASCII value
<	Less than	lt	60
>	Greater than	gt	62
&	Ampersand	amp	38
©	Copy right	--	169
¼	One quarter	--	188
½	Half	--	189
¾	Three quarters	--	190
®	Registered trademark	erg	174

Table 2.6 : Mnemonic codes and ASCII values for special characters

Here is an example demonstrating use of less than (<) and greater than (>) characters within an HTML code.

```
<p align="center"> &lt Basic Information &gt</p>
```

The above line will print following text on a web page.

```
< Basic Information >
```

Tags Covered in this Chapter

In this chapter we have discussed the tags mentioned in table 2.7.

Tag	Description
<!-- ...-->	To embed text of comments in an HTML code.
	Displays the content in bold fonts.
<base>	Specifies the base URL/target for all relative URLs in a page.
<basefont>	To set default font for the web page.
<body>	Defines body section of an HTML document. It has attributes such as background image, background colour, text colour, link colour, etc.

Tag	Description
<cite>	For citing important text such as titles of books, films, etc. the text is typically displayed in italics.
<code>	To demonstrate computer programming code segments. The text is displayed in a fixed-width font.
<dfn>	To publish the text content in defined fashion. Typically the text is displayed in italics.
	The content is displayed in emphasized manner.
	Font tag is used to set specific fonts and size.
<head>	Head section of an HTML document.
<hr>	Defines a horizontal rule. It has attributes such as size, colour, width, alignment, no shade etc.
<i>	Displays the content in italics.
<link>	Defines the relationship between a document and an external resource.
<meta>	Provides additional information to search engine and other utility programs about author, keywords, description, purpose etc.
<script>	Defines a client side script.
<strike>	Displays the content in a strikethrough manner.
	The content is displayed in much emphasized manner.
<style>	Defines style information for a document.
<sub>	Displays the content as a subscript.
<sup>	Displays the content as a superscript.
<tt>	Displays the content in a fixed typewriter-like font.
<u>	Displays the content as underlined text.

Table 2.7 : HTML tags discussed in Chapter 2

An Example

Let us consider a real life example. Suppose you want to build a simple website of your school. The website is compilation of school's resources and activities. The main page of the website of the school may be known as index or homepage. On the first web page, name of the school is published. We may have additional information about school's affiliation to a trust, registration number, full address and contact information on this home page itself. Besides this, the home page also has links to various other pages such as teaching activities, photo gallery, and programmes and events.

Description of these web pages is given as follows :

- Home page of my school: The main page of the schools website include the school name, brief information about the school trust, school registration number, affiliation of the school to some government body and school contact details besides menu for other pages.
- Staff and teaching activities: This page shows brief introduction of the staff members including full name, qualification, classes they normally teach and their specialization along with their email address.
- Photo gallery: This page shows photos of classes, library, laboratory and recent events. You may develop this page later when you learn about image handling.
- Circular and Notices: This page contains information about forthcoming programmes and events. It also displays circular and notice for students and parents. For example, notice for sports day, essay competition, results and announcement of scholarship programme.

Consider HTML code given in code listing 2.4.

```
<html>
<head>
    <title>About my school</title>
</head>
<body vlink="purple" >
<h1> My School </h1>

<p> <h3>
    <a href="Home.html">Home</a>
    <a href="Activities.html">Activities</a>
    <a href="Staff.html"> Staff </a>
    <a href="Gallery.html">Photo Gallery</a>
    <a href="Event.html">Events</a>
    <a href="Notice.html">Notice</a>
    <a href="Contact.html">Contact Us</a>
    </h3>
</p>
<h3>About My School</h3>

<p> My school is one of the best and most reputed school of this city. My school offers education right from kindergarten to 12th standard. My school is equipped with all sorts of modern day facility to enhance education. My school has a play ground, an assembly hall, library, and activity lab. </h3>
</p>

<p> My school offers teaching in English and Gujarati medium and is affiliated to Gujarat Secondary Education Board. The teachers in my school make learning easy and interesting by providing presentations and demonstrations. </h3>
</p>

</body>
</html>
```

Code Listing 2.4 : HTML code for the main page of My School website

Write the code given in code listing 2.4 in an editor. Save it as Home.html. View the code in a browser. The output will look similar to the one shown in figure 2.11.

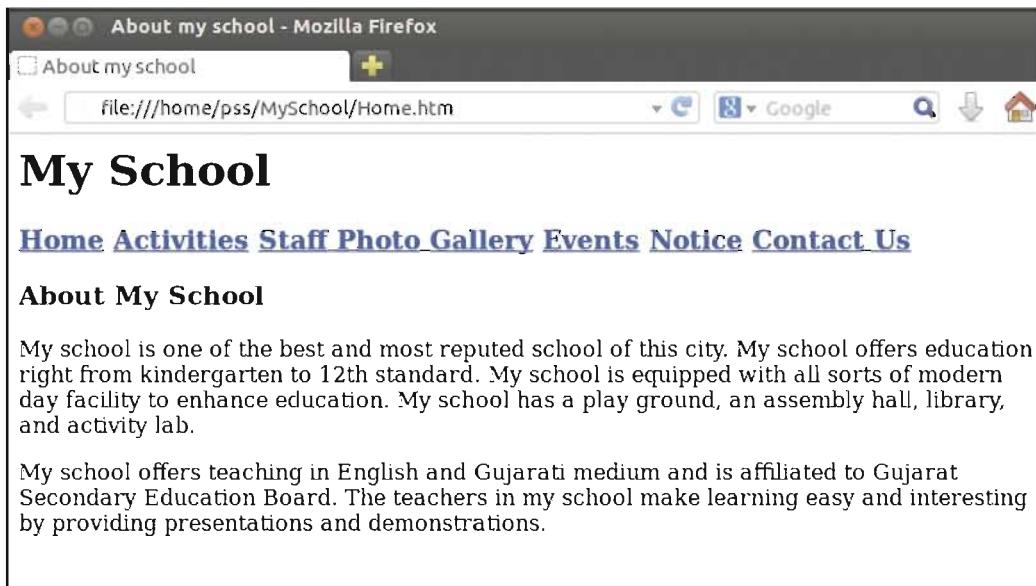


Figure 2.11 : Home page of school website

If you try to click on any link other than Home, you will get a message that file could not be found. This message comes because we have not yet created any other file besides Home.html.

To develop the files perform the steps mentioned :

- Write HTML code for other files such as activities, staff, photo gallery, notice and contact. Name them appropriately and save them in the same directory where you have stored home.html file. You may skip the photo gallery page. In next chapter when you learn about embedding images in to HTML documents, try to modify this project. Then you can also add your school's logo on main page. At this stage, you are able to add some images as a background of page.
- You have already created a notice following directions given in code listing 2.2 about "Practicing Rainbow Colours". You can rename the file and save it as Notice.html.
- Use necessary images, backgrounds and links beside the formatting tags. You may modify the text also. For example, on the home page, you may write your school's name, name of school trust, foundation date of school, contact information of school etc.

Summary

In this chapter we have learnt about meta-tag used in the head section of an HTML document. The meta-tags themselves may not display anything; however they often provide information regarding the author, keywords, description about the page. Such information is helpful to the search engine and other utility programs. Like head section, we have also discussed tags and attributes used in body section of an HTML document. HTML body section also provides physical and logical formatting style tags along with tag settings and displaying special symbols. We have also seen a way to add comment, default fonts and colours in an HTML document. At the end, we have seen a small website having many web pages describing activities of a typical school.

EXERCISE

1. Write a short note on head sections and meta tags used in it for an HTML document.
2. Explain how an image can be added on a web page background using HTML tag.
3. Explain colour representation in electronic media. Also explain colour coding schemes in brief.
4. Explain horizontal line `<hr />` element in HTML.
5. Explain font tag in HTML.
6. Explain how special characters will be presented using HTML.
7. Choose the correct option from the following :
 - (1) Which of the following form basic two sections of an HTML code ?

(a) Head and body	(b) Physical and logical
(c) Code and browser	(d) Meta-tags and body
 - (2) The meta-tags in an HTML document are written in which of the following section ?

(a) Body	(b) Code
(c) Head	(d) Special
 - (3) Which of the following about a web page is described when a meta-tag is used in HTML page ?

(a) Author, purpose and keywords	(b) Layout
(c) Style	(d) Size
 - (4) Title of a web page is embedded within which of the following tag ?

(a) <code><p></code> and <code></p></code>	(b) <code><body></code> and <code></body></code>
(c) <code><title></code> and <code></title></code>	(d) <code><h1></code> and <code></h1></code>
 - (5) In which of the following sections can we add comments in HTML document ?

(a) Head	(b) Body
(c) Both head and body	(d) Either head or body
 - (6) Which of the following is used to specify a colour in HTML code ?

(a) Colour code in hexadecimal	(b) Colour code in decimal
(c) Colour mixing model	(d) Pixel in percentage
 - (7) Which of the following is used to set a visited link in HTML code ?

(a) <code>alink</code>	(b) <code>vlink</code>
(c) <code>before link</code>	(d) <code>after link</code>

LABORATORY EXERCISE

1. Complete 'My School' web site development project explained in the chapter.
 2. Explore different websites of educational institute and study their design. Write a comment on these designs.
 3. Develop a page about yourself. Describe about you, your family, your friends, your education goals, and contact address. You may consider the design of page as follows.

I, me and myself...!				
<u>About Me</u>	<u>My Family</u>	<u>My Friends</u>	<u>My Goal</u>	<u>Contact Me</u>
.....
..... you may write about yourself here.....				
.....

Here all the underlined words are hyperlinks that link you to a different page. Also set active link and visited link property, light colour background. This should be a complete website as we have developed earlier website for 'My School'.

4. Go to <http://www.w3schools.com/>, find out HTML section from the menu and experiment "try it yourself". This page is an editor that takes your html code and presents a view of the web page as if it is viewed in a browser.



Handling Images in HTML

It is said that a picture is better than thousand words. Pictures or images attract majority of persons and convey an important message to the audience. If a web page shows a picture, the user will immediately pay attention to it. Further, it is observed that pictures increase degree of understanding and acceptance in general.

Images are added to a web page using the `` tag in HTML. We have already added an image as a background in the previous chapter. There are other ways to include an image on a web page. For example, the following line would add the image called `rainbow.jpg` into the page.

```

```

The above example uses `src` attribute which is described below along with other attributes that may be used with the image tag.

The image element (image tag with the image content and attributes) does not cause a line break, hence it is known as an inline element.

The `src` attribute

The `src` attribute provides information about location of the image. The location here refers to the source of the image. It tells the browser where to find the image. Many times we use URL as an image location. Without mentioning the source of the image, it is impossible for browser to find and display an image. To include an image from an URL given as "`http://pritisajja.info/images/img1.jpg`" we may write following tag.

```

```

The image file specified should have a proper image format such as `bmp`, `gif`, `tiff` or `jpg`.

Consider a scenario where you are surfing the Internet and liked an image. How would you get its complete URL to include it in an HTML code ? Answer is simple. The first option is to right click on the image and copy the image link. Another option is to save the image into the local memory of your computer. While opting for the second option you must be careful about the copyrights for the image.

When tags like `<p>` or `<h1>` are used with the content, the tags just have to display the content in a particular style. While using the image tag, we need not have to provide any content, but the source of image with some attributes. That is why the image tag is called an empty tag. However, when an HTML document containing image is displayed in a browser, the browser needs to retrieve the image. The image must be available to your local computer or server. If you have used an URL, check that your internet connection is working.

It is a good practice to create a separate folder/directory for images. This approach is better, especially when the website is large and uses multiple images. Code to add three images one by one in an HTML is shown in code listing 3.1.

```

<html>
<head>
    <title>My favourite food ....!</title>
</head>

<!-- ----- -->
<body>
    <h1> My favourite food ....!</h1>

    <!-- ----- -->
    <h1> <u> Chocolates </u></h1>

    <p>
        
        <h1>
            Chocolates are good for health, better for hunger and best for mood !
            Do not forget to clean your teeth until it is too late !
        </h1>
        </p>

    <!-- ----- -->
    <h1><u> Fruits and Dry Fruits </u></h1>
    <p>
        
        <h1>
            Do not think dry fruits are dry, though they are dried fruits !
            They are really really very interesting !
        </h1>
        </p>

    <!-- ----- -->
    <h1> <u> Ice creams </u> </h1>
    <p>
        
        <h1>
            Ice creams are really cool ! These become coolest when you taste them !
        </h1>
        </p>

    <!-- ----- -->
</body>
</html>

```

Code Listing 3.1 : Adding multiple images on a web page

Code listing 3.1 contains some dotted lines. This dotted line will not be displayed in the web page as they are embedded within comment tags (<!-- and -->). The line simply divides the code section into separate blocks for ease of reading and better understanding. As browser will accept only valid HTML statements, we cannot directly enter a dotted line. We have to embed the line within comments tag.

Figure 3.1 shows output of the HTML given in code listing 3.1.

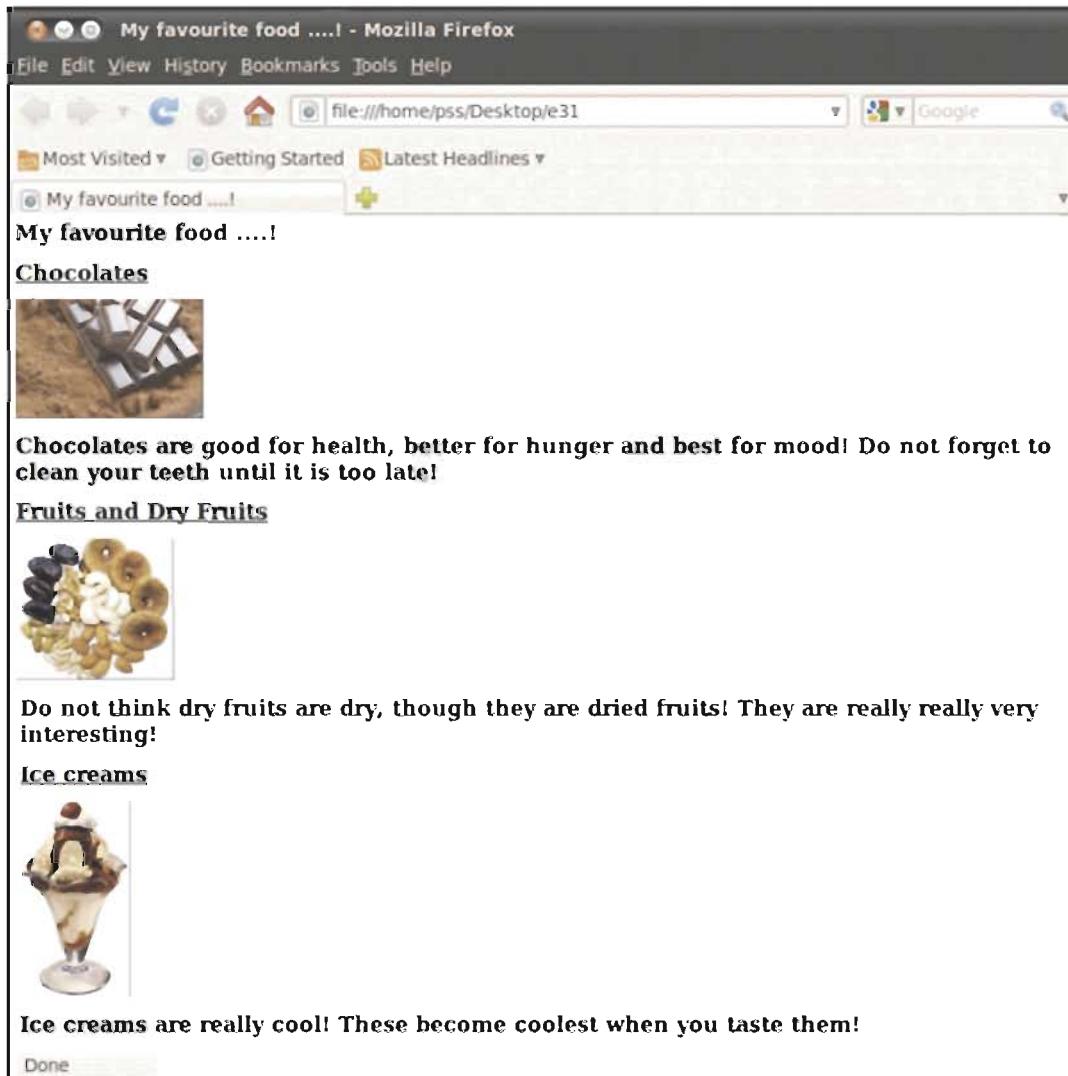


Figure 3.1 : Multiple images on a web page

The browser will get all the images from the given source one by one and display them in the order in which they are mentioned in the HTML code. Users generally see a page display along with all its images.

The alt attribute

The alt attribute along with an image tag describes the source image. It is always advisable to define this attribute, as it describes the image. The example of the alt attribute is as follows:

```

```

This attribute is quite useful. The first reason is it describes the image, which is many times necessary. Another reason is due to some reasons if the browser does not display the image, the reader will at least know, what to expect as a picture. The third reason is, when the web page is read by

the software such as screen reader (special software designed for visually challenged people) and search engine, the interpretation of the image becomes easy.

The height and width attributes

The height and width attributes specify the values of height and width of the image in pixels as illustrated below.

```

```

Here, the width attribute indicates the browser how wide the image should appear on the page. The height attribute specifies how tall the image should appear on the page. Values of both width and height attributes can be specified using the number of pixels. It is not compulsory to use both the width and height arguments together; you can use any one of them. However, using only one attribute will not be much useful.

By providing additional information about the height and width of an image to the browser, presentation and loading of the image in a browser become smooth and efficient. Here, the browser knows the amount of space to be allocated to the image. It is to be noted that your image should not be greater than your screen. However, it is possible to accommodate larger image into an HTML code. It is also not advisable to magnify or shrink your images by using much higher or lower height and width values. Doing this will ruin quality and visibility of the image.

The align attribute

The align attribute is used to align the given image within the page or any element that contains the image. Example for the same is as follows.

```

```

The align attribute can take one of the values shown in table 3.1.

Value	Purpose
Top	The image is aligned at the top of the current line of text.
Middle	The image is aligned in such a way that the middle of the image appears at the current text.
Bottom	The bottom of the image is aligned with the baseline of the current line of text.
Left	The image is aligned to the left side of the containing window or element.
Right	The image is aligned to the right side of the containing window or element.

Table 3.1 : Possible values for the align attribute

The border attribute

An image on a web page can be highlighted with a border. By default, images do not have borders. To create a border around an image, the following attribute can be used.

The border attribute was created to specify the width of the border in pixels:

```

```

Some browsers (such as the Internet Explorer) highlight image when link is given to the image; that is when an image is used as a link, it is highlighted with a border.

The id attribute

With the help of the id attribute, you can specify an identifier (name) for the image. Later the image can be referred by the identifier in a script written in a programming language such as java. The tag below shows how to use name attribute.

```

```

Adding space around image

To add space around an image vspace and hspace attributes are used. To add space over and under the image the vspace attribute is used. In a similar way, to add space to the left and right of the image the hspace attribute is used. The example given below illustrates the use of these attributes.

```

```

This attribute will be helpful when the image is inline with the text leaving no gaps. Leaving the vertical and horizontal gaps makes the image clearer and the page more readable. Code listing 3.2 shows an example of HTML code that leaves some gap surrounding the image of chocolates.

```
<html>
  <head>
    <title>My favourite food ....!</title>
  </head>
  <!-- ----- -->
  <body>
    <h1> My favourite food ....!</h1>
    <!-- ----- -->
    <h1> <u> Chocolates </u></h1>
    <p>
      
    <h1>
      Chocolates are good for health, better for hunger and best for mood !
      Do not forget to clean your teeth until it is too late !
    </h1>
    </p>
    <!-- ----- -->
  </body>
</html>
```

Code Listing 3.2 : HTML code illustrating hspace and vspace

When you see the HTML code illustrated in code listing 3.2 in a browser, it will look like figure 3.2. You may notice the horizontal and vertical gaps introduced by hspace and vspace attributes. Add some more images within the HTML code and see how your browser treats the images.



Figure 3.2 : Illustrating hspace and vspace

Now you have observed that the hspace and vspace attributes add spaces on both the sides of the image. That is, when hspace is used, it adds space to the left as well as to the right sides of the image. Similarly, when vspace is used, it adds space to the top as well as to the bottom of the image. To add image at only one side, we need to modify image appropriately. An alternative is to print a blank image beside the original image. However, the spacing and alignment will be difficult to manage in the second alternative.

Try to remove the "chocolate.png" file from your computer memory and see what happens. Will it print the alternative description of the image such as "Here comes the sweet image" ? You may find the output as shown in figure 3.3.



Figure 3.3 : Displaying alternative description in absence of image

Low Resolution Image

High resolution image takes lot of space and time for loading it on webpage. Till the high resolution (original) image is loaded, we may temporarily publish a low resolution image on the page as illustrated below.

```

```

To get low resolution image, you can use resize, crop or resample the image using techniques supported by a suitable image editing tool.

By adding an attribute managing the alternative low resolution image, we may solve problems of browser speed as well as size of the image; however, we have to create and store an alternative image. Further, the low resolution image may appear fuzzy and vague.

Embedding Images of Various Formats

So far, we have incorporated bmp (bit mapped picture) file into the image tag. But there are some other formats of image that can be embedded into the image tag. Among the various image formats, formats such as bmp, jpeg, png, tiff and gif are very popular. The description of these image formats are shown in table 3.2.

Image file format	Description
BMP	Bitmap graphic files for Windows and OS/2
GIF	Graphics Interchange File
JPEG	Joint Photographic Expert Group file
JPG	JPEG/JIFF Image file
PNG	Portable Network Graphics bitmap graphic file
TIFF	Tag Image File Format bitmap file

Table 3.2 : Image formats

Digital cameras and web pages normally use jpg/jpeg files; as these formats drastically compress the data in the files. The image formats jpg and jpeg are good for continuously toned images such as photographs. Commercially tiff format images are used very much, as they offer highest quality and good amount of compression of the image with minimum loss. Tiff format is actually one of the good lossless image formats. Other file formats such as gif and png also use the lossless compression technology. However, png format is comparatively slower in reading and writing.

Image as a Hot Spot

Many times the browser will take some time to load the big image. Above this, when we have multiple images on one or more pages, it will be confusing for users that which image should be considered first. Have you noticed some online shopping sites for books, jewelry, mobile phone and other such items ? On a single page you might have seen small images of many items displayed with brief information of each. When you find a particular item interesting, you may click the small image to go to the page containing detailed information as well as bigger, good quality image of the selected item. This can be done by applying a link to each of the small image on a page. You can say that here the image is considered as hot text. Since there is no text here, it is identified as hot spot. The small image itself has a link to the other suitable page. Code Listing 3.3 illustrates HTML code that uses such two small images and leads to two different web pages.

```

<html>
  <head>
    <title> Image Hot Spot </title>
  </head>
<!-- ----- -->
  <body>
    <h1> Either you can go to office or go to temple!</h1>

    <p> <h2>Click on any image below to see its larger version </h2> </p>

    <br>
<!-- ----- adding links to the images ----- -->

    <a href="big_office.html"></img></a></p>

    <a href="big_temple.html"></img></a></p>
<!-- ----- -->
  </body>
</html>

```

Code Listing 3.3 : HTML code for image as hot spot

Save this code as main.html. When you see the code in a browser, it will look as shown in figure 3.4.

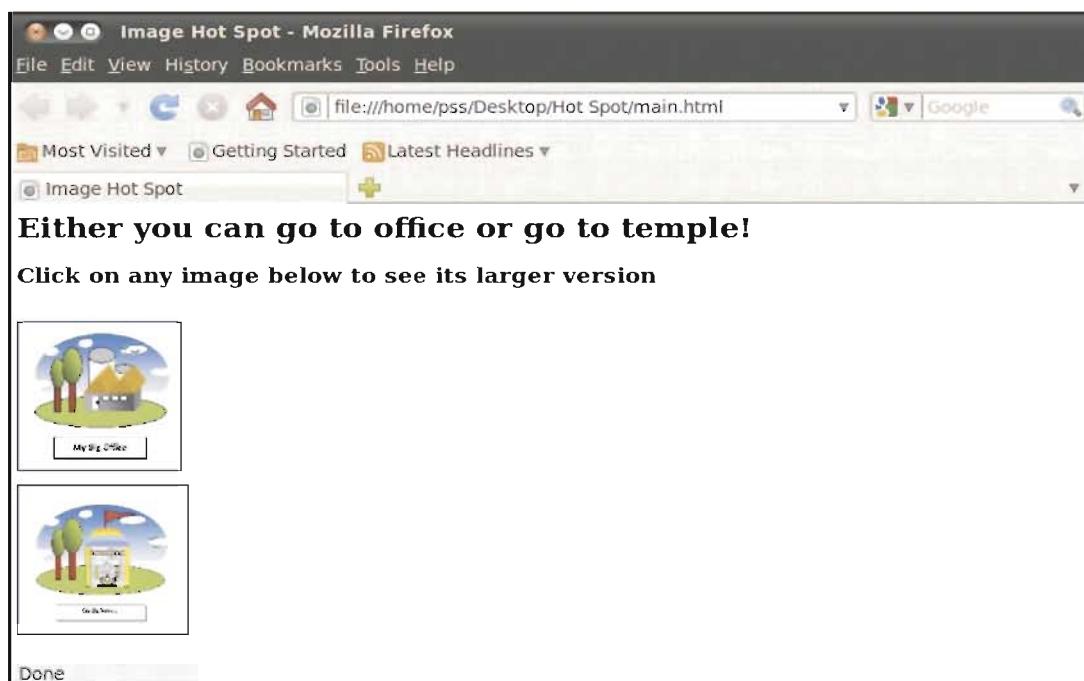


Figure 3.4 : Using image as hot spot

Besides the HTML code shown in code listing 3.3, we also need four images. We need one small image for office called `small_office.bmp` and another image called `big_office.bmp`. Similarly, we also need one small image for temple called `small_temple.bmp` and another image called `big_temple.bmp`. You can create the images or you may use the existing images by renaming them.

When you see the main.html file in a browser, you can see the output as shown in figure 3.4; provided two small figures for an office (small_office.bmp) and a temple (small_temple.bmp) are available.

Once you correctly visualize both the images in the main.html page in a browser; you may try clicking on images. When you click on first image the href tag (given below) redirects you to another page (big_office.html) considering the small image as hot spot.

```
<a href="big_office.html"></img></a></p>
```

Note that your big_office.html file must be ready with big office image embedded in it.

By clicking on the first image you will see the screen as shown in figure 3.5.



Figure 3.5 : When you click on the first image

HTML code for creation of big_office.html is given in code listing 3.4.

```
<html>
    <head>
        <title> The big image - office </title>
    </head>
    <!-- ----- -->
    <body>
        <h1> This is the bigger version of the office...! </h1>
        <br>
        <br>
        <br>

        </img>
    <!-- ----- -->
</body>
</html>
```

Code Listing 3.4 : Code for the first image

Similarly, when you click on the second image shown in figure 3.4, you will see output as shown in figure 3.6.

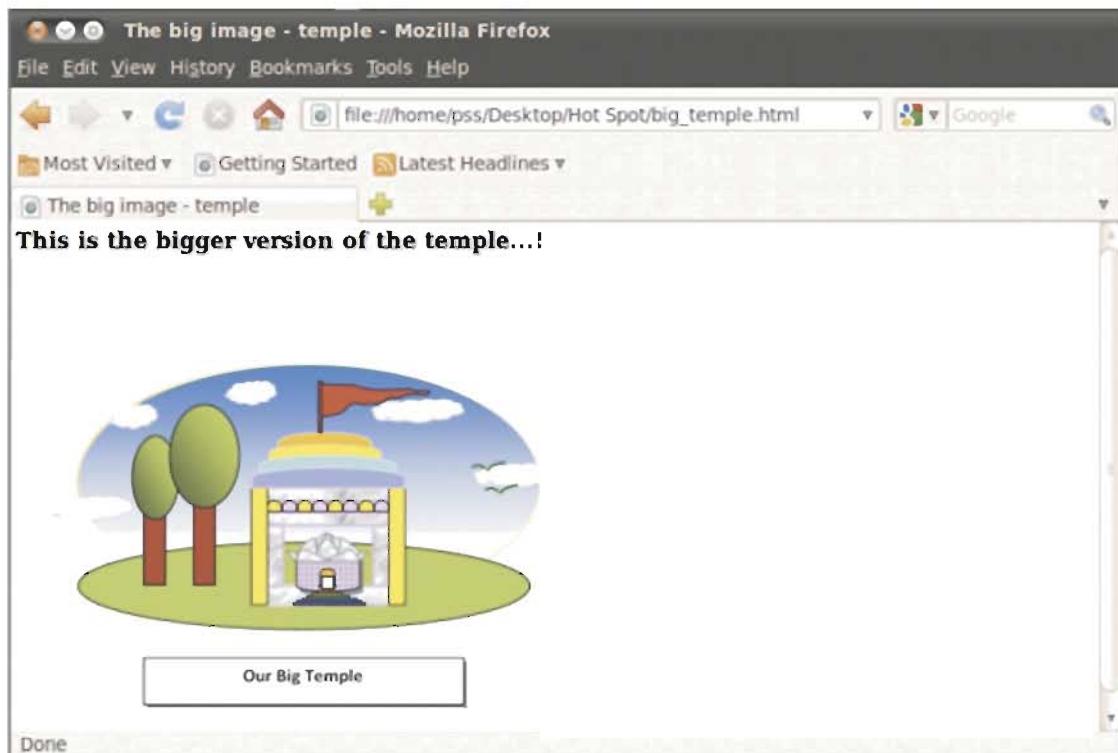


Figure 3.6 : When you click on the second image

HTML code (big_temple.html) that generates figure 3.6 is illustrated in code listing 3.5.

```
<html>
    <head>
        <title> The big image - temple </title>
    </head>
    <!-- ----- -->
    <body>
        <h1> This is the bigger version of the temple...! </h1>
        <br>
        <br>
        <br>

        </img>

    <!-- ----- -->
    </body>
</html>
```

Code Listing 3.5 : Code for the second image

Observe that to complete the above example you require four images and three HTML files as follows :

1.	Small_office.bmp	→	a small image of office in bmp format (or any valid image format).
2.	Small_temple.bmp	→	a small image of temple in bmp format.
3.	Big_office.bmp	→	a big image of office in bmp format.
4.	Big_temple.bmp	→	a big image of temple in bmp format.
5.	Main.html	→	a file HTML code with two small images, reference to the other two HTML files, and some text in it.
6.	Big_office.html	→	a file HTML code with bigger image of the office.
7.	Big_temple.html	→	a file HTML code with bigger image of the temple.

You may try to add some more small images in a main page. Small images are also identified as thumbnails. When you click on such small images arranged in a systematic way within a web page, each image will lead to the detailed (or bigger) version of it.

Image Map

In previous section we have added a link to an image. It is also possible to add multiple links to an image. Here each link points to a different reference. To create multiple links within a single image, we need to create multiple clickable locations within the image. Clicking on each such location, a particular reference (page) can be opened. Each such location is called a hot spot. Consider you have some kind of drawing of an area showing different places such as a corporate office, shopping mall, temple, police station, and fire brigade. On each of this location, we may create a link. For example on temple location, we may set a link that leads to a page containing information about the temple. The linked page may show some photos, news, brief history about the temple, and directions for how to reach the temple. In case you have a map (suppose map of India), then some regions of the map (states like Gujarat, Maharashtra, etc.) can be defined as hot spots. When one clicks on such hot spot, new web page can be opened.

Consider the image as shown in figure 3.7. The image contains a temple, an office, a mall, a fire brigade and a police station. These utilities are located on two main roads called "Shri M K Gandhi Marg" and "Shri Netaji Marg". On each of these five buildings and two roads a link is set. That is, on seven different areas seven different links are set. As stated earlier, these areas on which the links are to be set are known as hot spots. The hot spots must be big and visible enough; so that users can easily identify them and click. Otherwise, users will find it difficult to select the hotspot and follow the link. Further, the image should convey information that by clicking on each such area/hot spot, the user may be redirected to a new web page showing detailed information about the selected area. For example, "click on temple to know more ..." message can be given to the user when the image shown in figure 3.7 is displayed in a browser. Generate the image shown in figure 3.7 using appropriate image creation tool and name it as city.bmp.

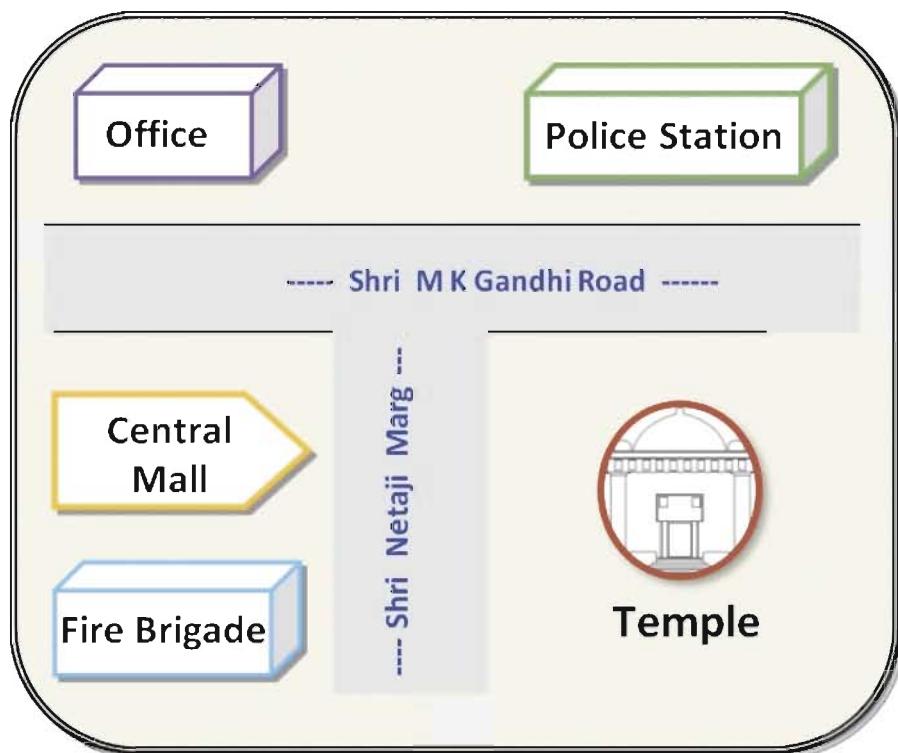


Figure 3.7 : A scenario of a city market place

The location of a hot spot is mentioned using x and y co-ordinates measured from the top left hand corner of the image. These coordinates are used in two ways :

- To specify where the hot spots are
- To compare the coordinates provided by user through a click.

The browser at the user's side identifies the coordinate of the user's click and takes user to the referred web page. For this, `<map>` and `<area>` tags are used along with a `usemap` attribute into its image tag. The map tag has an attribute called `map`. The `map` attribute value must be matching with the `usemap` attribute value. See the following example.

```

```

Here, the "#roadmap" is the name of identifier of `usemap` we have created. Within the `<map>` and `</map>` tags, the hot spot co-ordinates and their corresponding links are to be specified. Here, the temple co-ordinates are (518,378,70) and its shape is circular. We can embed this information in `map` tags as follows.

```
<map name="roadmap">
<area shape="circle" coords="518,378,70" alt="Temple" href="Temple.html">
<!-- ----About co-ordinates of other hot spots ----- --&gt;
&lt;/map&gt;</pre>

```

The co-ordinates given in the second line of the above HTML segment represents temple in the `city.bmp` file shown in figure 3.7. As the temple is specified in circular area, we have used "circle" value to its shape. Suppose, the temple is in a rectangular shape, we need to use shape value as "rect". The `area` tag must mention the shape of a hotspot. The valid possible shapes are circle,

rectangle and polygon. The rectangle is specified by rect; circle is specified by circle; and polygon is specified by poly. Alternatively, full names such as rectangle are also used. See example below illustrating area tag with different shapes.

- `<area shape="poly" coords="32,301,183,301,239,352,188,399,32,399" alt="Central Mall" href="CentralMall.html">`
- `<area shape="rect" coords="32,432,233,532" alt="Fire Brigade" href="FireBrigade.html">`
- `<area shape="circle" coords="518,378,70" alt="Temple" href="Temple.html">`

Code listing 3.6 shows a complete HTML code to generate an image map using the scene of city shown in figure 3.7.

```
<html>
<body>
<p>Click on the location presented on map to look in detail:</p>
<!-- ----- -->


<!-- ----- -->
<map name="roadmap">

<area shape="rect" coords="46,37,219,141" alt="Office" href="Office.html">

<area shape="rect" coords="407,38,632,142" alt="Police Station" href="PoliceStation.html">

<area shape="poly" coords="32,301,183,301,239,352,188,399,32,399" alt="Central Mall" href="CentralMall.html">

<area shape="rect" coords="32,432,233,532" alt="Fire Brigade" href="FireBrigade.html">

<area shape="circle" coords="518,378,70" alt="Temple" href="Temple.html">

</map>
<!-- ----- -->
</body>
</html>
```

Code Listing 3.6 : HTML code for image map

The output of code listing 3.6 will look as shown in figure 3.8 in a browser.

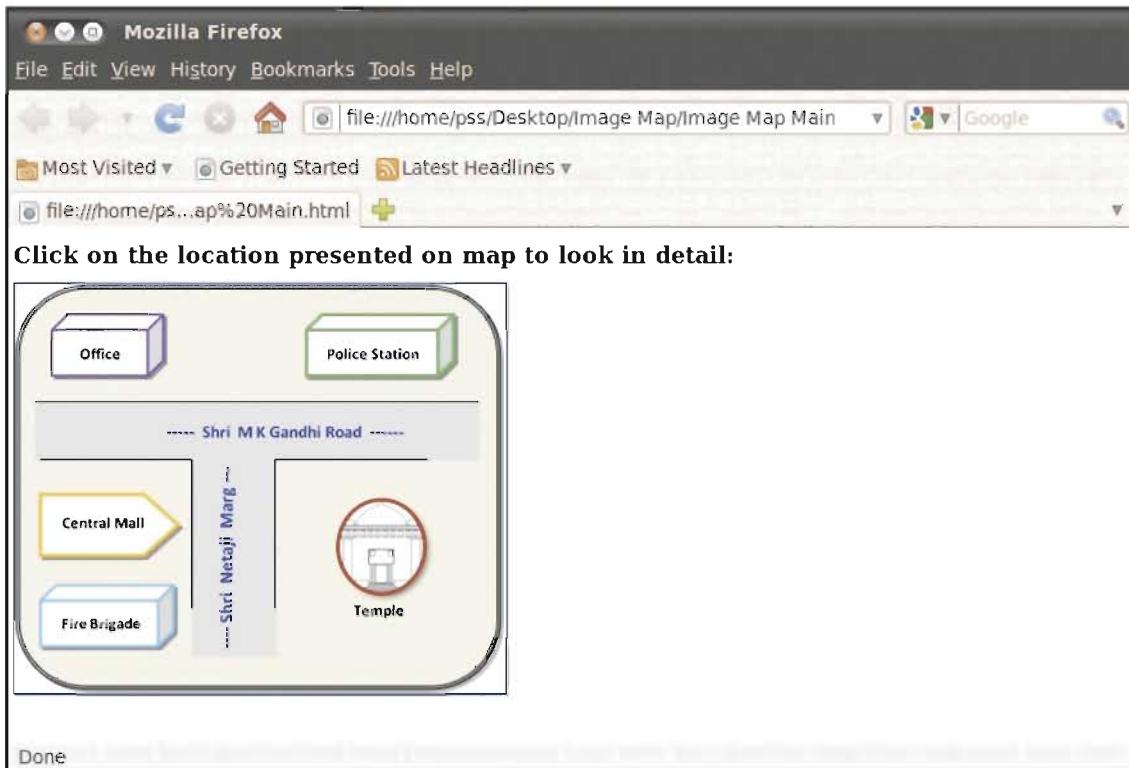


Figure 3.8 : Image map in a browser

When you click on the circular image of temple shown in figure 3.8, you will be redirected to a new page showing details about the temple. See figure 3.9 illustrating another file showing temple details.

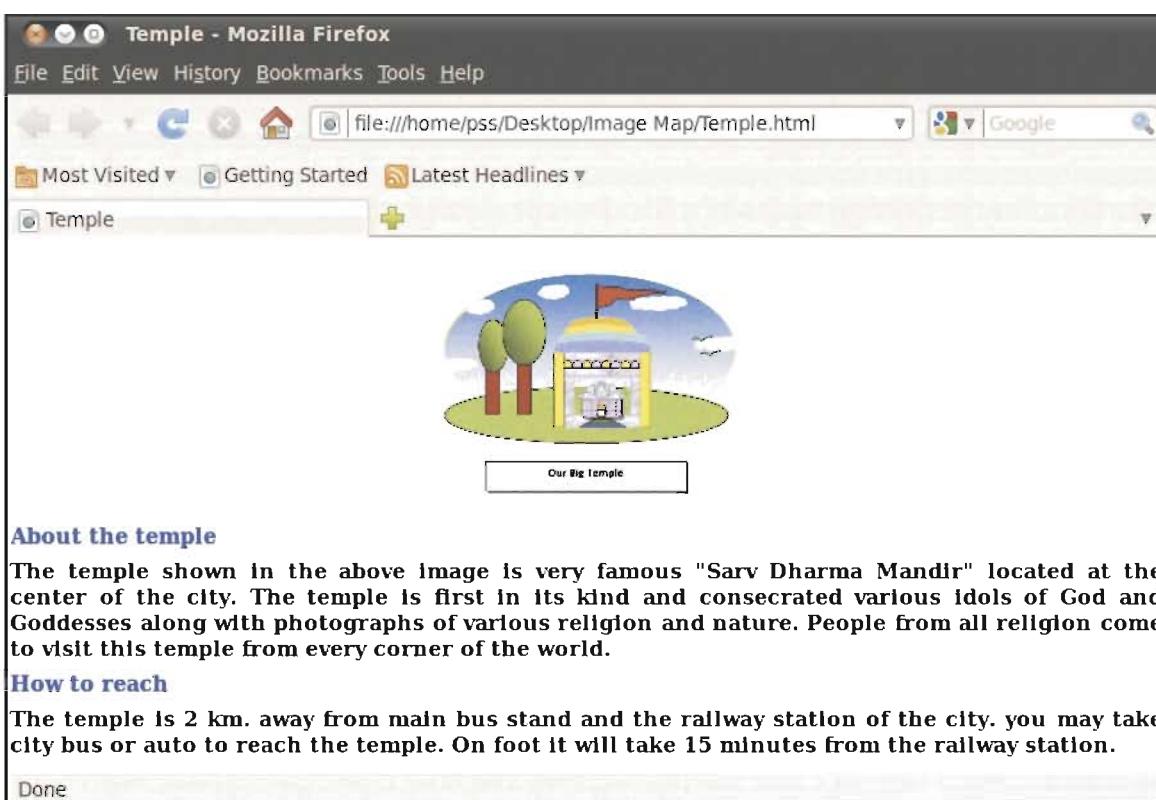


Figure 3.9 : Temple details

Code listing 3.7 shows the code which is required to generate the web page shown in figure 3.9.

```
<html>
<head> <title> Temple </title> </head>

<body>
    <center>
         </img>
    </center>

    <!-- ----- -->

    <h1> <font color="blue">About the temple </font> </h1>

    <p align="justify"> <b>
        The temple shown in the above image is very famous "Sarv Dharma Mandir"
        located at the center of the city. The temple is first in its kind and
        consecrated various idols of God and Goddesses along with photographs
        of various religion and nature. People from all religion come to visit
        this temple from every corner of the world.
    </b>
    </p>
    <!-- ----- -->
    <h1> <font color="blue"> How to reach </font> </h1>

    <p align =justify> <b>
        The temple is 2 km. away from main bus stand and the railway station of the city.
        you may take city bus or auto to reach the temple. On foot it will take 15 minutes
        from the railway station.
    </b>
    </p>
</body>
</html>
```

Code Listing 3.7 : Code for the temple detail

Similarly, you can prepare HTML files for office (Office.html), police station (PoliceStation.html), fire brigade (FireBrigade.html), and central mall (CentralMall.html). Verify that all the files work individually and try them from clicking the main HTML file showing overall scene of the city.

Linking Multimedia Files

A video as well as an audio file can be integrated into an HTML document using an anchor tag as illustrated below.

[follow this recipe and enjoy delicious food...!](food.mp4)

The example shown in previous line includes a movie file called "food.mp4". You should have a video/movie file with you. When the user clicks the words "Enjoy this... !" displayed on the web page, the video file will be shown in appropriate software. If the referred movie file is available on some other website or remote place, first it will be transferred to the user's computer. Such files are stored in local directory of user's computer as temporary internet files. If the required software is not found while executing the movie file, browser will ask you to choose alternative software from the computer or from the Internet.

Following is an example which adds a sound file in an HTML code.

[Enjoy this song...!](song.mp3)

Besides the mov file format, there are other formats for a video file. To name a few, these formats are avi, wmv, meg or mpeg, and swf. Some of these require downloading a video player that executes the files.

Summary

In this chapter we have learnt how to add image in an HTML document. Besides addition of one or more images into an HTML document, we have seen how to arrange image on the monitor screen using attributes like align, hspace and vspace. We have also used an image as hot spot. Not only the whole image, but part of image (such as a geographical map) can also be linked to different web pages using image map features. At the end, information about how to add video and audio files to an HTML is given. Using the information provided in this chapter, users are able to accommodate not only plain text and numbers into an HTML document, but video, animation, images and audio can also be included in a web page.

EXERCISE

LABORATORY EXERCISE

- 1.** Consider the example discussed in code listing 3.1 of this chapter. It prints images of chocolate, dry fruit and ice-cream. Extend the HTML code in such a way that if you click on chocolate image, it will lead you to a new HTML page describing chocolates. You may include more images of chocolates, some facts about chocolates, history of chocolates and recipe of making chocolates at home.

If you click on dry fruits image, it will lead you to a new HTML page describing the dry fruits. Similarly, if you click on ice-cream image, it will lead you to a new HTML page describing various ice-creams.

2. Create a page called index.html. Put some smaller three images of your favourite personalities on the index page. They can be your school teachers, your friends, sports persons or great authors. Also prepare three web pages that display some information about each of these three persons including a big photograph. Set link on each of the smaller photograph on the index page in such a way that it leads to the full biography of the selected person. You need to set three links on the three small photographs. (Hint: To create a link from an image, add an <a> element, and put the link of bigger picture in the href attribute of the <a> element.)
3. You may use the approach discussed in the above example 1 to develop your family tree. Create a web page with one or more images of your grandfather and grandmother. Develop other pages in such a way that, when you click on the image of your grandfather, a new page will appear with some more related pictures.
4. Implement the image map showing city scenario as discussed in this chapter.
5. Complete the school website project you did in Chapter 2 by adding the photo gallery. You may add logo of your school on the main page too.
6. Use an existing video file and embed it into a web page. Alternatively, you may create a video file through your mobile or any other device and embed it into a web page.





List and Table Handling in HTML

List Handling

While presenting information, sometimes we need to list different individual entities. Instead of writing such entities in continuous paragraph form, these individual items are presented as list with or without numbers. By presenting the entities in such way increases clarity in representation and aids readers in understanding them.

Let us see an example. When you want to shop some items from the market, you need to remember them. Figure 4.1 illustrates the list of some of such items in a classic way which can be taken with you while shopping.

Items to be purchased

- Chocolates
- Notebook
- Practice book
- Ball pen
- Markers

Figure 4.1 : An example of list

There are three basic types of HTML lists. These types are unordered list, ordered list and descriptive list. All the types are discussed in forthcoming sections of this chapter.

Unordered List

As illustrated in figure 4.1, an unordered list contains item along with a symbol. This symbol is also called a bullet. The list shown in figure 4.1 is an unordered list with bullet as 'o'. To create an unordered list in HTML, the `` tag and `` tag pair is used. The items of the list are enclosed within the `` and `` tag pair. HTML example that creates the unordered list is shown in code listing 4.1.

```
<html>
<!-- --- -->
<body>
<font color = "Blue">
    <h1>An Unordered List:</h1>
</font>
<!-- --- -->
<ul>
<font size="6">
```

```

<b>
    <li>Chocolates</li>
    <li>Notebook</li>
    <li>Practice book</li>
    <li> Ball pen </li>
    <li>Markers </li>
</b>
</font>
</ul>
<!-- ----- -->
</body>
</html>

```

Code Listing 4.1 : An unordered list

Write the code given in code listing 4.1 in the SciTE editor and check its output in a browser. The output will look similar to the one shown in figure 4.2.

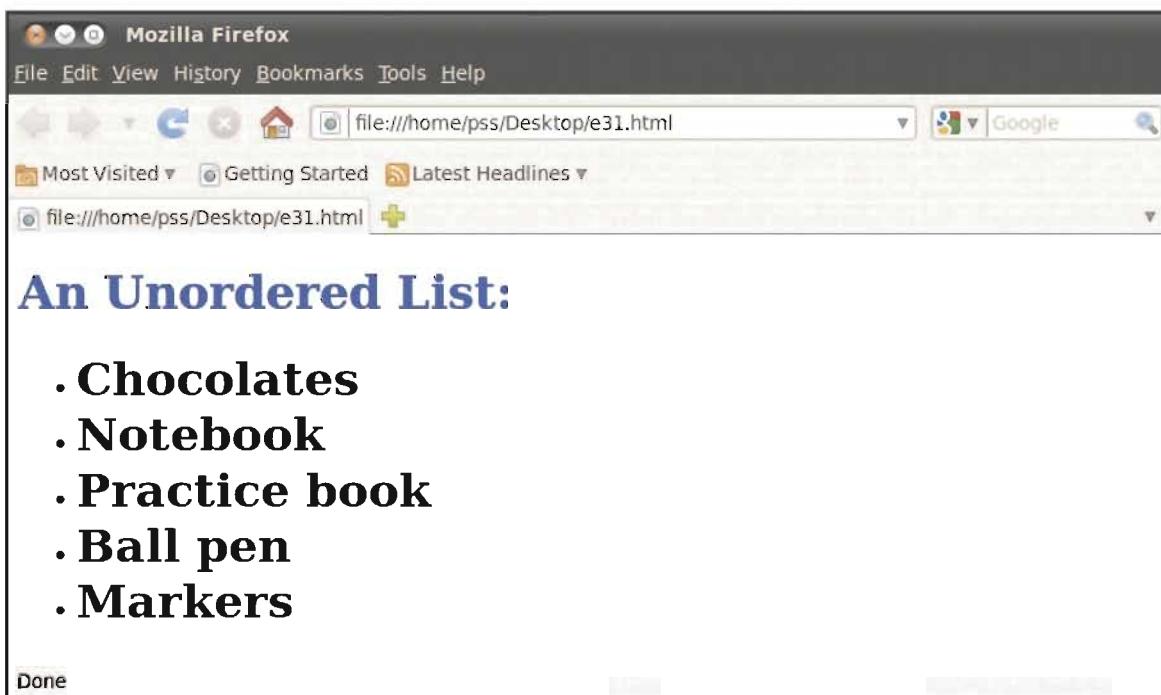


Figure 4.2 : An unordered list in a browser

Changing Bullets

The default shape of bullet is a filled circle. If you want to change the bullets, you can use an attribute called type. The types can be circle, square and disc. See the following example.

```
<ul type="square">
```

You can also change the bullets to unfilled circle by using the tag shown below :

```
<ul type="circle">
```

Modify the HTML code given in code listing 4.1 and experiment with different type of bullets.

List without Bullets

In case you do not want any bullet, you may use description list using tag pairs `<dl>` and `</dl>`. Here 'dl' is an abbreviated form of the word description list. Within the description list, we need to define description terms using `<dt>` and `</dt>`. That is, an item say Chocolates can be defined as follows:

```
<dl>
  <dt> Chocolates </dt>
</dl>
```

To define sub-items we may use description tag pair given as `<dd>` and `</dd>`. That is, we can add different types of chocolates and ice-creams. Figure 4.3 demonstrates different types of bullets on a web page.

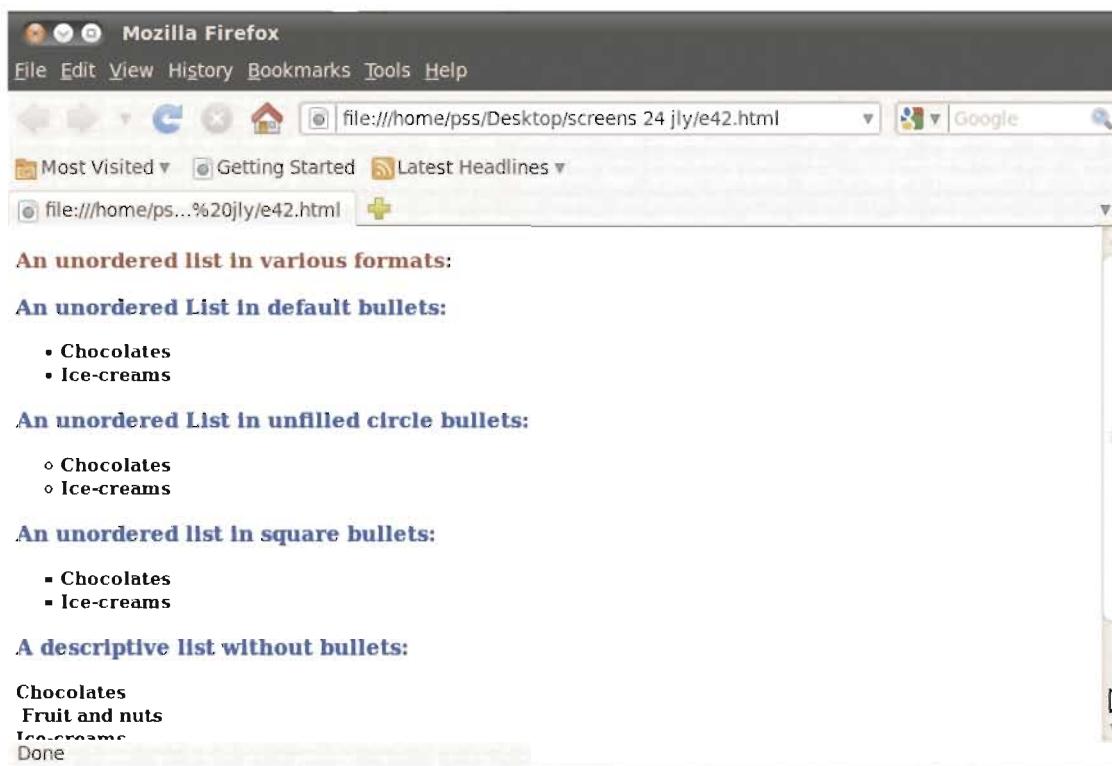


Figure 4.3 : Different types of bullets

HTML code to generate figure 4.3 is shown in code listing 4.2.

```
<html>
<body>
<font color ="Brown">
  <h1>An Unordered List in various formats: </h1>
</font>
<!-- -->
```

```

<font color ="Blue">    <h1>An unordered List in default bullets: </h1>    </font>
<ul>
<font size="6">
<b>
    <li>Chocolates    </li>
    <li>Ice-creams    </li>
</b>
</font>
</ul>
<!-- ----- -->
<font color ="Blue">  <h1>An unordered List in unfilled circle bullets: </h1> </font>
<ul type="circle">
<font size="6">
<b>
    <li>Chocolates    </li>
    <li>Ice-creams    </li>
</b>
</font>
</ul>
<!-- ----- -->
<font color ="Blue"> <h1>An unordered list in square bullets: </h1> </font>
<ul type="square">
<font size="6">
<b>
    <li>Chocolates    </li>
    <li>Ice-creams    </li>
    <li>Dry fruits    </li>
</b>
</font>
</ul>
<!-- ----- -->
<font color ="Blue">
    <h1>A descriptive list without bullets: </h1>
</font>
<dl>
<font size="6">
<b>
    <dt>Chocolates    </dt>
        <dd> Dark </dd>
        <dd> Fruit and nuts </dd>
        <dd> Milk  </dd>

```

```

<dt>Ice-creams    </dt>
    <dd> Vanilla </dd>
    <dd> Chocolate chips </dd>
<dt>Dry fruits    </dt>
    <dd> Almonds </dd>
    <dd> Cashew nuts </dd>
</b>
</font>
</dl>
<!-- ----- -->
</body>
</html>

```

Code Listing 4.2 : HTML code for different types of bullets

Heading of List

To provide heading of a list, we have to simply use `<lh>` tag. Obviously, the heading must appear before the list. Hence, before starting the `` and `` tag pair, you need to use the `<lh>` tag with the required heading. See the example given below.

`<lh> My Shopping List </lh>`

Ordered List

An ordered list contains items along with numbers or alphabets instead of bullets. To create an ordered list in HTML, the `` tag and `` tag pair is used. As usual, the items of the list are enclosed within the `` and `` tag pair. Code listing 4.3 shows the HTML example that creates an ordered list.

```

<html>
<body>
<font color ="Blue">
    <h1>An ordered list: </h1>
</font>
<!-- ----- -->
<ol >
<font size="6">
<b>
    <li>Chocolates    </li>
    <li>Ice-creams    </li>
    <li>Dry fruits    </li>
</b>
</font>
</ol>
<!-- ----- -->
</body>
</html>

```

Code Listing 4.3 : HTML code for an ordered list

Output of the code listing 4.3 is shown in figure 4.4.

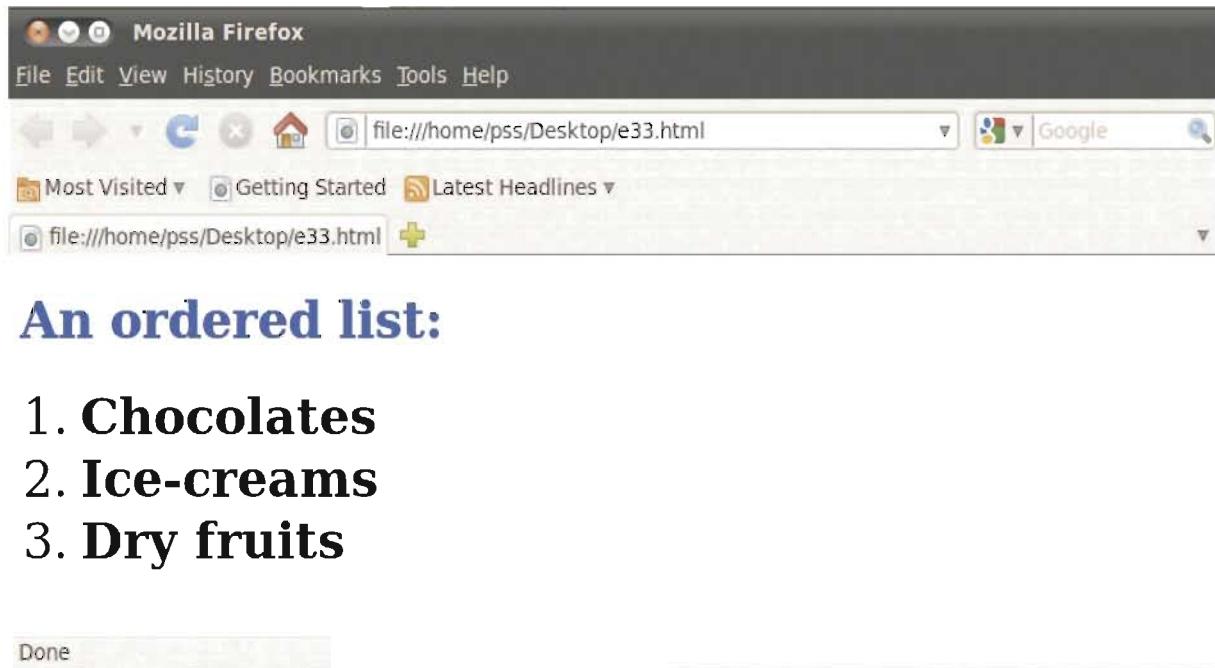


Figure 4.4 : Ordered list

Generally the ordered list always starts with number 1. You may start the ordered list with a specific number. Say, you want to start your list with a number 6; you may use start attribute with `` tag as shown below.

```
<ol start ="6">
```

Modify the HTML code given in code listing 4.3 in order to start the list with number 6.

Ordered List with Alphabets

Instead of numbers, we may use alphabets such as 'A', 'B', 'C' or 'a', 'b', 'c'. We may also use Roman numbers. This can be done by using type attribute with the `` tag. Table 4.1 illustrates possible values for the type attribute.

Value	Description
1	Numbers
A	Uppercase alphabets
a	Lowercase alphabets
I	Uppercase Roman numbers
i	Lowercase Roman numbers

Table 4.1 : Values of type attributes

An example of HTML code is provided in code listing 4.4. The code will print two ordered lists one with alphabets and another with small Roman numbers starting with number 10.

```

<html>
<body>
<font color ="Blue">
    <h1>An ordered list: </h1>
</font>
<!-- ----- -->
<font size="6">
</b>
<ol type=A>
    <li>Chocolates    </li>
    <li>Ice-creams    </li>
    <li>Dry fruits    </li>
</ol>
</b>
</font>
<!-- ----- -->
<font size="6">
</b>
<ol type='i' start="10">
    <li>Chocolates    </li>
    <li>Ice-creams    </li>
    <li>Dry fruits    </li>
</ol>
</b>
</font>
<!-- ----- -->
</body>
</html>

```

Code Listing 4.4 : HTML code to print Roman numbers as bullets

Output of the HTML code listing 4.4 is given in figure 4.5. You may try working with different values of the type attribute.

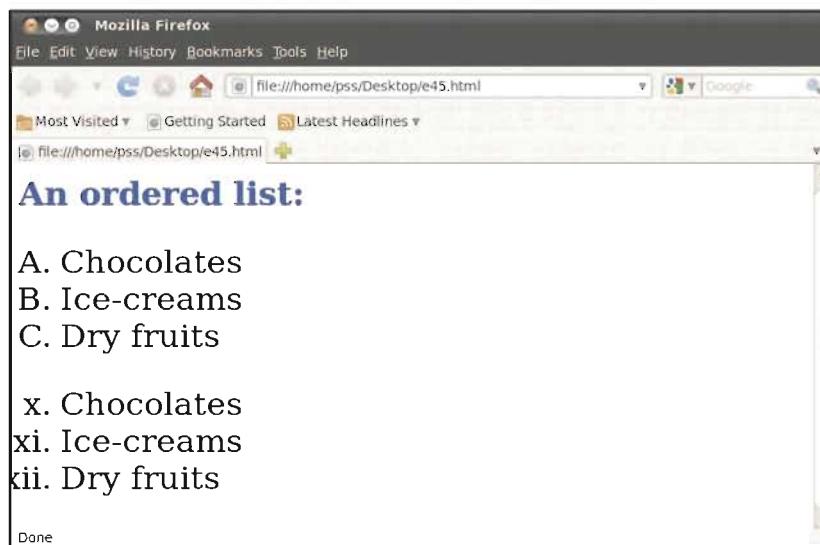


Figure 4.5 : Ordered lists with alphabets and Roman numbers

Nested List

It is possible to have a list within a list. This is called nesting of list or nested lists. Types of both the lists may differ. An example HTML code is given in code listing 4.5.

```
<html>
<body>
<font color ="Blue">
    <h1>An ordered list: </h1>
</font>
<!-- ----- -->
<ul >
<font size= "6 ">
<b>
    <li>Chocolates    </li>
    <font size= "4">
        <ol type="a">
            <li> Dark chocolates      </li>
            <li> Fruit and nuts Dark </li>
        </ol>
    </font>
    <!-- ----- -->
    <li>Ice-creams    </li>
    <ol type="a">
        <font size="4">
            <li> Vanilla          </li>
            <li> Chocolate chips   </li>
        </font>
    </ol>
    <!-- ----- -->
</b>
</font>
</ul>
</body>
</html>
```

Code Listing 4.5 : HTML code for nested lists

Output of code listing 4.5 is shown in figure 4.6.

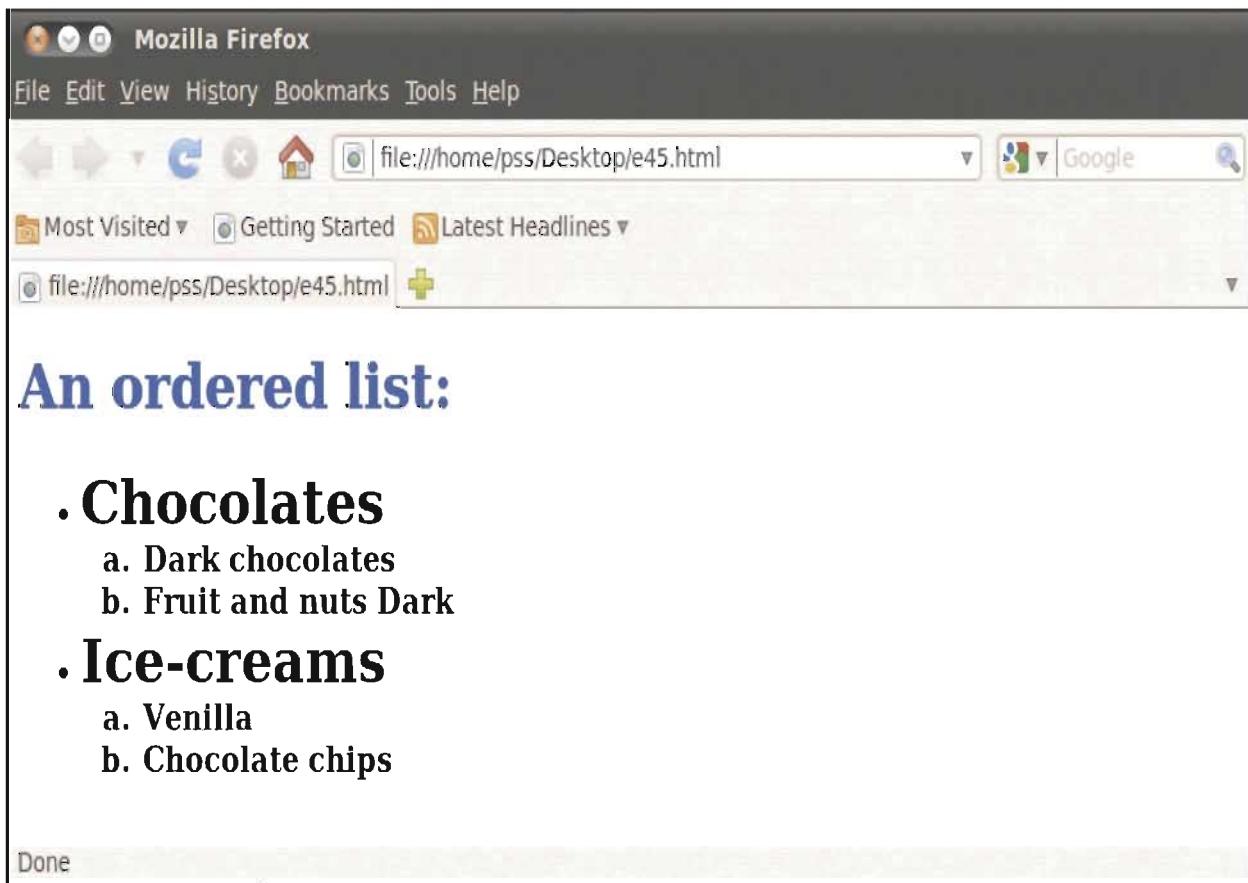


Figure 4.6 : Nested lists in HTML

There are other structures that publish the content in more structured way. One of such structure is called table. Following section presents tables in HTML.

Table Handling in HTML

Representation of information in proper format increases degree of understanding and facilitates ease of its use. Can you imagine a shopping bill printed in continuous paragraph manner ? What about your results sheet showing marks of different subjects? If it looks like the figure 4.7, it will be certainly very difficult for you to read. Not only it is effort taking and time consuming, but it also leads to misinterpretation of information.

Modern School Affiliated to Star Education, Gujarat Mr. Arvind B Patel Seat No 123
March 2013 Subjects: English, Gujarati, Science, Maths, SS*

Internal Marks (40): 20, 30, 25, 35, 25 External Marks (60): 35, 32, 48, 40, 35
Class: First Class SS*: Social Science

Figure 4.7 : Weird mark statement example

Now look at figure 4.8. Obviously there is no confusion at all. Name of the school, name of the student, date, all the marks, and result are clearly visible.

Modern School					
<i>Affiliated to Star Education, Gujarat</i>					
Mr. Arvind B Patel		Seat No 123		March 2013	
Subject:	English	Gujarati	Science	Maths	SS *
Internal Marks (40):	20	30	25	35	25
External Marks (60):	35	32	48	40	35
Class:	First class				
SS*:	<i>Social Science</i>				

Figure 4.8 : A sample mark sheet in proper format

Tables in HTML are very efficient in presenting structured information. A table contains information separated in form of grids. You might have used special note book for your mathematics calculation having rows and columns. Similarly, information can be presented in form of row and column in a table. The next section illustrates how a small table can be displayed on a web page using HTML tags. Later, we will see each tag that generates a table in detail.

Creating a Small Table

To create table in HTML `<table>` and `</table>` tags are used. That is, starting and ending of a table are marked by these tags. Within the `<table>` and `</table>` tags, we may use attributes such as caption (title) of the table, table border, table rows and columns.

Main content of a table is formed by rows and columns. A row in a table is defined by `<tr>`, the table row tag. The first row of a table is heading row, which is denoted by the `<th>`, table heading tag. Entries of remaining rows are entered using `<td>` tag. See code listing 4.6 that shows an example HTML code.

```

<html>
<body>
<h1>My First Table </h1>
<p>My first table is as follows.</p>
<table border="2">
<tr>
    <td> This is Row 1, Column 1 </td>
    <td> This is Row 1, Column 2 </td>
</tr>
<tr>
    <td> This is Row 2, Column 1 </td>
    <td> This is Row 2, Column 2 </td>
</tr>
</table>
</body>
</html>

```

Code Listing 4.6 : HTML code for simple table

You can write the code in your editor and see the output in a browser. It will resemble the output as shown in figure 4.9.



Figure 4.9 : A simple HTML table displayed on a web page

The HTML code shown in code listing 4.6 uses table tag with some attribute and component. The first attribute used within the table tag itself is the border attribute to print border of pixel size 2. The table further defines a row with <tr> tag. The table shown in figure 4.9 has two rows; hence two sets of <tr> tags must be used.

Within a row, with the help of <td> tag pairs, cells are defined. The attributes and tags used in this example are discussed later in this chapter. The objective of the example is to understand the row and column formation of the table. This arrangement is also known as grid type arrangement. Let us take a detailed view on other table tags.

Table Tag and its Attributes

The first tag that is used to define a table is table tag. The <table> tag can carry many attributes. Some of them are not much popular now-a-days; however, you may use them. The attributes are shown in table 4.2.

Attribute	Description
align	This attribute indicates alignment of table.
bgcolor	This attribute specifies background of the table.
border	This attribute specifies table border.
cellpadding	This attribute leaves specified gap between edges of the cells and their content.
cellspacing	This attribute manages space between each cell of the table.
dir	This attribute specifies the direction of text that is displayed in the table.
frame	This attribute controls the outermost border of the table.
rules	This attribute controls the presentation of inner borders of table.
summary	This attribute presents description of the table.
width	This attribute specifies width of the table.

Table 4.2 : Table tag attributes

Some commonly used attributes are discussed below:

Border Attribute

In the example discussed in code listing 4.6, a border with pixel size 2 is created using the border attribute. This attribute creates a border around the table as well as around each individual cell. The width of the border is given in form of pixel. Use of the border attribute is optional. If you use the attribute with value 0 (zero), no border will be visible. Consider the HTML code that we have created earlier using the code listing 4.6. Let us modify the value of the border attribute border by changing the value of border attribute as shown below.

```
<table border="4" align="left">
```

Now once again check the appearance of the table. It should look similar to the one shown in figure 4.10.

You might have noticed one more attribute we have used in the example given in previous paragraph; that is align attribute. Did you notice anything? You might not have noticed any change. Try to change the alignment of the table to the right and now you may able to notice change in the table alignment. We will see other alignment values later.



Figure 4.10 : Table with changed border value

Bgcolor Attribute

Let us make the table more attractive and colourful by adding background colour. To do this, we have to use bgcolor attribute. The bgcolor attribute sets the background colour for the table. The value of this attribute is either a colour name or a six digit hexadecimal code. Modify the first line of the HTML code you are currently experimenting (as given in code listing 4.6) as follows.

```
<table border="4" align="right" bgcolor="yellow">
```

Check the output by viewing the code in a browser.

Cellpadding Attribute

If two table cells are placed next to each other and both contained text, there may be a problem. If there is not enough space between the edges of the cells and the text, the words would combine with each other, making them hard to read. Similarly, if there is a border around each cell and

the text touches the border, it will be hard to read. By adding some space as padding to the cell makes their contents easier to read.

The cellpadding attribute is used to create a space between the edges of a cell and its contents. The cellpadding attribute pads some spaces inside each wall of the cell of the table. The value can be specified either in pixels or as a percentage value. Here, the value specified in percentage refers to percentage of the width of each cell of the table. See the following example.

```
<table border="4" align="center" bgcolor="pink" cellpadding="25">
```

Alternatively, you may use percentage value as follows.

```
<table border="4" align="center" bgcolor="yellow" cellpadding="20%">
```

You might have noticed that we have also changed values of align and bgcolor attributes. Check the changed appearance of the table in a browser.

Cellspacing Attribute

The cellspacing attribute is used to create a space between the cells of the table. The amount of the space can be specified either in pixels or as a percentage value. Here, the percentage is a percentage of the width of each cell of the table. See the following example.

```
<table border="4" align="center" bgcolor="yellow" cellspacing="20%">
```

Or

```
<table border="4" align="center" bgcolor="yellow" cellspacing="25">
```

Width Attribute

The width attribute is used to specify the width of the table. The value as usual is given either in pixels or in percentages of the available space as follows.

```
<table border="4" align="center" bgcolor="pink" cellspacing="25" frame="box" rules="cols" width="50%">
```

Or

```
<table border="4" align="center" bgcolor="pink" cellspacing="25" frame="box" rules="cols" width="50%">
```

Align Attribute

The align attribute specifies the position of the content of all of the cells in the row. Just like simple text the cell content can also be aligned. Table 4.3 lists the possible values for the align attribute.

Value	Description
left	Content is left aligned. This is the default case for normal text.
right	Content is right aligned.
center	Content is centered horizontally within the cell. This is the default case for headings.
justify	Text within the cell is justified to fill the cell.
char	Cell contents are aligned horizontally around the first instance of a specific character (for example, numbers could be aligned around the first instance of a decimal point).

Table 4.3 : Possible values for the align attribute

If the align attribute has a value of char, then the contents of each cell of the table within the row will be aligned around the first instance of a specific character. The given character is known as an axis character. The default character for this attribute is the decimal place. By using the decimal point as a char for alignment (char=".") , the existing decimal figures on the page would be aligned by the decimal point as shown as mentioned:

1	2	3	.	5	6	
				1	2	7
3	8	1	6	.	4	5

After having a detailed look at the table tag and its attributes, let us now see table row tag in following section:

Table Row Tag

The <tr> tag is used to display a row in a table. Everything appears within a <tr> tag should appear on the same line. It can carry three attributes as shown in table 4.4.

Attribute	Description
align	Content of the row is aligned as specified.
bgcolor	Background colour for selected row.
valign	Specifies the vertical alignment of the contents of each cell in a row.

Table 4.4 : Attributes of the table row tag

Let us now see how these attributes can be used in the HRML code.

Align Attribute

The align attribute indicates position of the content of all of the cells in the row. The general form of an alignment attribute is as follows :

align="alignment"

The align attribute takes values such as left, right, center, justify and char as shown in table 4.3. Use these attributes in any valid HTML code displaying table.

Bgcolor Attribute

Earlier, we have seen the bgcolor attribute to paint background of the table by giving colour code in hexadecimal or name of colour. What if, only a row of a table is required to be painted with a specific colour? The answer is again the bgcolor attribute with either colour name or colour code in hexadecimal. The bgcolor attribute sets the background colour for the row.

The bgcolor attribute is commonly used on the <tr> element to paint alternate rows of a table with different colours. This will make it easier to read across each row. Try the HTML example given in code listing 4.7.

```

<html>
<body>
<table border="1">
<tr >
    <th>Name of player</th>
    <th>Points earned</th>
</tr>
<tr bgcolor="lightGreen">
    <td>Disha</td>
    <td>110</td>
</tr>
<tr >
    <td>Sweety</td>
    <td>100</td>
</tr>
<tr bgcolor="lightGreen">
    <td>Gayatri </td>
    <td>90</td>
</tr>
</table>
</body>
</html>

```

Code Listing 4.7 : HTML code for painting alternative rows with light green colour

The output of code listing 4.7 will look as illustrated in figure 4.11.

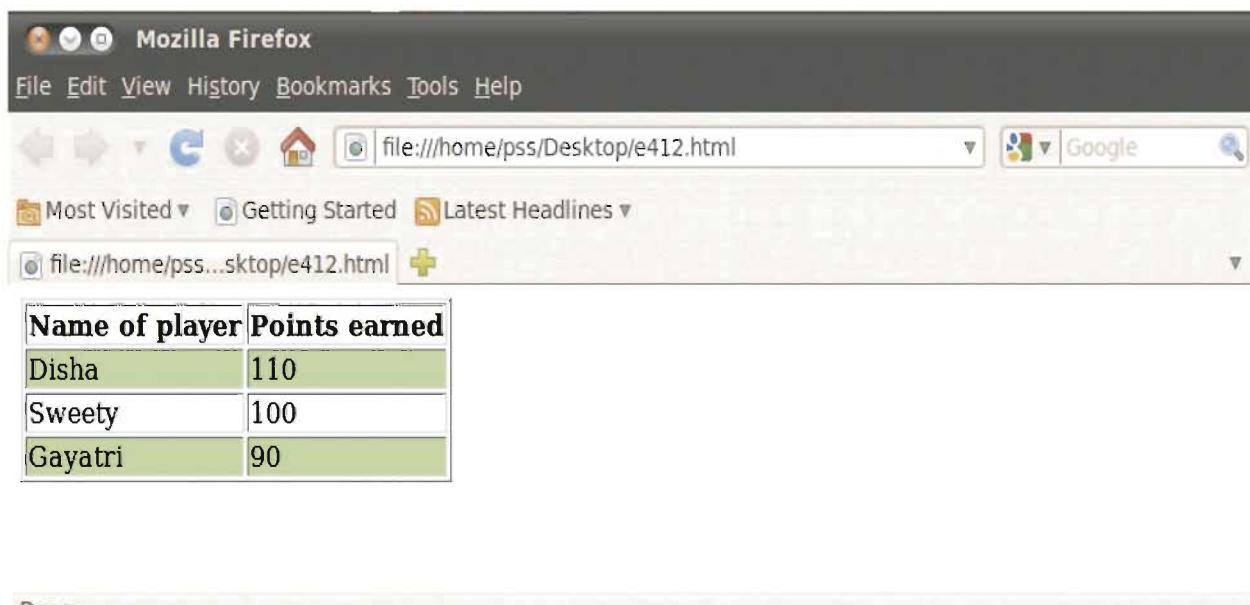


Figure 4.11 : Table with rows with colourful background

Valign Attribute

The valign attribute specifies the vertical alignment of the contents of each cell in a row. This can be done by using the general syntax as follows.

valign="Position"

Table 4.5 shows the possible values of the valign attribute.

Value	Description
top	Aligns content with the top of the cell.
middle	Aligns content in the center of a cell.
bottom	Aligns content with the bottom of the cell.
baseline	Aligns content so that the first line of text in each cell starts on the same horizontal line.

Table 4.5 : Possible values for the valign attribute

An example of tr tag is <tr width="150" valign="bottom">, embed it in the HTML code you are practicing and check it in a browser. Try HTML code given in code listing 4.8.

```
<html>
<body>
<!-- ----- -->
    <table border="2">
        <tr>
            <th> </th>
            <th> Monday </th>
            <th> Tuesday </th>
            <th> Wednesday </th>
            <th> Thursday </th>
            <th> Friday </th>
            <th> Saturday </th>
            <th> Sunday </th>
        </tr>
    <!-- ----- -->
    <tr align="middle">
        <th> 11 to 12 </th>
        <td> Maths </td>
        <td> Science </td>
        <td> Gujarati </td>
        <td> Maths </td>
        <td> Science </td>
        <td> Guajarati </td>
        <td> Assembly </td>
    </tr>
</table >
<!-- ----- -->
</body>
</html>
```

Code Listing 4.8 : An HTML code to create timetable

The code shown in code listing 4.8 creates two rows of a simple timetable for a class. If you see this code in a browser it will look as shown in figure 4.12. Do you think the table is incomplete? In that case, complete the timetable by creating some more rows.

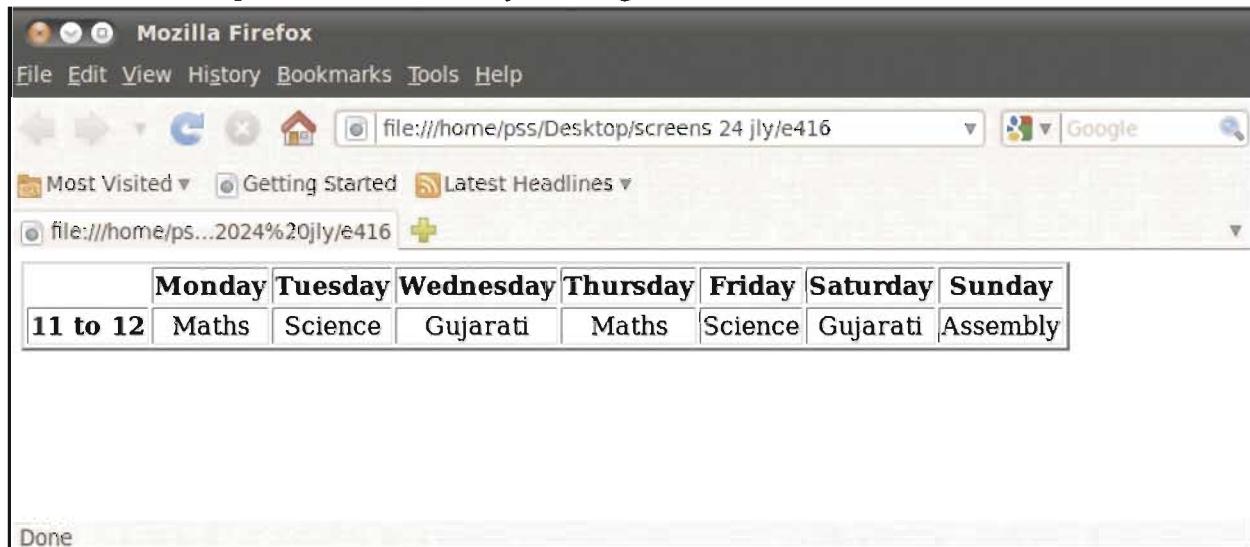


Figure 4.12 : Time table generation using HTML code

Cell Representation using th and td

Each cell in a table is represented by either a `<td>` or `<th>` tag. In a way they facilitate breaking of a row into multiple columns. If a heading is to be specified, a `<th>` tag is used. If table data is to be presented, `<td>` tag is used.

By default, content of a `<th>` tag is usually displayed in a bold font, horizontally aligned in the center of the cell. The content of a `<td>` element is displayed as left aligned normal text. Both the `<th>` and `<td>` tags can hold the same set of attributes. Effect of these attributes is limited to a single cell which carries them. Any effect these attributes cause, will override settings for the whole table or any upper level containing element such as a row. Effect caused by the `<th>` and `<td>` attributes is final.

In addition to the universal attributes and the basic event attributes, the `<th>` and `<td>` tags can also have the attributes shown in table 4.6.

Attribute	Description
<code>abbr</code>	Provides an abbreviated version of content of the cell.
<code>align</code>	Aligns content of the cell.
<code>bgcolor</code>	Adds background to the cell.
<code>char</code>	Manages the cell content to be aligned around the first instance of the specified character.
<code>colspan</code>	Indicates number of columns that the cell spans across.
<code>headers</code>	Indicate corresponding headers to the cell.
<code>height</code>	Specifies height of the cell.
<code>nowrap</code>	Stops text from automatically wrapping into a new line within the cell.
<code>rowspan</code>	Indicates number of rows that the cell spans across.
<code>valign</code>	Specifies vertical alignment of the cell.
<code>width</code>	Specifies width of the cell.

Table 4.6 : Attributes of `<td>` and `<th>`

We will learn about these attributes in detail as and when we proceed with the chapter.

Adding Caption to a Table

To give name to the table `<caption>` tag is used. This tag is required when you display a specific table along with the table name. Table name indicates what the table is for; for a result purpose, for timetable or for a diet chart. Most of the browsers display the contents of the caption above the table on a centered fashion.

Addition of the following line before the first row of any valid HTML encoded table generates table captions as "This is our time table".

`<caption> This is our timetable </caption>`

Code listing 4.9 shows how to create a time table using HTML.

```
<html>
  <body>
    <table border="2">
      <!-- ----- -->
      <caption> <h1> <font color="Brown" >
          This is our time table
        </font> </h1> </caption>
      <!-- ----- -->
      <tr>
        <th> </th>
        <th> Monday </th>
        <th> Tuesday </th>
        <th> Wednesday </th>
        <th> Thursday </th>
        <th> Friday </th>
        <th> Saturday </th>
        <th> Sunday </th>
      </tr>
      <!-- ----- -->
      <tr align = "middle">
        <th> 11 to 12</th>
        <td> Maths</td>
        <td> Science </td>
        <td> Guajrati</td>
        <td> Maths</td>
        <td> Science </td>
        <td> Guajrati</td>
        <td> Assembly </td>
      </tr>
    </table >
  </body>
</html>
```

Code Listing 4.9 : Adding caption to the table

Adding caption to the table increases the degree of understanding and user friendly presentation of table on web page. Figure 4.13 shows the output of code listing 4.9.



Figure 4.13 : Table with caption

Observe that the time table shown in figure 4.13 is incomplete. Students may complete the same and observe the output.

Nested Tables

Many times we need to add table within a table. The example shown in code listing 4.10 shows a nested table.

```
<html>
<body>
<!!-- ----- -->
<table border="4">
<caption> <h1>Conference Activities </h1></caption>
<tr>
    <th > </th>
    <th width ="40%"> Morning </th>
    <th> Afternoon </th>
</tr>
<tr>
    <th> Day 1</th>
<td> Inauguration </td>
<td> Key-note Address</td>
</tr>
<!!-- ----- -->
```

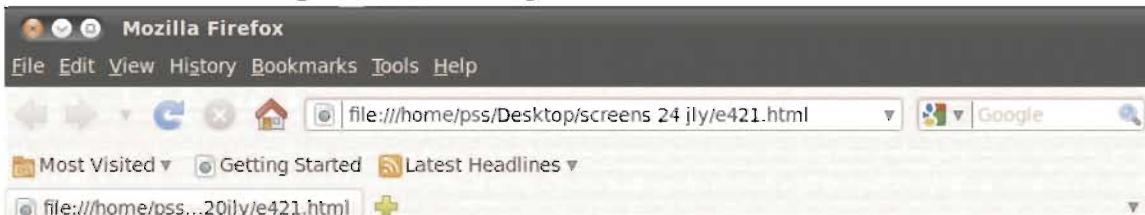
```

<tr>
    <th> Day 2 </th>
    <td> Paper Presentations</td>
    <td>
        <table border="1" bgcolor="pink" frame="box" align="right">
            <caption> <h3> Inner Table </h3></caption>
            <tr>
                <th> Teachers</th>
                <th> Students </th>
            </tr>
            <tr>
                <td> Meeting </td>
                <td> Quiz </td>
            </tr>
        </table>
    </td>
</tr>
</table >
<!-- ----- -->
</body>
</html>

```

Code Listing 4.10 : Nested table

Figure 4.14 shows the output of code listing 4.10.



Conference Activities

	Morning	Afternoon	
Day 1	Inauguration	Key-note Address	
Day 2	Paper Presentations	Inner Table	
		Teachers	Students
Teachers	Students		
Meeting	Quiz		

Done

Figure 4.14 : Nested tables

You might have noticed that the inner table has background colour and border. Both the outer as well as inner tables have their captions.

An Example of a Table

Figure 4.15 shows an example of a table that you have practically observed. It is about a menu of a typical Punjabi restaurant containing items of your choice along with prices.

Star Restaurant					
Starters		Roti and Bread		Curries	
Item	Price	Item	Price	Item	Price
Paneer Tikka Dry	100.00	Plain Roti	10.00	Paneer Bhurji	100.00
Panner kabab	100.00	Tandoori Roti	20.00	Mixed Veg	100.00
Drinks		Desserts		Water	
Item	Price	Item	Price	Item	Price
Tea	20.00	Gulab Jamun	30.00	Regular	---
Coffee	20.00	Rasgulla	35.00	Mineral	15
Koko juicy	25.00	Pudding	40.00	Sparkling	28
Mengo delight	25.00	Ice-cream	50.00		
The prices are in Indian rupees. Customers have to pay extra taxes. Please wait for 15 minutes after giving order.					

Figure 4.15 : Restaurant Menu card

Code listing 4.11 shows partial HTML code to generate restaurant menu card shown in figure 4.15.

```
<html>
  <head> <title> Star Restaurant Menu </title> </head>
  <body>
    <center>
      <table width="75%" border="1" bordercolor="#000000" bordercolordark="#000000"
      cellspacing="0">
        <tr align="center">
          <th align="center" colspan="6"> <h1> Star Restaurant </h1></th>
        </tr>
        <tr>
          <td colspan="2" align="center"> <h2> Starters </h2></td>
          <td colspan="2" align="center"> <h2> Roti and Bread </h2></td>
          <td colspan="2" align="center"> <h2> Curries </h2></td>
        </tr>
      </table>
    </center>
  </body>
</html>
```

```

</tr>
<tr>
    <td align="center"> <strong> <font color = "Blue" > Item </font> </strong> </td>
    <td align="center"> <strong> <font color = "Blue" > Price </font> </strong> </td>
    <td align="center"> <strong> <font color = "Blue" > Item </font> </strong> </td>
    <td align="center"> <strong> <font color = "Blue" > Price </font> </strong> </td>
    <td align="center"> <strong> <font color = "Blue" > Item </font> </strong> </td>
    <td align="center"> <strong> <font color = "Blue" > Price </font> </strong> </td>
</tr>
<tr bgcolor="#CCCCCC">
    <td align="center"> <strong> Paneer Tikka Dry </strong> </td>
    <td align="center"> 100.00 </td>
    <td align="center"> <strong> Plain Roti </strong> </td>
    <td align="center"> 10.00 </td>
    <td align="center"> <strong> Paneer Bhurji </strong> </td>
    <td align="center"> 100.00 </td>
</tr>
<tr>
    <td align="center"> <strong> Paneer Kabab </strong> </td>
    <td align="center"> 100.00 </td>
    <td align="center"> <strong> Tandoori Roti </strong> </td>
    <td align="center"> 20.00 </td>
    <td align="center"> <strong> Mixed Veg </strong> </td>
    <td align="center"> 100.00 </td>
</tr>
<!-- --Remaining part of the table may be completed using similar code.... -->
<tr align="center">
    <td align="center" colspan="6"> The prices are in Indian rupees. <br>
    Customers have to pay extra taxes. <br>
    Please wait for 20 minutes after giving the order.
</td>
</tr>
</table>
</center>
</body>
</html>

```

Code Listing 4.11 : Displaying restaurant menu card using HTML tables

Note that the HTML code given in code listing 4.11 displays a partial table. Students can complete the remaining part of the table by adding similar HTML code.

Frames in HTML

Frame in an HTML document is used to combine multiple web pages and display them as a single web page. Frames divide a browser window into several parts or sub windows, each containing an independent web page. By dividing the browser window in many frames, you can handle different HTML codes individually and manage loading and reloading of them. A collection of frames in the browser window is known as a frameset.

It is to be noted that some browsers do not support the frameset. To create a frameset document, first we have to create a `<frameset>` element, which is treated as container of different frames. The frameset defines the division of the browser window. Within the frameset, each frame is represented by a `<frame>` and `</frame>` tag pair.

Within the frameset, you may add `<noframes>` element, which provides an alternative message for users, when the browser used does not support frames. See the example given in code listing 4.12.

```
<html>
<head>
<title>Creating example</title>
</head>
<frameset rows="20%,60%,20%">
<frame src="top.html" />
<frame src="main.html" />
<frame src="bottom.html" />
<noframes>
<body>
    Your browser does not support frames.
</body>
</noframes>
</frameset>
</html>
```

Code Listing 4.12 : HTML code to create frames

Write the HTML code shown in code listing 4.12 and save it as frame.html. Create the required files such as top.html, main.html and bottom.html. Add appropriate contents of your liking within these files. Once you create these files we are ready to test them in a browser. Our output looks similar to the one shown in figure 4.16. Note that the output that you will get will differ based on the contents that you have added in the files.

From figure 4.16 it may be observed that the browser window is divided into three parts. The top portion of the window is painted with blue colour and it shows the contents of HTML file, top.html. The middle portion prints simple table for a restaurant menu. The middle portion is displayed through the HTML file called main.html. The bottom portion, painted with light yellow colour is displayed through the HTML file called bottom.html.



Figure 4.16 : Frames in HTML

Note that it is also possible to divide the browser window in vertical parts. We may use frameset element with columns (cols) as follows to split browser windows vertically

```
<frameset cols="25%,*,25%">
```

Here we have tried to divide the browser window in three vertical parts. Change the code `<frameset rows="20%,60%,20%">` in code listing 4.12 to `<frameset cols="25%,*,25%">` and try to see the output again.

Summary

In this chapter we have seen different ways to create lists and tables in HTML. We have seen the ordered as well as unordered lists with different attributes. The nested lists are also introduced in this chapter. We learnt to create simple tables as well as nested tables. The chapter has discussed attributes that can be applied to a whole table, attributes for individual table rows and attributes for individual cells.

EXERCISE

1. Explain how list can be defined in HTML document by giving suitable example.
2. Explain various types of lists in HTML.
3. Write a short note on nested lists in HTML.
4. Write a short note on table handling in HTML.

5. Choose the correct option from the following :

LABORATORY EXERCISE

1. Write an HTML code to define a table showing name of your friends and score in percentages they obtained in test of a school. Give appropriate caption and column heading for the table.
2. You may have seen your parent managing household expenses and incomes by writing accounts in a diary. Write an HTML code to prepare a table showing expenses and income for household activities. Include sources of income, major expenses and balance remained.
3. Write an HTML code to prepare the statement of marks (mark sheet) as shown in the first section of this chapter.
4. Write an HTML code to prepare time table of your class. Take hints from the time table shown in the chapter.
5. Write an HTML code to prepare a table for a multiplex cinema showing movies for its display as well as advertisement. The multiplex is having three screens and four shows on 9:00 am, 1:00 pm, 5:00 pm and 9:00 pm for all the screens. The late night show on third screen offers three different regional movies: one in Gujarati, second in Marathi and third in Telugu. Tentative look of the table is as follows. Add movies of your choice in each cell. Also give caption to the inner table as "Regional Movies".

Timings→	9:00 am	1:00 pm	5:00 pm	9:00 pm
Screen 1				
Screen 2				
Screen 3				Regional Movies

6. Write an HTML code to create list of the items that you want to shop. Categories items in groups such as books, vegetables, and food items. See the following sample framework which shows how the list should be.

List of items to be purchased

- I. Books
 - a. Textbook of science
 - b. Practice book of maths
 - c. Notebook
- II. Vegetables
 - a. Tomatoes
 - b. Spinach
 - c. Peas
- III. Food items
 - a. Chocolates
 - b. Butter
 - c. Bread



Introduction to Calc

Calc is an electronic spreadsheet package from the OpenOffice suits. Spreadsheet is a type of package that is used for fast and accurate calculations as well as formatting of data in a document. The spreadsheet package is used for managing financial and accounting documents, creating data reports, generating invoices, analysing data from scientific and statistical researches, and for doing variety of calculations on data. A spreadsheet program can also store, manipulate and create graphical representations of data.

A spreadsheet allows entering data in row and column fashion. You may recall; in your childhood, you might have used special notebook with small cells marked with rows and columns to practice mathematics. Spreadsheet is a long sheet of rows and columns on computer screen to do data analysis and calculation. In other words, a spreadsheet is a grid which interactively manages and organizes data in rows and columns. To facilitate the operations, spreadsheet packages also allow formulas besides data and formatting mechanisms. User can enter data interactively into a spreadsheet page, format it and calculate/analyse them for decision making. Besides user's data and formulas, spreadsheet packages also provide built-in formulas/functions for common mathematical, financial, statistical, and logical operations in very sophisticated manner. Because of these abilities the spreadsheet packages are used as a universal programme for structured data preparation and processing. LibreOffice Calc [www.libreoffice.org/], which closely resembles the Calc is also a free and open-source spreadsheet package.

Beyond a spreadsheet

As spreadsheets became larger, they became more difficult to manage. To handle with the increasing size of the spreadsheet, a concept of a workbook was identified. Main purpose of a workbook is to manage collections of spreadsheets. Besides workbook, online spreadsheets are also becoming popular.

Typical Applications of Spreadsheet Packages

Spreadsheet packages are widely used for data analysis and accounting applications. Table 5.1 presents a list of typical activities that can be generally done through a spreadsheet package.

Activity	Description
Balance sheet	Statements of financial position and summary in typical account format.
Result analysis and merit list preparation	Calculating, sorting and filtering results of some activities such as examinations.
Statistical data analysis	Sorting, arranging, and applying statistical operations such as finding mean, median, probability, etc.
Financial activities	Financial activities such as loan instalment calculation, interest calculation, etc.
Personal activities	Personal activities such as tracking personal weights, managing list of activities, items and events; such as preparing list of guests for a party or a forthcoming event.

Table 5.1 : Popular applications of spreadsheet

Getting Started with Calc

The first thing you require to know is how to open the Calc. Follow the command sequence Applications → Office → OpenOffice.org Spreadsheet as shown in figure 5.1.

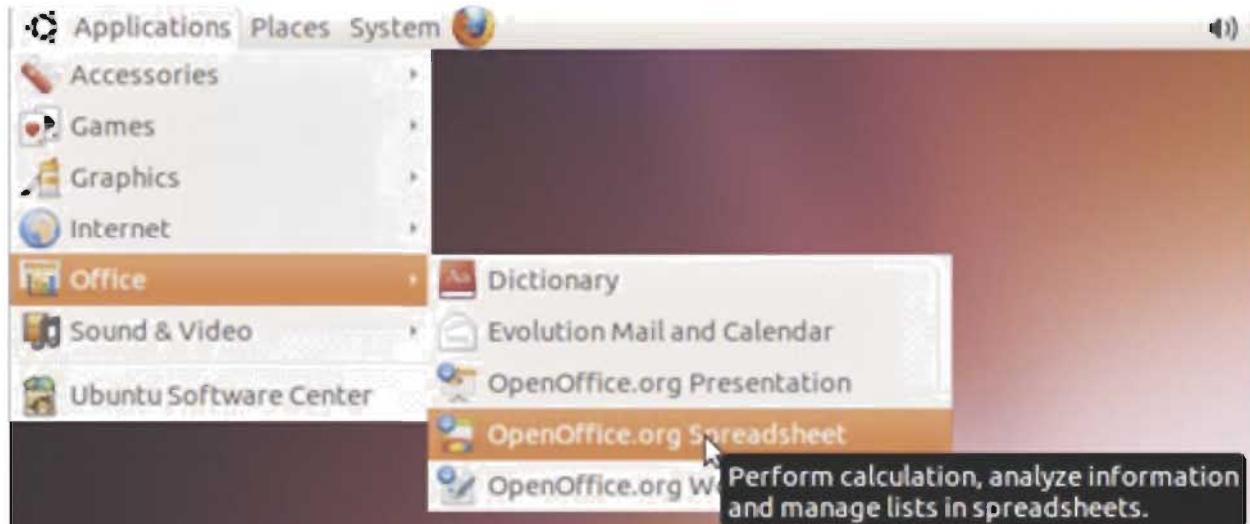


Figure 5.1 : Getting started with Calc

It will open an application document called a spreadsheet. A spreadsheet consists of multiple sheets; also called worksheets. By default, a new spreadsheet contains three worksheets, but the number of worksheets can be added or removed as per requirement. There are maximum 256 worksheets per Calc spreadsheet document.

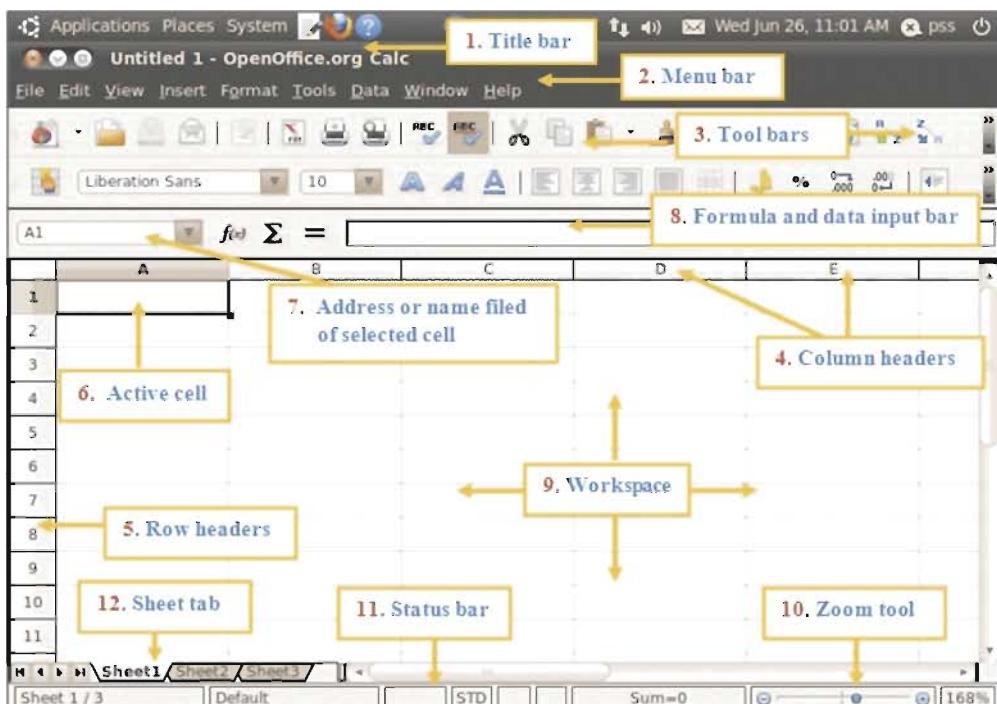


Figure 5.2 : An empty worksheet

Figure 5.2 presents an empty spreadsheet that you will see when you open a new document in the Calc. This is also known as a user interface of Calc. It shows different items with the number tags. Let us discuss them in detail.

Title Bar

The title bar is located at the top. It shows name (title) of the current spreadsheet. If you have just opened a new file, and not given name of the spreadsheet, it is an untitled spreadsheet. It is denoted as Untitled X, where X is a number. When you save a spreadsheet for the first time, you are asked to enter a name of your choice. It will let you know which workbook you are currently working with. Figure 5.3 shows the title bar.



Figure 5.3 : Title Bar

Menu Bar

The menu bar contains menus with commands for various tasks. Each menu item represents a separate submenu. This submenu is also called pull down menu. To view the menu, just put the mouse cursor over it, click once and submenu appears. Figure 5.4 shows a typical menu bar.



Figure 5.4 : Menu Bar

The main menu commands are listed in table 5.2 with brief description of each.

Menu	Description
File	Commands to operate the entire document such as creating a new document, saving file, printing, print preview, etc.
Edit	Editing commands such as copy, paste, find & replace, etc.
View	Adding or removing elements of the user interface, page break preview, etc.
Insert	Insertion of rows, columns, worksheets, elements, objects, etc.
Format	Formatting cells, grouping of elements, sorting, conditional formatting, etc.
Tools	Additional tools such as spellchecking, document protection, formulas, error correction, etc.
Data	Data processing, data sorting, data filter, etc.
Window	New window, freezing cells, list of open OpenOffice.org documents
Help	Help about function, information about applications and version of software, etc.

Table 5.2 : Menus in Calc

If you select particular menu, then its corresponding submenu appears. You may have noticed a triangle (►) shape along with options available in the submenu. When such sign of triangle (►) appears, there are more options available; from which you can select the suitable one. Figure 5.5 shows this situation. In figure 5.5 you can see the 'New' choice under the File menu with the triangle (►) shape showing more options. These options are presented in form of another black column, as a vertical pop down menu.

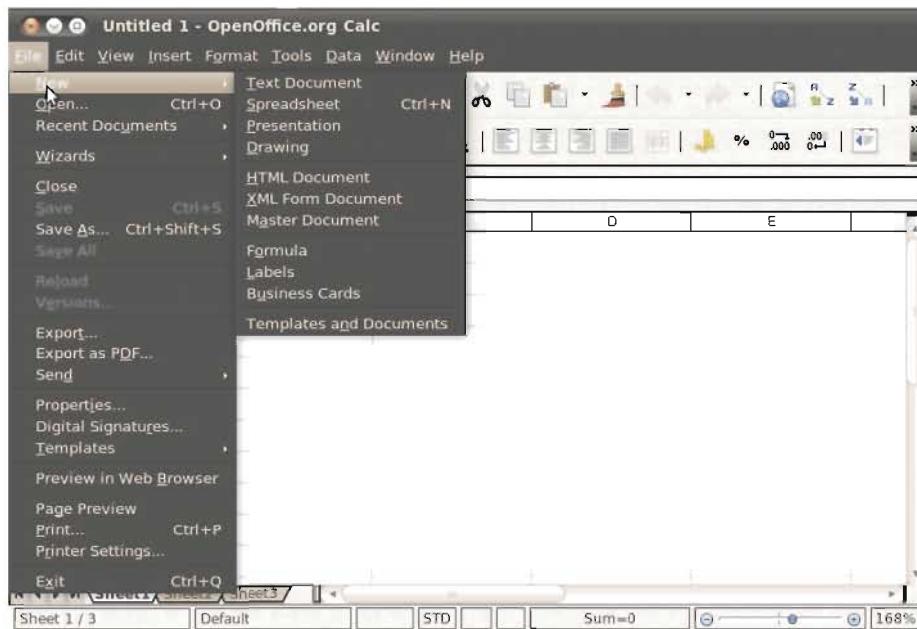


Figure 5.5 : Available options

If additional information is required to follow user's instructions, a dialog box appears indicated by (...) sign. Suppose you select **Save as** option; the Calc will further ask you about the filename and location where the file is to be saved as illustrated in figure 5.6.



Figure 5.6 : Save dialog box

Similarly if you see "**Print**" option, you may also see the sign of dots (...) at the end of "**Print**" option; which indicates that a dialog box will appear with various print options. We will see about these menus later in detail.

Toolbars

A toolbar presents the most common commands in form of onscreen buttons. Just by clicking the mouse pointer over a button, you can select the utility. In case, you do not know the purpose of the button, you have to hover (put) the mouse pointer on it. Immediately it will display name of the corresponding function. Toolbars can be turned off and on as per the need by selecting **View** → **Toolbars**. The default (system given) toolbars are as follows.

- **Standard Toolbar :** As the name suggests, this toolbar contains most frequently used standard commands from the File and Edit menus. Recently used icons are reflected in this bar. Most commonly you will see icons for creation of a new Calc document, opening an existing document, checking spelling within the document, cutting and pasting content and printing document etc. On this standard toolbar. Figure 5.7 displays a standard toolbar.

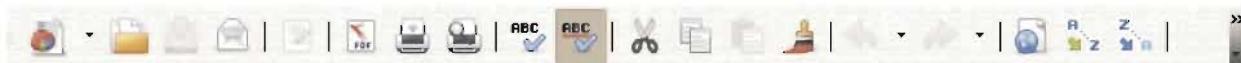


Figure 5.7 : Standard Toolbar

- **Formatting Toolbar :** The formatting toolbar presents the most frequently used commands for formatting content of a cell. This toolbar allows you to do formatting quickly. The icons in this bar will vary as you use the application. If any icon is not available, you may go to far end of the formatting toolbar and select the option to add an icon. Figure 5.8 illustrates a typical formatting toolbar.



Figure 5.8 : Formatting Toolbar

- **Formula Toolbar :** Formula toolbar presents facilities for entering and editing formula within a cell. A typical formula toolbar is shown in figure 5.9.

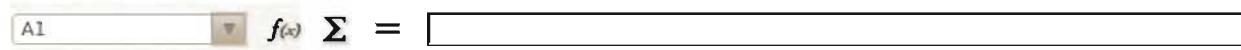


Figure 5.9 : Formula Toolbar

Buttons in toolbars can be modified. User can add buttons to the toolbar, remove them and adjust their sequence. You can add or remove toolbars as shown in figure 5.10.

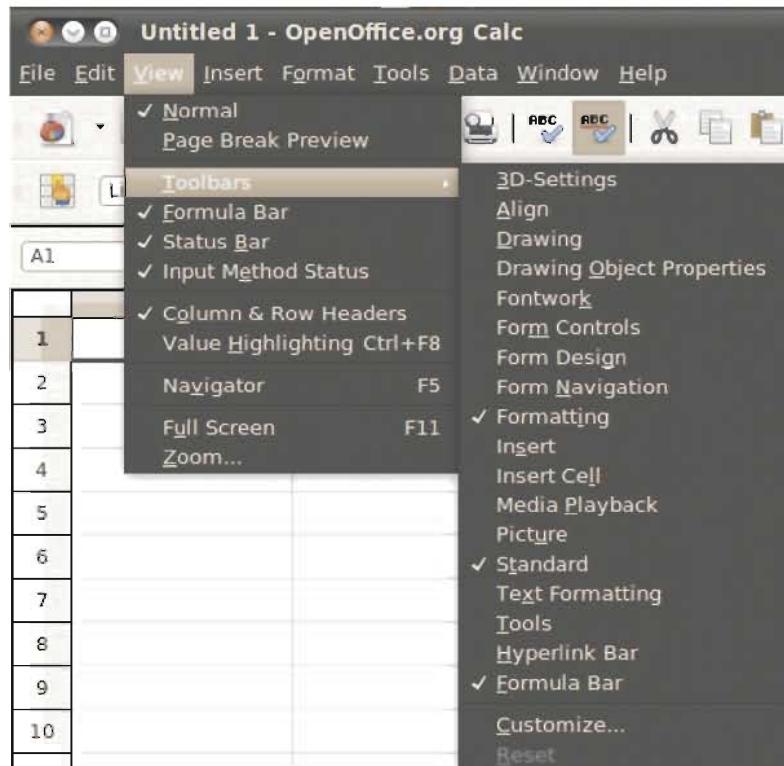


Figure 5.10 : Adding/Removing Toolbars

Rows, Columns and Cells

Each worksheet is divided into vertical columns and horizontal rows, forming cells. Both the rows as well as columns are numbered; columns with alphabet (and their combinations), and rows with numbers. Figure 5.11 highlights 3rd row in a worksheet.

Each **Column** in Calc is vertical series of cells. A column as a whole can be selected by clicking on the corresponding letter on the spreadsheet. Figure 5.11 shows 1st column in a worksheet.

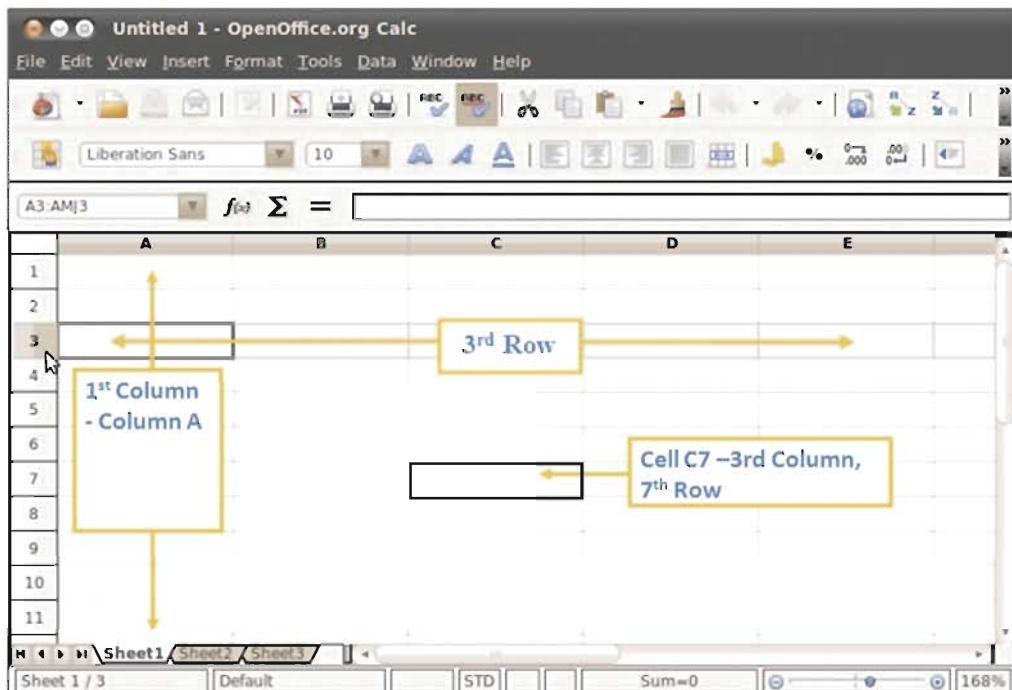


Figure 5.11 : Row in Calc

A cell is an intersection of a row and a column. It is identified by its column letter (letters) and row number, e. g. C7. Figure 5.11 displays cell C7. The cell which is currently selected is known as an active cell. Here C7 is shown to be an active cell. (Note that there will be a plus marker on bottom right side of the active cell in the actual screen). The square box appears here is an indicator of an active cell. The corner of the active cell is highlighted with filled small rectangle which is known as an autofill handle, which will be discussed in next chapter.

As we have a few alphabets (26), and number of rows is more, we need to use combination of alphabets as AA, AB, ..., AZ, BA,, BZ, CA,etc. The Calc contains 1,024 columns and 1,048,576 rows.

A cell is the basic element of a spreadsheet. Data, formulas are entered in a cell. That is, a cell holds individual elements such as text, numbers, and formulas. You can select multiple rows, columns or cells by clicking and dragging the cursor over the letters.

Formula and Data Input Bar

Check the big empty line on the top area of the worksheet, immediately below the toolbars. You have seen this in previous section as figure 5.9. This space is provided for user to enter data and formula. The data once entered, will appear in a particular cell.

The formula bar shows the cell, which is currently selected, on its left. The box indicated by fx on the right provides an area in which you can enter data or formulas into the cell.

Workspace

The empty grids of cells form an area where user's data appears. While entering data, you may directly click on a specific cell (to make it active) and enter data as well as formula. The cell may not show a formula, but displays its result in form of data.

Zoom Tool

The zoom tool can be used to zoom in/out a worksheet. You may zoom out your worksheet for better view. Figure 5.12 highlights the zoom tool from which you can observe that the worksheet is zoomed to 168%.

Scroll Bar

As the worksheet size is more than the computer screen (monitor) size, the Calc automatically provides you scroll bars on either sides of the sheet as shown in figure 5.12. That is, you can see vertical or horizontal scroll bar on screen. It is convenient to visualize data using the scroll bar, especially when the data do not fit into the screen.

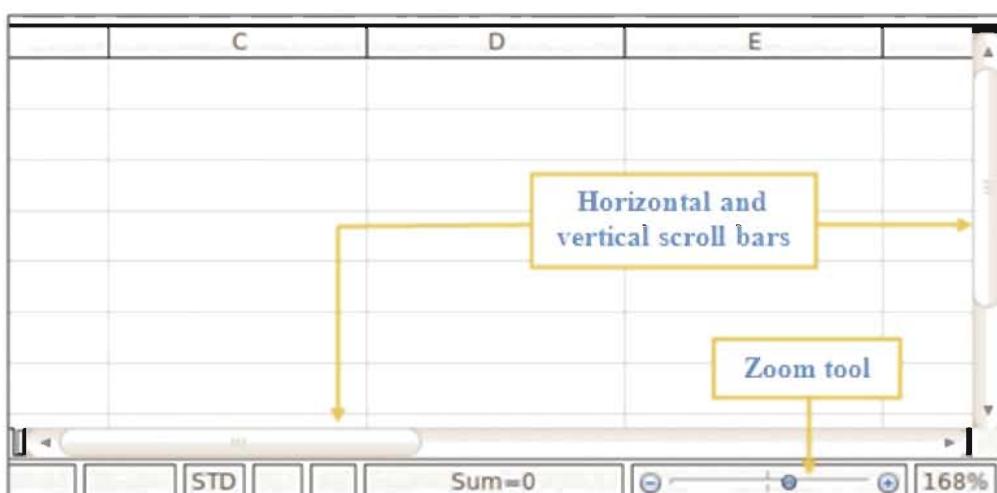


Figure 5.12 : Vertical and horizontal scroll bars with zoom tool

Status Bar

This bar shows present status of the document along with other data, such as the sum of the selected cells, page style, selection mode and unsaved changes. Figure 5.13 represents a typical status bar.



Figure 5.13 : Status Bar

Sheet Tab

Using the sheet tabs you can move between worksheets (also called sheets) of the document. The sheet tab serves as a navigation tool for the worksheets. The current worksheet tab name has the white background and other sheets have the grey background. You may just click on required worksheet in order to select it. You can also change name of any sheet just by right clicking on the text "Sheet1", choosing the "Rename sheet..." option, and typing the required name. Figure 5.14 displays a sheet tab. Alternatively, in specific situations, the arrows shown in figure 5.14 can also be used to move between the worksheets. You can jump to next (right), last (right most), previous (left) and first (left most) worksheet using the arrows.



Figure 5.14 : Sheet tab to select a worksheet

Creating a Calc Document

Let us create a simple document containing a shopping bill. The bill enlists company's name and address, items purchased, units of items purchased, price per unit, and total amount for the items. The bill also required to have grand total of prices, amount of tax, discount (if any), and total amount payable. The steps to generate the document are as mentioned:

Step 1 : Choose Applications → Office → OpenOffice.org Spreadsheet. This will open the Calc. You can see the user interface of Calc.

Step 2 : Observe that the Calc has created three worksheets automatically. However, as stated earlier, you can delete worksheets or add additional worksheets on need by just right clicking it. These three default spreadsheets are named Sheet1, Sheet2 and Sheet3. To give specific name to any of the worksheets, do the following :

- Select the worksheet that is to be renamed by clicking on the worksheet tab (figure 5.14) located just above the Status Bar.
- Click the Format menu, select Sheet from the menu options and select Rename from the submenu options available here as displayed in figure 5.15.
- Give appropriate name (say "bill") and click on OK to complete the operations. Notice the change in the worksheet name.

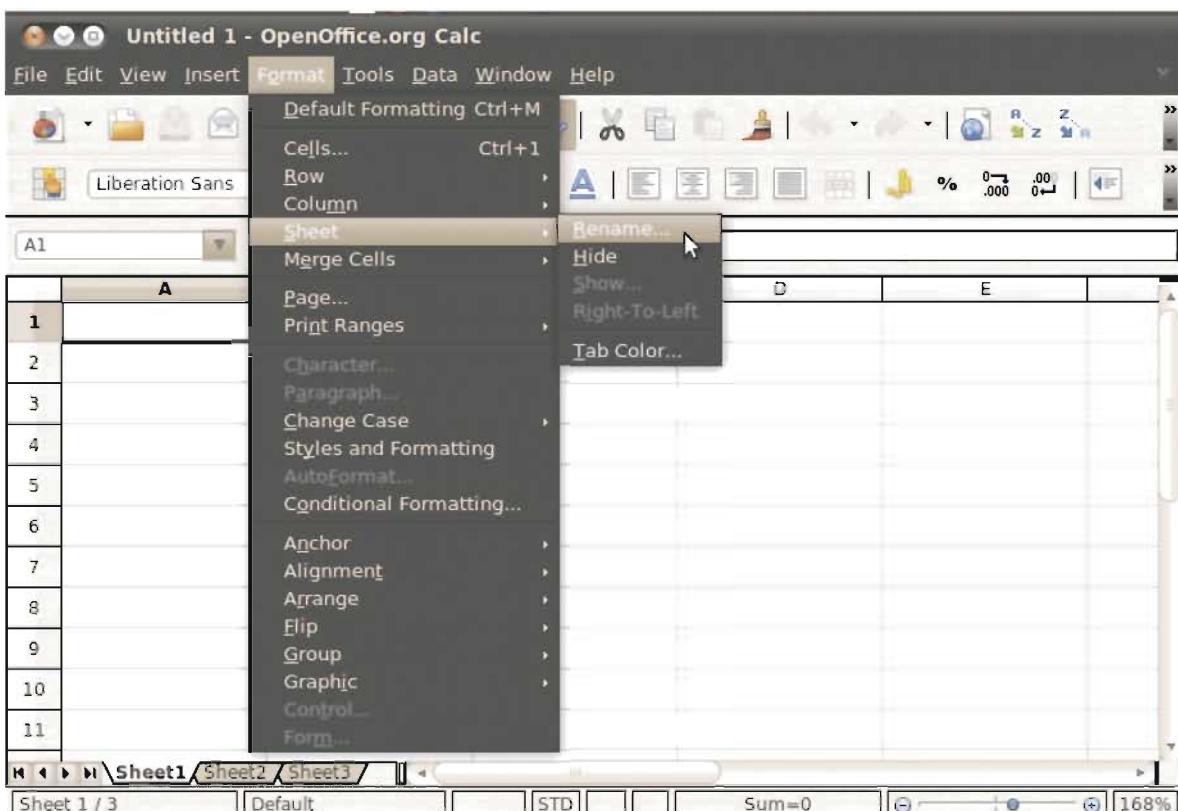


Figure 5.15 : Renaming sheet

Step 3 : Enter data given in table 5.3 into the worksheet named 'bill'.

Sr. No.	Item	Quantity	Unit price
1	Pen	5	10
2	Pencil	6	2
3	Pencil box	1	50
4	Notebook	10	20
5	Notebook cover	10	2

Table 5.3 : Shopping bill data

To enter the above data into the 'bill' worksheet do the following :

- Left-click on the cell **A1**. Enter the word "**Sr. No.**" and press the enter key.
- Left-click on the cell **B1**. Enter the word "**Item**" and press the enter key.
- Left-click on the cell **C1**. Enter the word "**Quantity**" and press the enter key.
- Left-click on the cell **D1**. Enter the word "**Unit price**" and press the enter key.

You may drag the column to resize it. It may be required to resize a column when you are entering name of items. To do so, put the mouse at the edge of the column, which you want to resize, and drag it to the required length. You may see that the first heading line is prepared. You may select the cells and make them bold using formatting toolbar.

Now enter the number "1" in cell A2, the word "Pen" in cell B2, the number "5" in cell C2 and the number "10" in cell D2 respectively. Similarly, enter the remaining lines. The worksheet will now look as shown in figure 5.16.

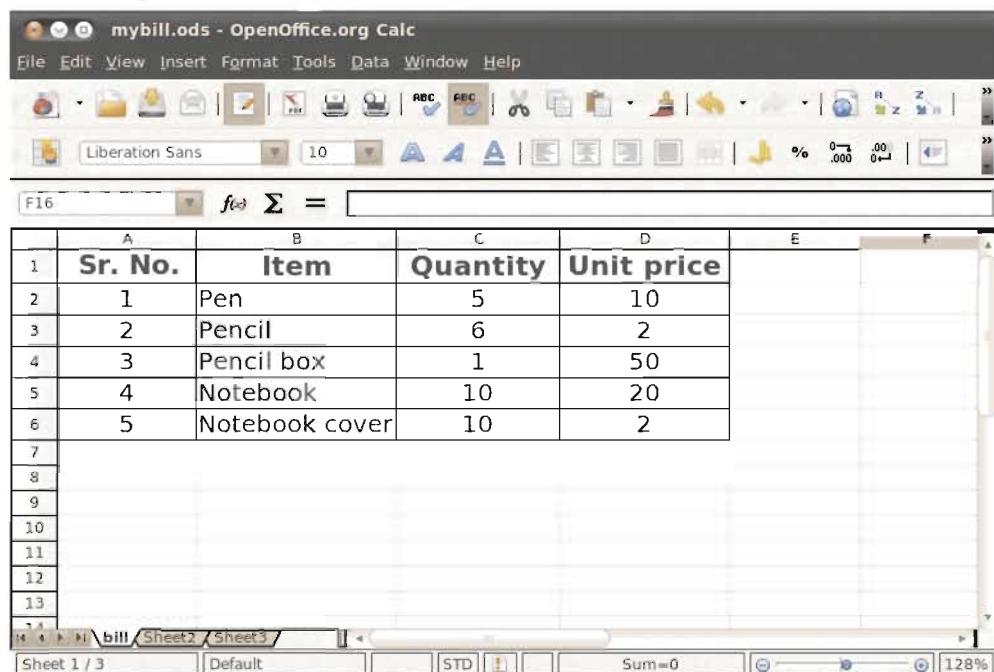


Figure 5.16 : Data entered in a worksheet bill

Step 4 : Save the worksheet with appropriate name so that the entered data is not lost accidentally. Perform the following command to save the worksheet.

- Click on **File → Save**, a dialog box as shown in figure 5.17 will appear; provide filename and path (where you want to store the file).
- Click on the Save button.



Figure 5.17 : Saving Calc document

The **OpenOffice.org Calc** saves a spreadsheet in a file format with the extension ods. You need to remember the location of the file, so that you can retrieve it in future.

We have just entered the necessary data and saved the file. We have not yet calculated the item-wise bill, and total bill for all the items, taxes and net amount payable. To calculate amount of one particular item (such as Pen), we need to multiply the item price (cost of a single Pen) and quantity (number of Pens) purchased. According to the first row, 5 pens of 10 rupees each were purchased. Total amount for the first item is 50 rupees. Similarly amount payable for other items are as follows :

1 st row → 5 pens 10 ₹ each	= 5*10 ₹	= 50 ₹
2 nd row → 6 pencils 2 ₹ each	= 6*2 ₹	= 12 ₹
3 rd row → 1 pencil box 50 ₹ each	= 1*50 ₹	= 50 ₹
4 th row → 10 notebooks 20 ₹ each	= 10*20 ₹	= 200 ₹
5 th row → 10 notebook covers 2 ₹ each	= 10*2 ₹	= 20 ₹

According to the above calculation, we can deduce a general rule that if x units of an item are purchased, each having unit price y, the amount payable is $x*y$. To insert this result in last column entitled as "**Amount**", we may manually multiply figures and write it in a cell. However, the better alternative is that we may enter a generic formula.

To enter a formula in cell we may provide direct data (such as =5*10) or we may provide cell reference where these data are available. That is if 5 is available at cell C2 and 10 is available at D2, in this case value of (=5*10) is equal to (=C2*D2). Typing '=' is necessary in beginning of any formula, failing to do so, the content is identified as regular text in a cell (even if a number is entered) and does not perform calculations.

Using cell reference instead of the direct value has some advantages. The first advantage is that we need not have to manually perform the arithmetic operation; which may lead to mistake. Second advantage is that when we change values in the cell, result of the formula will also automatically change. Third advantage is, when the data block is moved, or some rows or columns are inserted/deleted, the reference in formula will automatically change.

Let us add one column in our previous worksheet 'bill' to calculate item-wise amount to be paid. Do the following :

- Select cell E1 and enter the word "**Amount**". Press enter key. You may make the word bold.
- Select Add formula in cell E2 by typing =C2*D2 as displayed in figure 5.18.

	A	B	C	D	E	F
1	Sr. No.	Item	Quantity	Unit price	Amount	
2	1	Pen	5	10	=c2*d2	
3	2	Pencil	6	2		
4	3	Pencil box	1	50		
5	4	Notebook	10	20		
6	5	Notebook cover	10	2		
7						
8						
9						
10						
11						
12						
13						
14						

Figure 5.18 : Adding simple formula

- After entering formula in cell E2, when you press the enter key, you will see the result of multiplication in the cell.
- You may add formulas in cells respectively in cells E3 (=C3*D3), E4 (=C4*D4), E5 (=C5*D5), and E6 (=C6*D6).
- Another alternative is just drag the content of E2 to the remaining cells of the columns upto E6 cell. Automatically the formula will be copied. To drag the content of the cell, click on the cell. Drag the corner of the cell to the required number of cells.
- You may see the output as displayed in figure 5.19.

A	B	C	D	E	F
1	Sr. No.	Item	Quantity	Unit price	Amount
2	1	Pen	5	10	50
3	2	Pencil	6	2	12
4	3	Pencil box	1	50	50
5	4	Notebook	10	20	200
6	5	Notebook cover	10	2	20
7					
8					
9					
10					
11					
12					
13					

Figure 5.19 : Amount calculated for one cell and dragged to other cells

We also need to make grand total of all the payable amounts for different items. To make total of the amount column, SUM function is used. The SUM function button is located in the Function Bar at the top region of the screen. Using the SUM function button, you can automatically add the numbers in the cell range you select. Figure 5.20 shows the sum function.

A	B	C	D	E	F
1	Sr. No.	Item	Quantity	Unit price	Amount
2	1	Pen	5	10	50
3	2	Pencil	6	2	12
4	3	Pencil box	1	50	50
5	4	Notebook	10	20	200
6	5	Notebook cover	10	2	20
7					
8					
9					
10					
11					
12					
13					

Figure 5.20 : Sum function

Sometimes the symbol available in the top formula bar (denoted as ' Σ ') as displayed in figure 5.21 can be used.

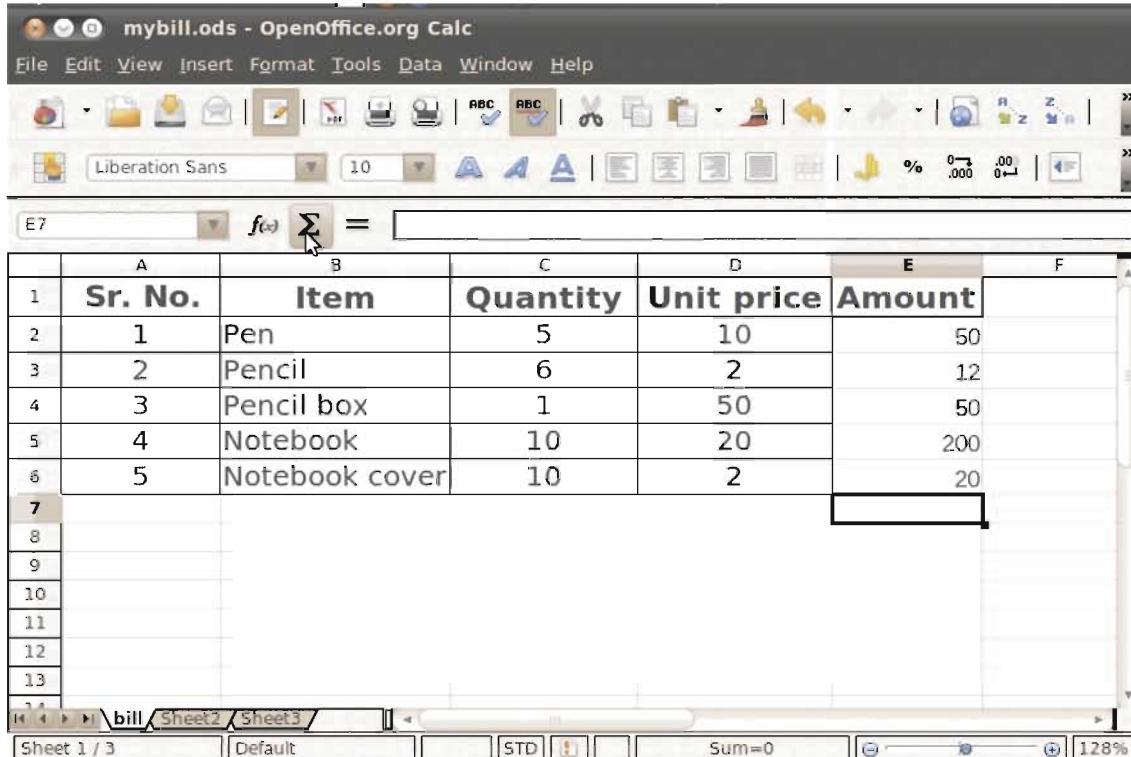


Figure 5.21 : Using autosum symbol

In spreadsheets, a cell range is denoted by the address of the first cell and the address of the last cell, separated with a colon; e. g. E2:E6 refers to cells from E2 to E6. Alternatively you can also select the cell E7 and type formula **=E2+E3+E4+E5+E6**. Select cell D7 and write caption "Total" and press enter. You will see screen as shown in figure 5.22.

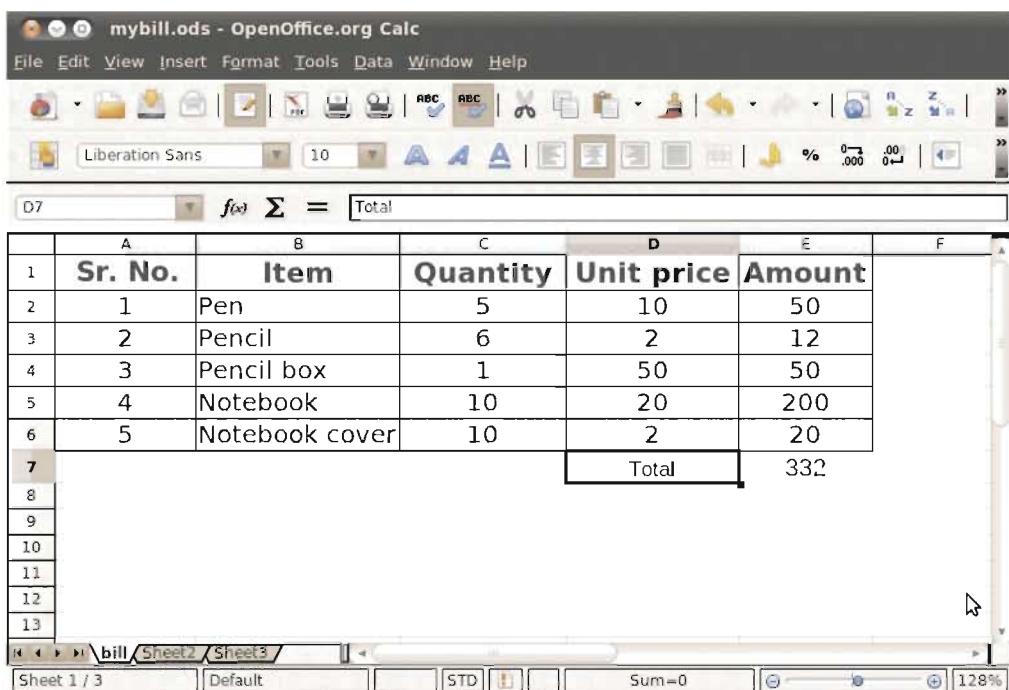


Figure 5.22 : Calculation of total amount

Let us now calculate the tax to be paid on the total amount calculated above, we need to calculate taxes. If we consider the tax is 5% of the total bill amount, then the formula to get tax amount is as follows.

$$\text{Taxes} = \text{Total amount} * 5/100$$

- Select cell D8 and write caption as "**Taxes**".
- Select cell E8 and write the formula $=E7*5/100$

To calculate the net amount to be paid, we need to add taxes to the total amount. That is the content of the cell E7 and E8 is to be added and displayed in the cell E9. You may use sum function as discussed above or may write direct formula ($=E7+E8$) by selecting cell E9.

Select the cell D9 and write caption "**Net Amount**" and press enter. You can see the result as displayed in figure 5.23.

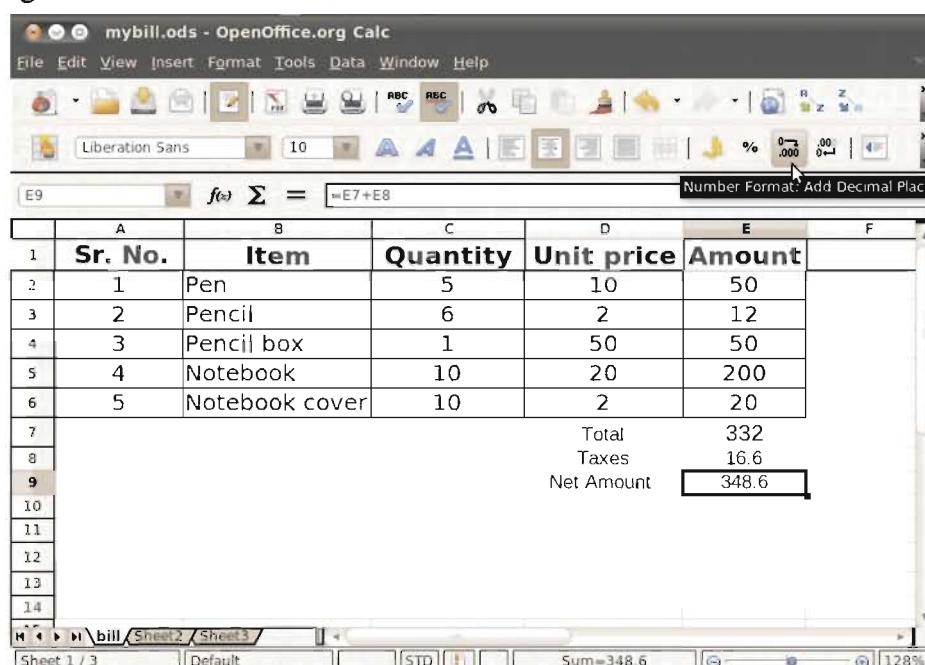


Figure 5.23 : Calculating net amount

You can see that the net amount calculated here 348.6. To make it more familiar, we may add a decimal point by clicking on toolbar as shown in the figure 5.23. Adding one decimal point will make the Net Amount value as 348.60.

You may want to move the whole content in such a way that you can insert name of the company and date. Perform the following steps.

- Using mouse, select all the cells containing data. In the above example, the cells that contain data are A1 to E9.
- Cut the selected data by selecting edit menu and choosing "**Cut**" operation.
- Go to cell B3 and paste the data.
- Ensure that your data is not changed.

You might have noticed the change in formulas. Total, Taxes and Net Amount now refer to F column instead of E column. Calc automatically changes the column references if the cells are moved from one location to another location unless specifically told not to do so. In next chapter we will learn more about such relative or absolute (fixed) addressing techniques.

Saving and Re-opening File

While working with a spreadsheet (or any other computer application), frequently you need to save your work. There might be a power fluctuation, or any other such problem; because of which you may lose the data you have entered. Saving the file frequently is a good practice and prevents accidental loss of data.

Once operations of files are finished, you need to close the file. You may click on the cross button on the top of the window or select File menu and choose Close operation. To reopen the file again, you need to select File menu and choose Open operation. You may take help of standard tool.

Meet the Developers of the OpenOffice Suit via Calc

Now you know how the spreadsheet in OpenOffice works. Do the following. Open a workbook in Calc. Select any cell and type =starcalcteam() in any of the cell, which will display the picture of OpenOffice development team members. Figure 5.24 displays the Calc development team.

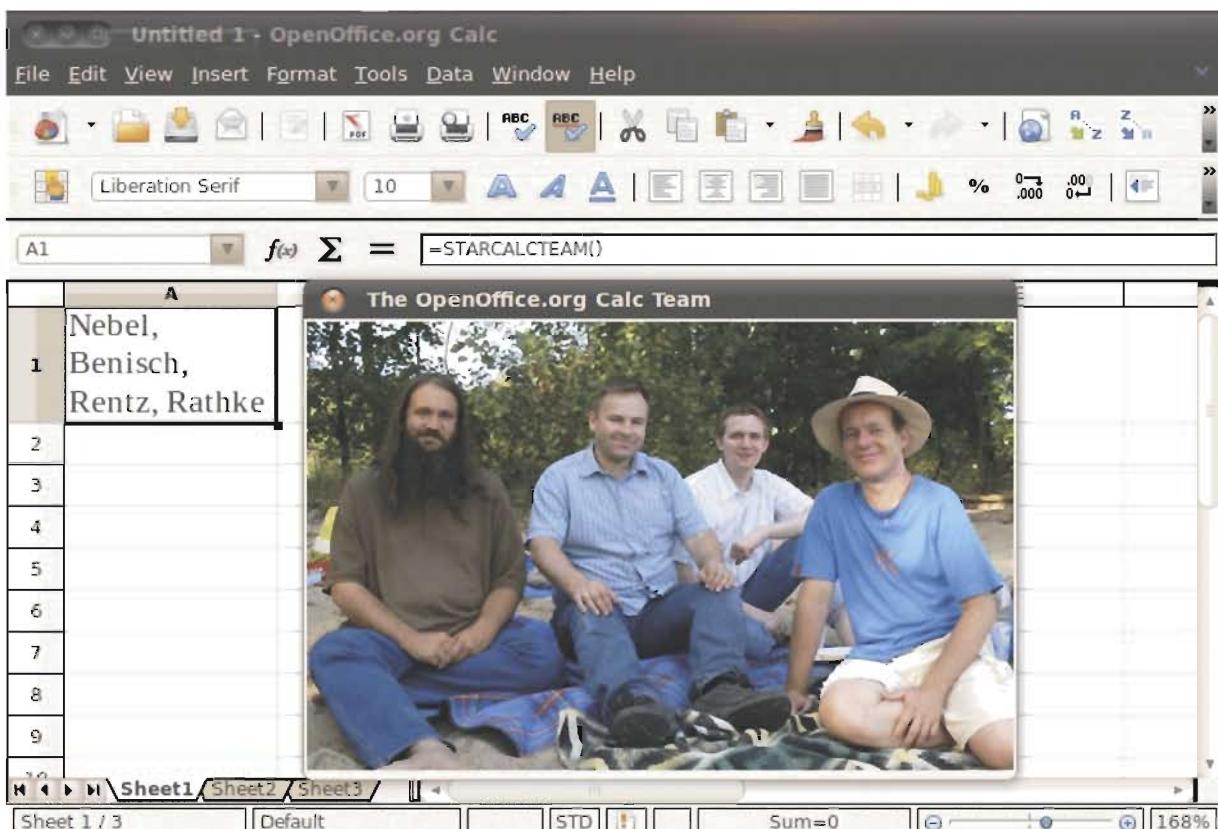


Figure 5.24 : Calc developer's team

Summary

In this chapter we have learnt what is spreadsheet and typical applications that can be developed using such packages. The chapter has focused on introduction and basic functionalities of a spreadsheet package from OpenOffice suit. The features and functionalities discussed have been illustrated with help of a real life example in step by step manner. With the concepts learnt in this lesson you may create a new spreadsheet, edit data in it, analyse and process the data with some basic formulas, save it and reopen it whenever required.

EXERCISE

- 1.** List suitable applications of spreadsheet packages.
 - 2.** Explain file saving and reopening in Calc.
 - 3.** Explain working of formulas in Calc worksheet.
 - 4.** Can we rename an exiting sheet of Calc ? How ?
 - 5.** What would you do if a standard toolbar is not visible in a Calc document ?
 - 6.** Choose the correct option from the following :
 - (1)** Which of the following type of package does Calc refer to ?

(a) Spreadsheet	(b) Multi-sheet
(c) Double sheet	(d) Cannot be determined
 - (2)** Which of the following applications are not suitable for Calc ?

(a) Balance sheet preparation	(b) Result analysis
(c) Presenting an idea about a product	(d) All of these
 - (3)** Which of the following is the extension of a worksheet created in Calc ?

(a) .ods	(b) .odd	(c) .xls	(d) .obj
----------	----------	----------	----------
 - (4)** Which of the following will be inserted in a worksheet if =starcalcteam() is inserted in a Calc cell ?

(a) Stars	(b) Photograph of Calc developer team
(c) Calc licence information	(d) Calc version information
 - (5)** How can one calculate total of values entered in a worksheet in a Calc document ?

(a) By manual entry	(b) By autosum
(c) By formula	(d) All of these
 - (6)** If we move a cell containing a formula having reference to another cell in the worksheet what will happen to the cell numbers used in formula ?

(a) The cell row and columns are changed at destination	(b) The cell row numbers are changed at destination
(c) The cell column numbers are changed at destination	(d) Nothing will be changed

LABORATORY EXERCISE

1. Implement the example of the shopping bill discussed in this chapter.
 2. Study any bill you get from a super store and implement it in Calc.
 3. Generate a Calc document having your marks of six different subjects. Make total of the marks, find out average and percentages from the data.
 4. Extend the third example given in this exercise by adding marks of your friends.





Data Editing and Formating in Calc

As discussed in the earlier chapter, basic unit of data storage in a spreadsheet is a cell. A spreadsheet is made up of rows and columns intersecting each other forming multiple cells. All the data, formulas and functions are to be written within these cells. This chapter gives you brief outline about the basic operations regarding data editing and formatting.

Basic Worksheet Operations in Calc

Operations such as opening a new or an existing spreadsheet, saving the spreadsheet, renaming the whole spreadsheet with save as option; renaming work sheets, adding/deleting worksheets etc. are frequently used operations on spreadsheet or work sheet level. This section discusses the commands for the same.

Creating a New Document

When you open the OpenOffice.org Calc application, it will automatically open a new (empty) Calc document. If you have already used the document and wrote some data in it, you may open another new spreadsheet document. To open a new Calc document, that is a Calc spreadsheet, perform the following procedures.

- Select **File → New → Spreadsheet;** from the Menu Bar.
- Press **CTRL + N;** or
- Click on the **New Document Icon** on the toolbar at the top of screen and select Spreadsheet. The icon (encircled) will look as shown in figure 6.1.

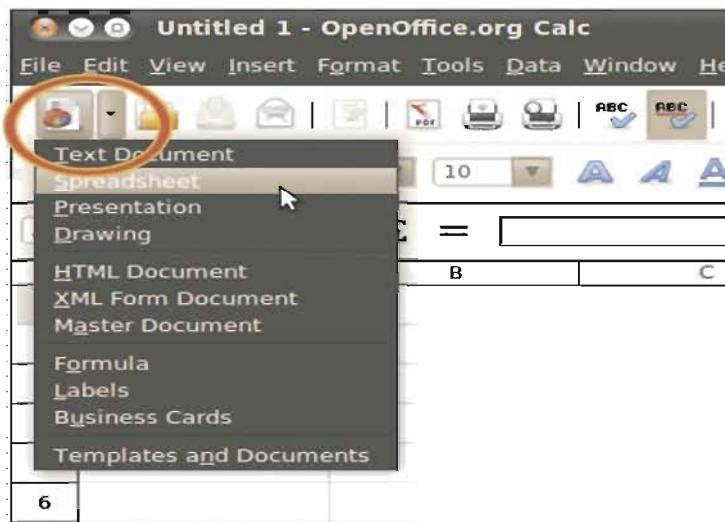


Figure 6.1 : An icon for opening new spreadsheet

When a new spreadsheet is opened, it will give you three separate worksheets; just like you have purchased a notebook for mathematics with only three pages having grid of rows and columns. In Chapter 5 you have learnt how to open a spreadsheet with three worksheets, for entering data of a shopping bill.

An existing Calc document can be opened in multiple ways. The first way is to open the Calc first and go to file menu and select open. See the following command sequence.

- Select **File → Open;**
- In the **Open** dialog box, select the required file from a default folder or any other specified folder;
- Complete the action by clicking on the **Open** button.

Another alternative is first to find the required file from computer and then open it within the Calc software as follows.

- Open the folder **Documents** by choosing **Places → Documents;**
- Select the required spreadsheet file and open it.

Still if you cannot find your file by performing the steps mentioned above; you can find the file through search utility of Ubuntu. This utility is helpful, especially when you do not remember the location of the required file. To do so, perform following actions.

- Select **Places → Search for Files** as shown in the vertical menu in figure 6.2;
- When you click **Search for Files** (see mouse arrow in figure 6.2), a dialog box appears asking for file details. The details include name of file, folder in which the file is to be searched and some content within the expected file. We can also search file by specifying some dates like last modified date. See block arrow pointing to the dialog box in the figure 6.2;
- In the **Name contains:** field of the **Search for Files** dialog box, enter a full or a part of the file name;
- Click on the Find button.

These steps are illustrated in figure 6.2. It will show a list of files as a result of the search operation. To open a file, double click on the required filename in the list using mouse.

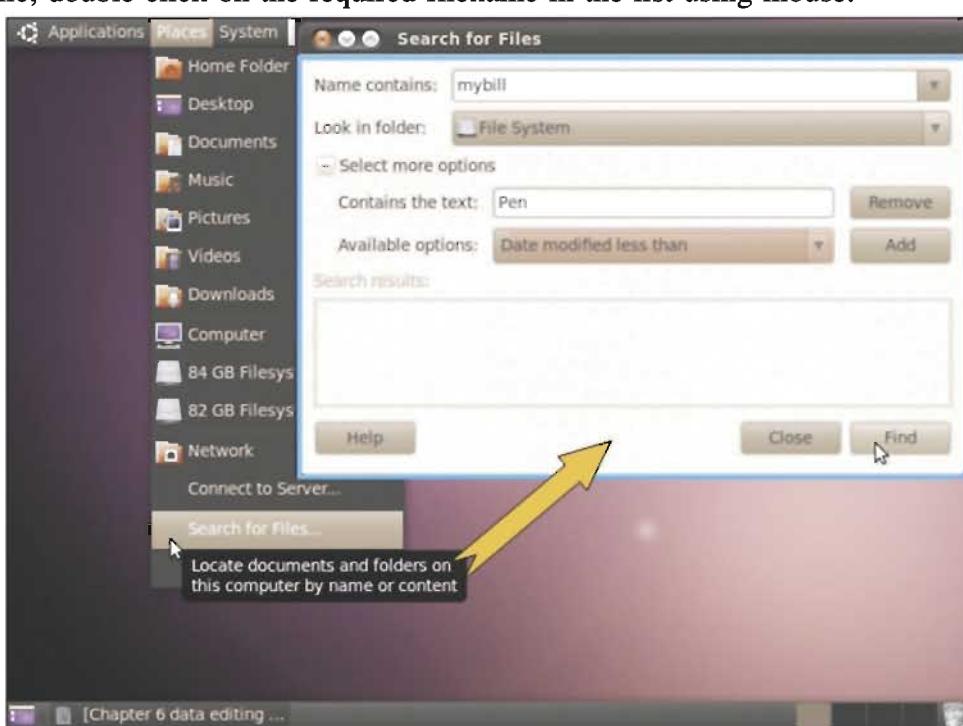


Figure 6.2 : Result of search operation to find a file

Saving and Closing Spreadsheet

You have already opened a new or an existing spreadsheet document for viewing and editing in the previous section. Once you finish working with the opened spreadsheet, you need to save the changes. It is advisable to frequently save the changes to prevent any loss of data.

We have already discussed how to save a worksheet in Chapter 5. To save spreadsheet choose, **File → Save**. You can also choose Save as option; if you want to save the document with different name or different type. As stated earlier, if you are saving the spreadsheet first time, you will be prompted to give file name and path.

To close the current file, choose **File → Close**. If you have not saved the latest changes, the Calc will give you an opportunity to save the file before closing it.

Saving Worksheet in Different Format

By default, the Calc saves the spreadsheet in file format with extension ods. Some of the other formats are listed in table 6.1.

Format	Extension	Description
ODF Spreadsheet	ods	OpenOffice.org Calc format.
ODF Spreadsheet Template	ots	Calc spreadsheet template format.
dBASE	dbf	Database file format.
Text CSV	csv	Text file containing comma-separated values; such files are typically used for data exchange among various programmes.
HTML Document (OpenOffice.org Calc)	html	Web page format.
Portable Document Format	pdf	Most frequently used format; this is a universal Adobe format of a portable document.
Microsoft Excel 2007 XML	xlsx	Spreadsheets of MS Office 2007/2010.
Microsoft Excel 2003	xls	Spreadsheets of MS Office 2003.

Table 6.1 : File formats in which a worksheet can be saved

To save a spreadsheet in other file format :

- Go to File and choose Save (for unsaved document) or the Save As option (for opened and already saved document);
- Open the file type menu (encircled) as shown in figure 6.3;

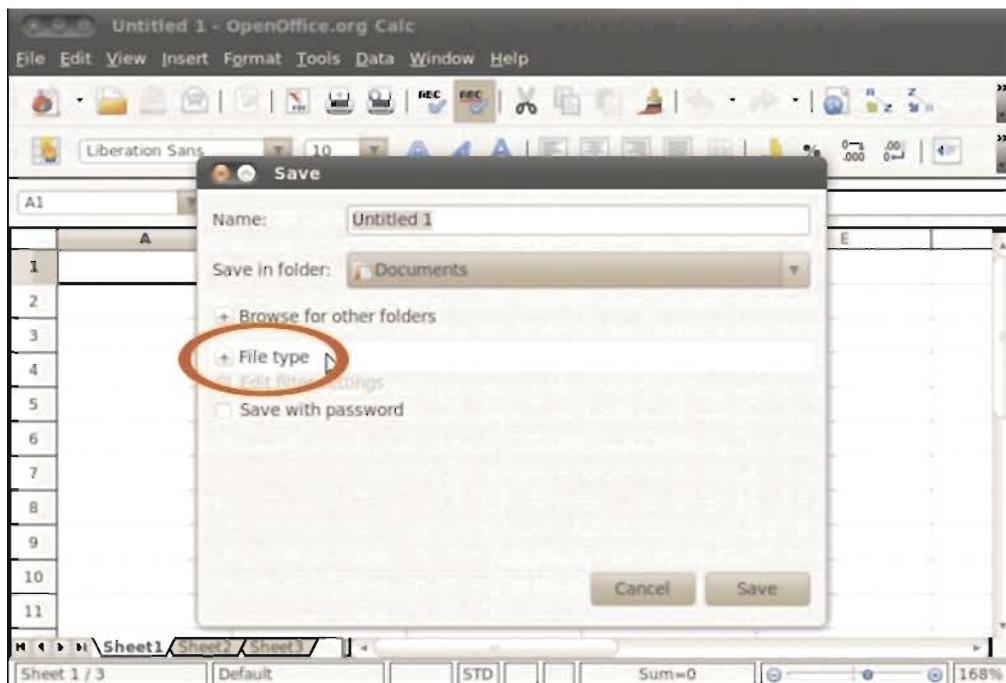


Figure 6.3 : Opening file type menu

- Choose required file format as shown in figure 6.4.

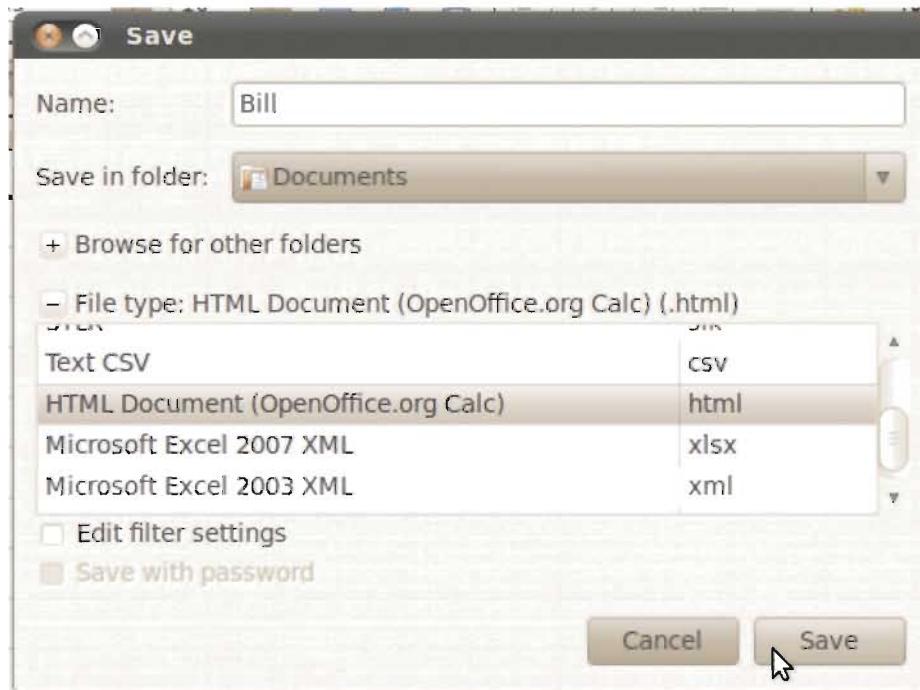


Figure 6.4 : Choosing file type

- Save the changes by clicking on the **Save** button.

Renaming and Re-colouring Worksheet

In the previous chapter we have changed the name of a worksheet by selecting a worksheet from status bar (right click) and changing its name. If you have many worksheets in a spreadsheet document, better to assign a suitable name (and colour) for easy management of the document. To change the name of a worksheet, perform following actions.

- Select any cell in an opened worksheet.
- Select **Format → Sheet → Rename;**
- In the dialog box **Rename Sheet**, enter a new name;
- Save changes by clicking on the **OK** button.

Alternatively, click the right mouse button on the name of a worksheet shown on the sheet tab and select **Rename sheet** option as follows. You have already learnt this option. The method is given below :

- Go to appropriate sheet, which you would like to rename. The selected sheet must be highlighted (with white background);
- Right click on the sheet. A vertical menu appears as displayed in figure 6.5. From the menu, select **Rename Sheet** option; A new dialog box appears as shown in form of rectangle in middle of the figure 6.5;
- Provide new name of the sheet.

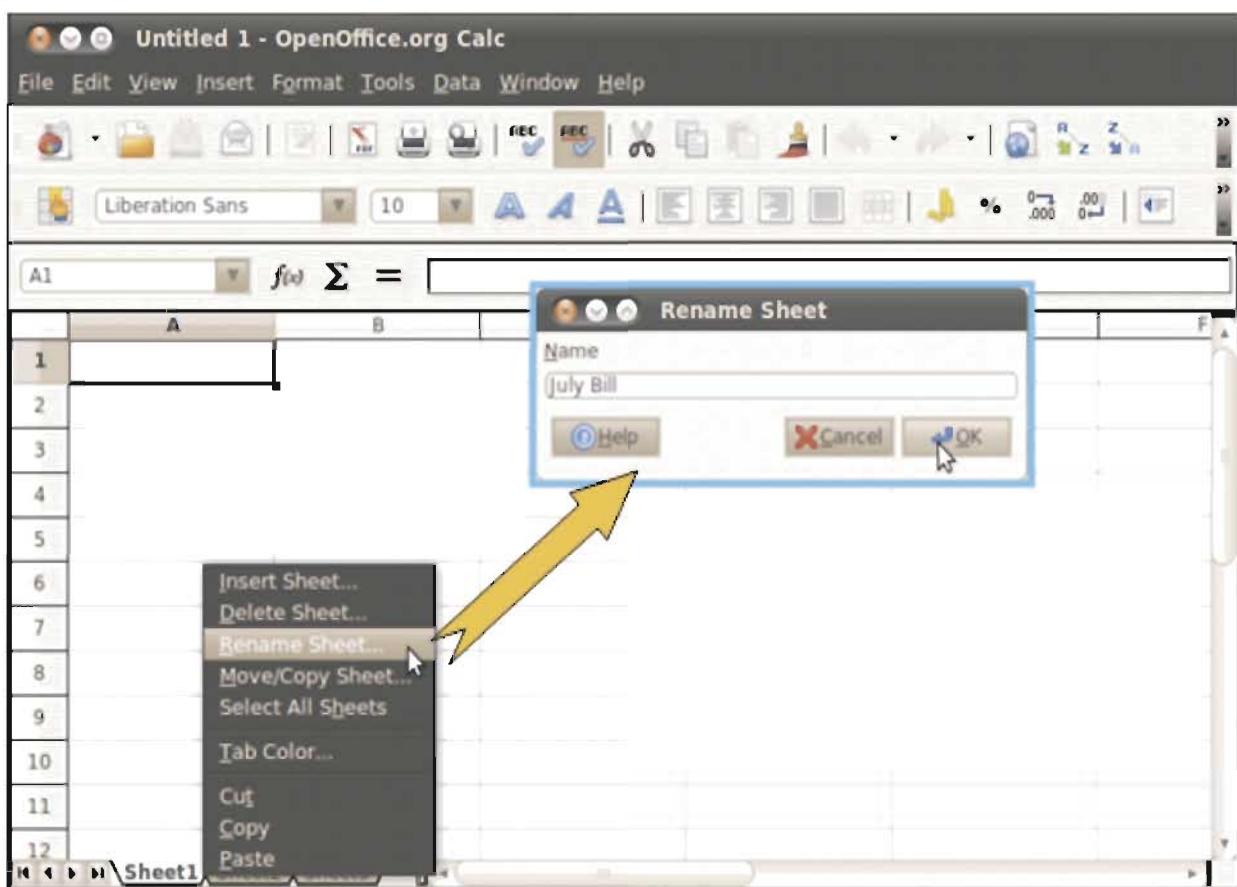


Figure 6.5 : Renaming worksheet with sheet tab

Changing Colour of the Worksheet Tab

To change the tab colour of a worksheet, we just need to right click the worksheet name and choose the command **Tab Color** from a vertical menu appears as shown in figure 6.6. We need to select an appropriate colour for the tab from the colour choices presented to us.

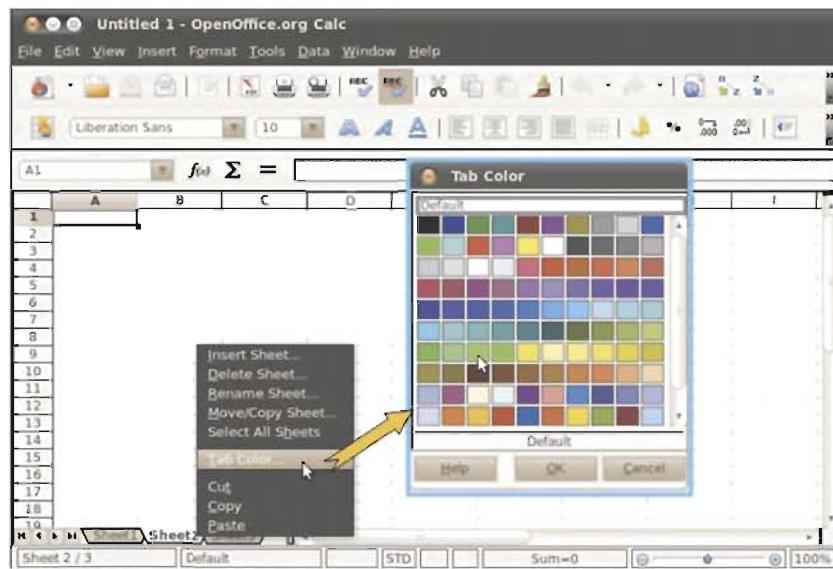


Figure 6.6 : Changing colour of a worksheet tab

Inserting and Deleting Worksheets

As you know, many times three worksheets provided by the Calc may not be sufficient to enter all the data related to the application. In such cases we may need to insert additional worksheets. For example, there are five groups of students in an activity, say sports. For each of the group, an attendance report is to be made. In this case, it is better to have five worksheets; each having an attendance report of an individual sport. Similarly, rows, columns can be added to a formatted spreadsheet, width can be changed and merging cells can be executed.

To insert a new worksheet do the following actions.

- Select any cell in a worksheet before or after which a new worksheet is to be added;
- Select **Insert → Sheet**. An Insert Sheet dialog box will appear;
- In the Insert Sheet dialog box, specify additional options.
- At the end, click **OK** button to confirm action.

Figure 6.7 illustrates above actions for inserting an additional worksheet.

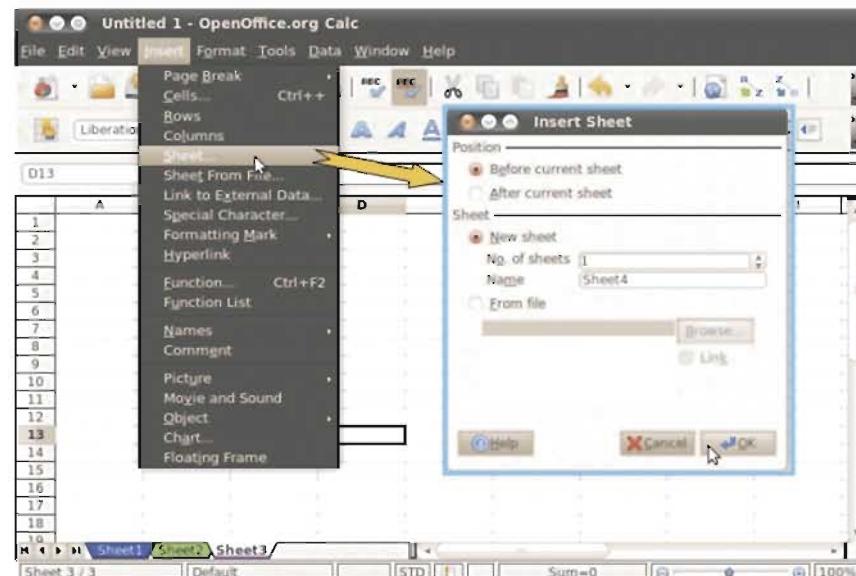


Figure 6.7 : Adding new worksheet in a document

You might have also thought the possibility of inserting a worksheet through the worksheet tab. To insert a worksheet using the sheet tab, perform the steps mentioned.

- Select a worksheet and right click on it;
- A vertical menu will open;
- Select **Insert Sheet**;
- You will see a new Insert Sheet dialog box;
- Select required options and click OK.

Deleting Worksheet

You can delete one or more worksheets in Calc. To delete a single worksheet, right click on the sheet tab of the worksheet you want to delete. It will give you a list of possible actions. Select **Delete Sheet** from these choices. Alternatively you can select **Edit → Sheet → Delete**.

Row and Column Level Operations

Sometimes we need to add some columns or rows in a worksheet. Though the worksheet provides many rows and columns, it may be possible that we would like to add some blank rows or columns in the middle of the data already entered. This section discusses how to add columns and rows in a worksheet along with other operations at row or column level. To insert a row do the following:

- Select a cell in the row above which a new row is to be added;
- Select **Insert → Rows**.

To insert a column do the following:

- Select a cell in the column besides which a new column is to be added;
- Select **Insert → Column**.

Width of a Column and Height of a Row

Calc presents all rows and columns with same height and width. When you write some text in a cell, often the cell content becomes invisible due to the adjacent cell content. Also at times result of some formula may not fit into the cell. To change the width of a column or height of a row, do the following.

- Click on the dividing line between columns (rows) headers;
- Without releasing the left mouse button, drag it in the direction needed.

To provide specific size to a column or a row, do following :

- Select a cell from a column (row);
- Select **Format → Column → Width** or **Format → Row → Height**;
- In the dialog box that appears, set the width of the column or height of the row in inches;
- Click on **OK** button.

If you are not comfortable with inches, go to **Tools → Options → General**. Modify the options provided by the category as per your need as illustrated in figure 6.8.

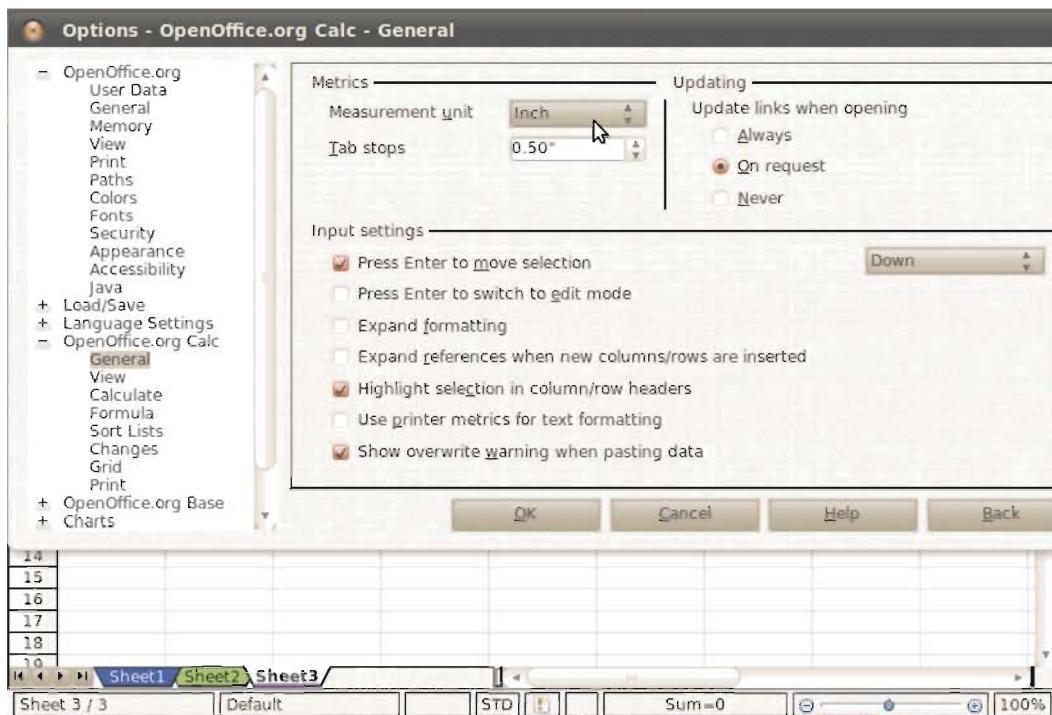


Figure 6.8 : General options of Calc

Deleting Rows and Columns

To delete a row, perform following steps:

- Go to a specific row which you want to delete;
- Right click on the row;
- Select **Delete Rows**.

In a similar way you can delete a column also. Figure 6.9 demonstrates delete operations for rows and columns.

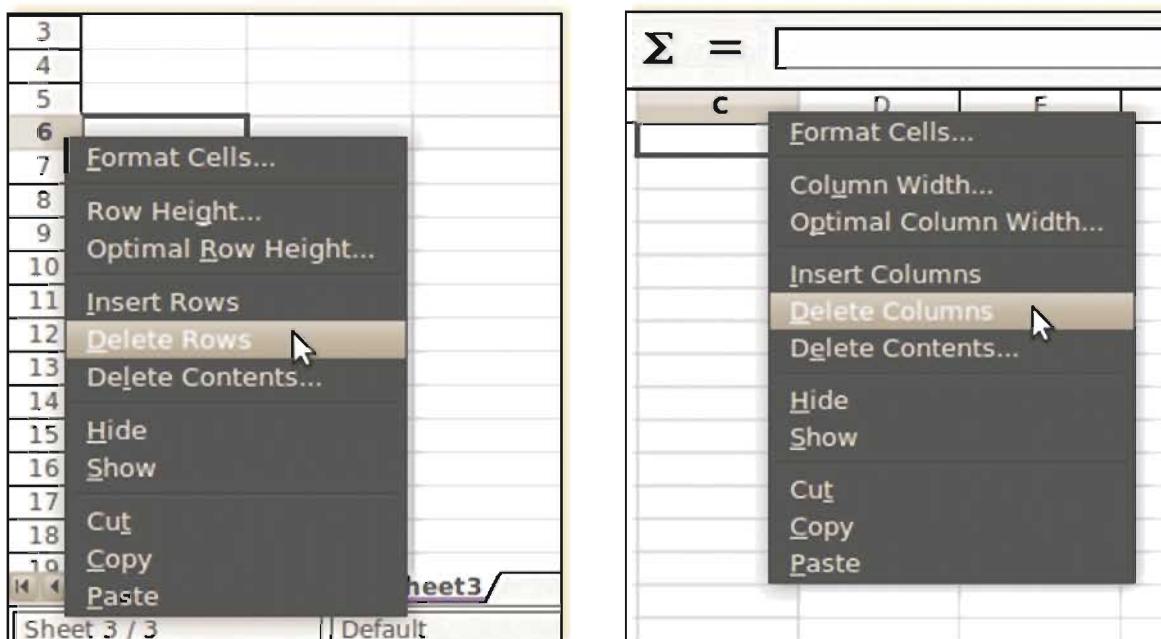


Figure 6.9 : Deleting rows and columns

Alternatively, to delete a row or a column, you may perform following steps:

- Select a column or a row;
- Select **Edit → Delete Cells**.

This is illustrated in figure 6.10. It is possible to delete entire column or row and shift cells as per our need using this option.

Selecting Rows and Columns

To select an entire column, click on the column header; you may see that the column is highlighted. Similarly, to select an entire row, click on the row header; and the row will be highlighted.

Freezing a Pane

When many rows and columns are accommodated in a single worksheet, the row or column headers do not remain visible; which makes handling the data difficult. To make headers of rows and columns always visible, do the following:

- Select a cell below the row containing headers. In case of column, select the right of the column that must always be visible;
- Select **Window → Freeze**.

To unfreeze cells, deselect the Freeze option by selecting **Window → Freeze** again.

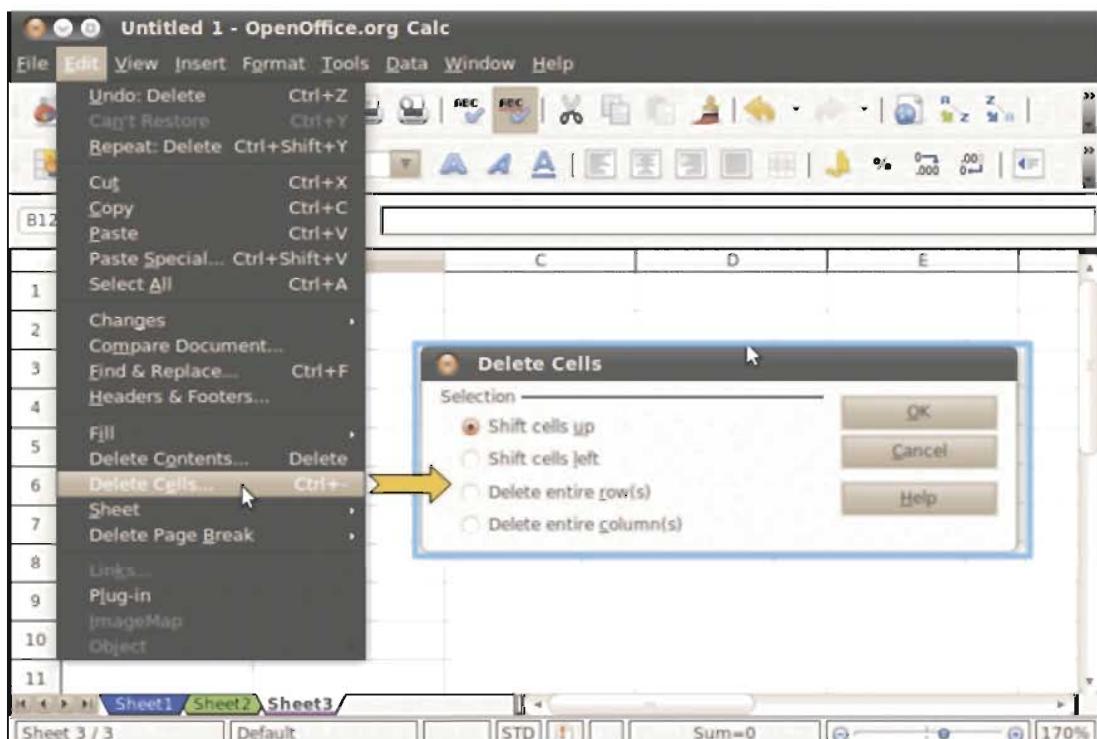


Figure 6.10 : Deleting cells

Cell Level Operations

This section gives you an idea that how content can be written in a cell. The content may be text, number, functions, formulas and reference to other cell. You will also learn to copy and move content of a cell/multiple cells.

Selecting Cells

As stated above, text, numbers, formulas and references to other cells can be entered in cells. But before we enter anything into the cell, we need to select the cell, otherwise the content may not be written in desired cell. To select the cell, you need to just left click on it. You may enter your content in the cell.

Selecting a Range of Cells

To select a cell range, go to the first cell using mouse and left click on it. After selecting the first cell with left mouse click, without releasing the mouse button, drag the mouse pointer over the cells to be included, until the last cell of the desired range is reached. Once the cell in this range is highlighted properly, then release the mouse button.

This is applicable to adjacent cells only. If cells are not physically adjacent to each other, you need to hold down the **CTRL** key. To select all the cells in a worksheet, you need to just click the button in the beginning of the row and column headings. The selection range can also be specified by giving row and column reference such as A1:B12 in the address box.

To cancel the selection of cell or a range of cells, all you have to do is a left click on any cell using mouse.

To Delete the Content of a Cell

To delete the content of a cell or cell range, perform following steps.

- Select a cell or a range of cells;
- Press the **Delete** key on the keyboard, a dialog box will appear;
- Give your choice by clicking. If you want to delete text, then click on **Text**. You may select more than one choice.
- Confirm your choice by pressing **OK** button in the Delete Contents dialog box as shown in figure 6.11.

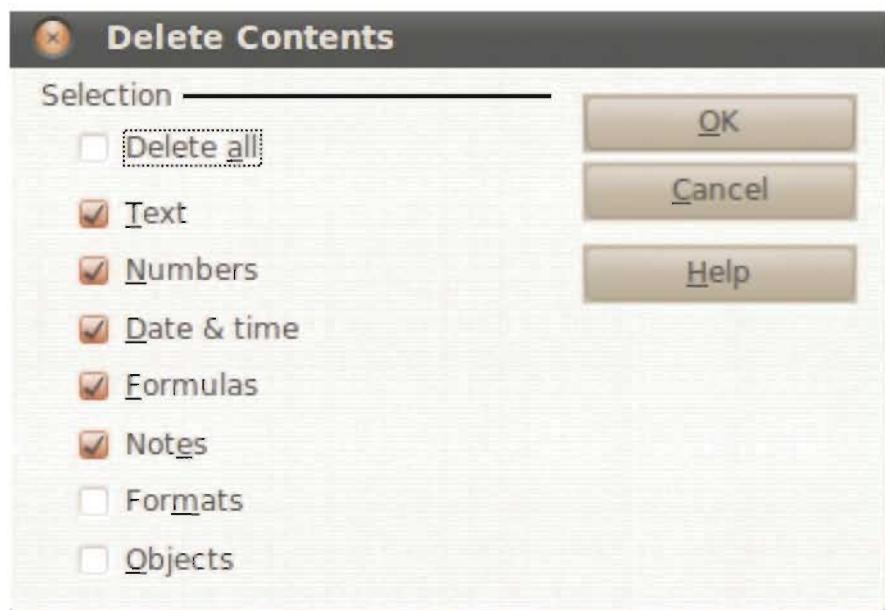


Figure 6.11 : Delete options for cell

The dialog box presents options about deleting rows and columns. Table 6.2 lists the options available in the **Delete Contents** dialog box.

Choice	Description
Delete all	Deletes the entire content of a cell/cell range.
Text	Deletes text from the cell/cell range selected.
Numbers	Deletes numbers from the cell/cell range selected.
Date & time	Deletes only date and time from the cell/cell range selected.
Formulas	Deletes formulas and outcomes from the cell/cell range selected.
Notes	Deletes notes for a cell if added.
Formats	Deletes cell formatting but keeps data.
Objects	Deletes additional elements of a cell such as images.

Table 6.2 : Options available in the Delete Contents dialog box

Editing the Cell Content

To edit the content of a cell, just select the cell, and double click with mouse and edit content. When you finish, press enter key.

Copying and Moving Cells

Cells in spreadsheets can contain many things such as numbers, text, formulas, format, and references to other cell. The content of a cell can therefore be complex, consisting of many values. To copy a cell, a worksheet or a range of cells, do the following.

- Select the cell/part of cell/ range of cells/worksheet that you want to copy;
- Select **Edit → Copy**. You may select the cells and right click on it; from which you can select **Copy** option. The copied data will be highlighted with dotted rectangle;
- Select the paste location and perform the **Paste** command.

For additional paste options, you select **Edit → Paste Special** as shown in figure 6.12.

The Paste Special dialog box provides various parameters for pasting contents. These choices are listed in table 6.3.

Label	Description
1	Pastes only text.
2	Pastes only numbers.
3	Pastes date and time.
4	Pastes only formulas.
5	Pastes notes only.
6	Pastes formatting of a cell or value.
7	Pastes objects only.

Table 6.3 : Paste Special choices

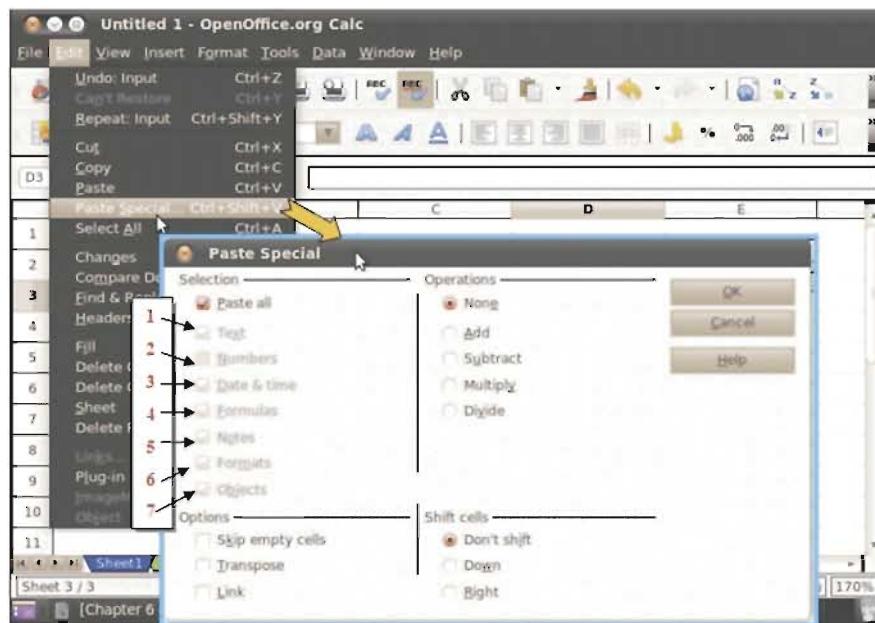


Figure 6.12 : Paste Special options

With the paste special operation you can control the way how content is pasted on destination cells.

Autofill Tool

To automatically fill data into the cells, the autofill tool is used. Perform following steps.

- Enter a number in the first cell;
- Enter the next number in the next cell (row or column);
- Select both the cells;
- Left click on the handle in the cell;
- Without releasing the mouse button, drag across the desired cell range;
- Release the mouse button.

You may see screen as shown in figure 6.13.

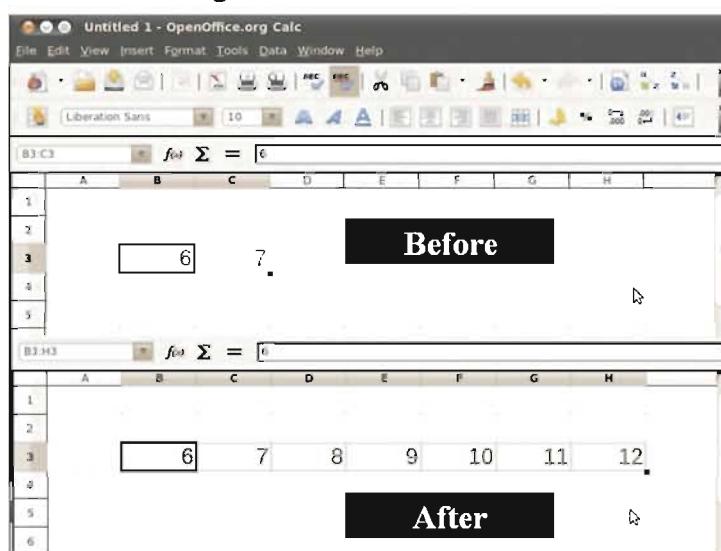


Figure 6.13 : Autofilled numbers

Auto fill of numbers as mentioned above is also known as an arithmetic progression. Try the arithmetic progression in a different way as follows.

Write an odd number (say 1) in a cell. In its adjacent cell, write successive odd number (here, the next odd number is 3). Select both the cells together and try autofill tool to fill next ten cells. Observe the numbers filled automatically. Are they all odd numbers?

Try different series of numbers such as:

- 10, 20, 30,
- 25, 50, 75,...
- 2, 4, 5,....

You can create number tables ($1*1=1$, $2*1=2$, ...) using this tool.

If you have only single number in a cell and you try autofill tool with it, you may have list of numbers increased by 1.

Just think of this. Instead of a number, what if an address or reference is there in a cell ? Write some cell address in a cell and try to autofill some adjacent cells.

Relative and Absolute Address

Earlier we have copied a formula from one cell to many other cells. If a formula written in a cell is copied to other cell, the Calc automatically changes the addresses of cells included in a formula (as we have seen in the example of shopping bill in Chapter 5).

A cell address in the cell A1 is =C1. That is, the cell A1 refers content of the cell C1; third element from the cell (A1) itself. If you copy the content of the cell A1 to B1; the B1 should refer to content of the third cell from the cell B1; that is D1. Such address mechanism is called relative. When a relative address is copied from one cell to another, it will be automatically changed. To avoid this we need to write address such as =\$A\$1 (with the dollar sign (\$) added before the row and/or column) to make it as fix address. This is called an absolute address. The absolute address does not change when the formula is copied or moved. Absolute address always indicates the value of a specific cell. For example, the address =\$A\$4 will always be replaced with the value of the cell A4 in an expression; wherever it is copied or moved.

You may fix a column and vary rows by putting \$ sign against column and vice-versa. For example, if you write address as =A\$4, and try to copy it to different location, only the column will be changed, not the row. It will always remain row 4.

To autofill cells with a formula, do the following.

- Select a cell that contains a formula;
- Left click on the handle in the cell;
- Without releasing the mouse button, drag across the desired cell range;
- Release the mouse button.

Formatting a Cell

We may change property of a cell. We may apply colours to a cell, add image within a cell, add border to a cell, and change fonts of content of the cell. We can also apply different formats to content of a cell; such as changing format of dates and numbers. That is, formatting can be applied to numbers, content, alignment, cell contents, cell border and cell background. For this purpose, we may use a dialog box Format Cells or formatting toolbar. Figure 6.14 displays formatting toolbar.



Figure 6.14 : Formatting toolbar with labels

The description of labels used in figure 6.14 is given in table 6.4.

Label	Description
1	Apply style.
2	Font.
3	Font size.
4	Bold, italic, and underline fonts.
5	Alignment as: left, centred, right, and justified.
6	Merge selected cells.
7	Currency.
8	Percent.
9	Add/delete decimal place.
10	Decrease/increase indent.
11	Cell border format.
12	Background colour of a cell.
13	Font colour in a cell.
14	Add/remove unformatted cell border.

Table 6.4 : Formatting toolbar options

Let us see some formatting styles. Later you may experiment the above formatting styles on some cells of a worksheet.

Number Format

The number format affects the appearance of numbers in cell. The number format can be applied to a single cell, multiple selected cells and range of cells. Choose Format cells, open tab Numbers and set the choices through the dialog box appear which is illustrated in figure 6.15. The description of labels used in figure 6.15 is given in table 6.5.

Label	Description
1	Selection of appropriate format such as numbers, dates etc.
2	Typical appearance of the format selected.
3	Regional settings.
4	Preview of the selected cell; inserting a thousand separator.
5	Number of decimal places.
6	Number of leading zeros.
7	To make negative numbers red.
8	Thousand separators.

Table 6.5 : Number formatting options for cells

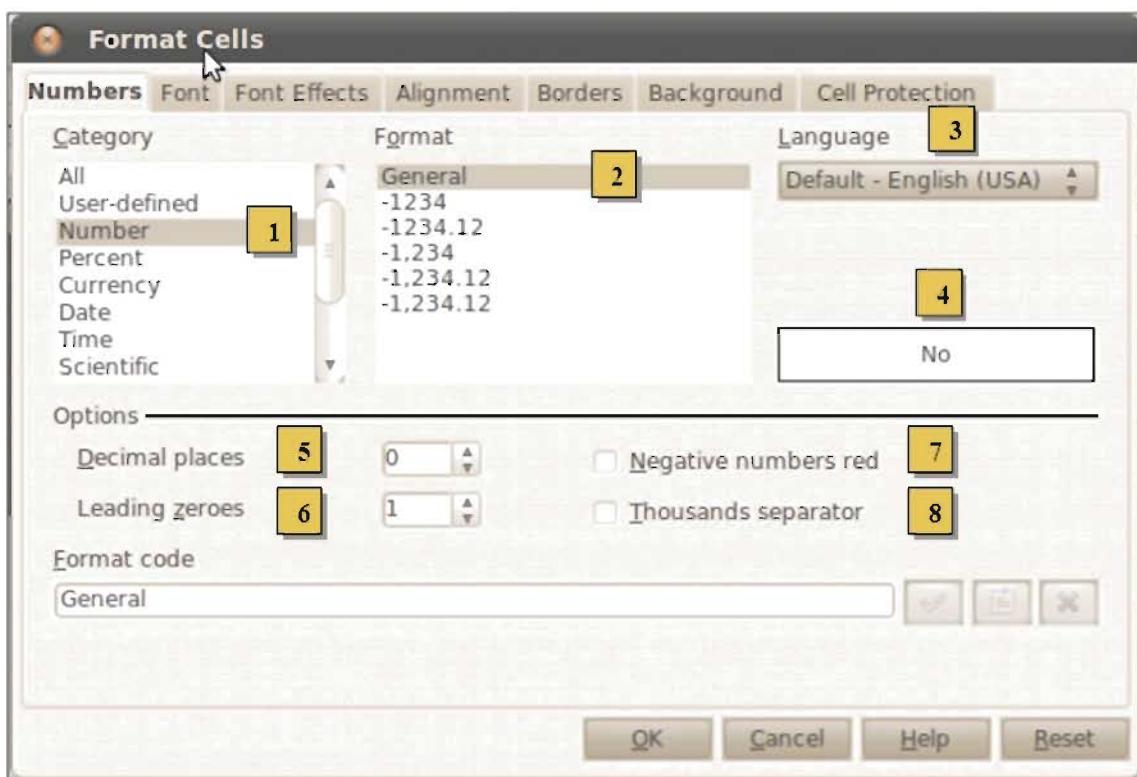


Figure 6.15 : Number formatting options for cells

Figure 6.15 shows typical number formatting options for cell containing numbers. To change date format, currency format or percentage format, appropriate choice is to be selected.

Fonts can also be changed through the **Font** tab of the **Format Cells** dialog box; effects can be changed in the **Font Effects** tab as illustrated in figure 6.16.

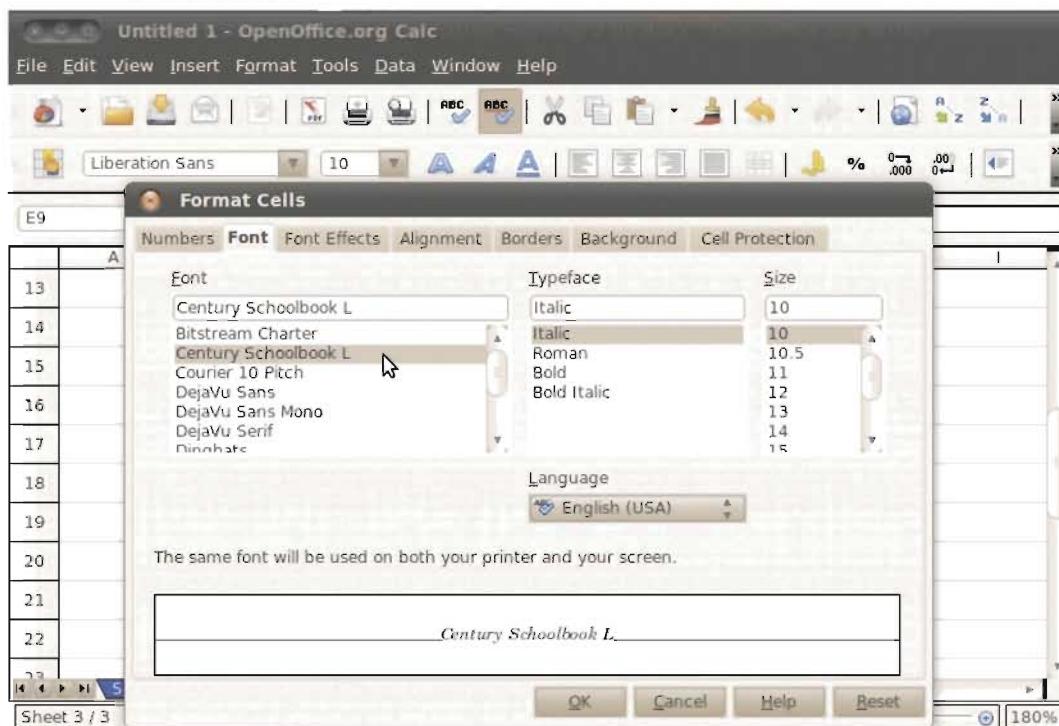


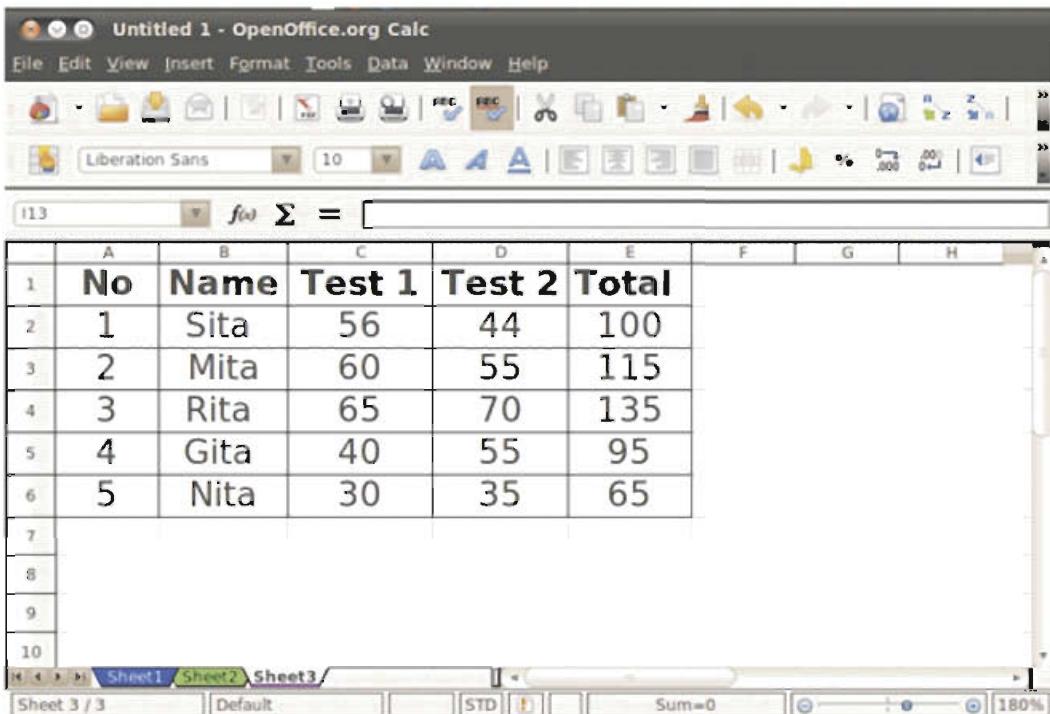
Figure 6.16 : Changing fonts

You might have applied some of these font effects; such as bold, italic, underline, and font size. Some of these effects are presented as icons on formatting toolbar. Just selecting the proper icons, these effects can be directly applied.

Data Sorting and Filtering

It is easy to understand and use data when it is arranged in some order. Arrangement of data in particular order is known as sorting. Operations to sort data in ascending or descending order are available in Calc. If there is a single column or row of data, it is sorted in desired fashion. However, if data is spread in more than one row and column, then all the columns or rows are shifted and sorting is done.

Consider a set of marks of students as an illustration. To find out who has secured maximum total marks and to prepare merit list, we need to do sorting in descending (largest value comes first) order. To experiment this, open a new spreadsheet and enter the data as shown in figure 6.17. Save the file with appropriate name.



The screenshot shows a spreadsheet titled "Untitled 1 - OpenOffice.org Calc". The menu bar includes File, Edit, View, Insert, Format, Tools, Data, Window, and Help. The toolbar below has various icons for file operations, cell selection, and data manipulation. The font dropdown shows "Liberation Sans" and the font size dropdown shows "10". The formula bar shows "f13" and "Σ = []". The main area contains a table with the following data:

	A	B	C	D	E	F	G	H
1	No	Name	Test 1	Test 2	Total			
2	1	Sita	56	44	100			
3	2	Mita	60	55	115			
4	3	Rita	65	70	135			
5	4	Gita	40	55	95			
6	5	Nita	30	35	65			
7								
8								
9								
10								

Figure 6.17 : Data entered in worksheet

To sort the data as shown in figure 6.17 based on values of cell E i.e. **Total**, we need to do following.

- Click on a cell in the column by which data are required to sort;
- Click the **Sort Descending** button in the **Formatting** toolbar as shown in figure 6.18.



Figure 6.18 : Sorting using formatting toolbar

There is another way to do the same operation. All you need is to select **Data → Sort**. A dialog box will appear. In the sort criteria of the dialog box, select **Descending**. Figure 6.19 illustrates a typical view of sort dialog box.

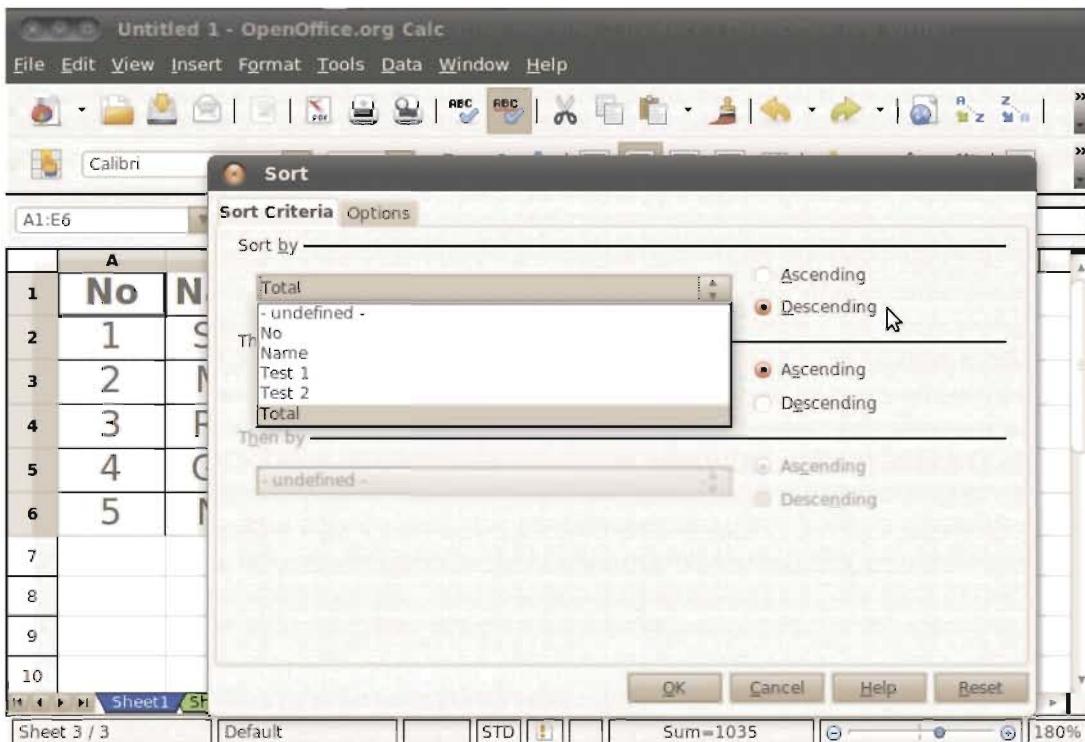


Figure 6.19 : Sort dialog box

After sorting, you will find that the total marks **135** will come at first position in the column. Obviously these marks are scored by **Rita**, and her name should come first in the student's name column. If you see the result, along with Rita's total marks all other data related to her have been moved to the first row. That means, if you sort by a column, many rows may be moved as illustrated in figure 6.20. Note the changed order of the records (data) in the figure 6.20.

I13						
	A	B	C	D	E	F
1	No	Name	Test 1	Test 2	Total	G
2	3	Rita	65	70	135	
3	2	Mita	60	55	115	
4	1	Sita	56	44	100	
5	4	Gita	40	55	95	
6	5	Nita	30	35	65	
7						
8						
9						
10						

Figure 6.20 : Sorted data in ascending order

Not only on values but you can sort items according to alphabets too. Take an example of an English language dictionary, in which words are sorted in alphabetical order. Look at the student's data we are experimenting right now, and try to sort the data set on the students' name. Do the following to sort the above data according to students name in ascending order.

- Select **Data → Sort;**
- Give column name as **Name;**
- Check the sort order, by default it is **Ascending.**

You may sort data on two or more fields. Consider a scenario, you have list of students with information such as their full names, their locations and their marks. You may want to sort the data first by location and then by name. Consider the example given below :

1. Enter some sample data provided in table 6.6 in a Calc worksheet.

Name	Location	Marks
Nita	Vallabh vidyanagar	65
Gita	Anand	95
Sita	Baroda	100
Mita	Ahmedabad	115
Rita	Gandhinagar	135
Kavita	Vallabh vidyanagar	110
Punita	Ahmedabad	105
Sangita	Anand	85
Sunita	Ahmedabad	70
Babita	Anand	90

Table: 6.6 : Sample data for sorting

2. Select the data including headings.
3. Choose **Data → Sort.**
4. Select "**Location**" in the **Sort by** field.
5. Select "**Marks**" in "**Then by**" filed.
6. Click **OK** button.

These operations are shown in figure 6.21.

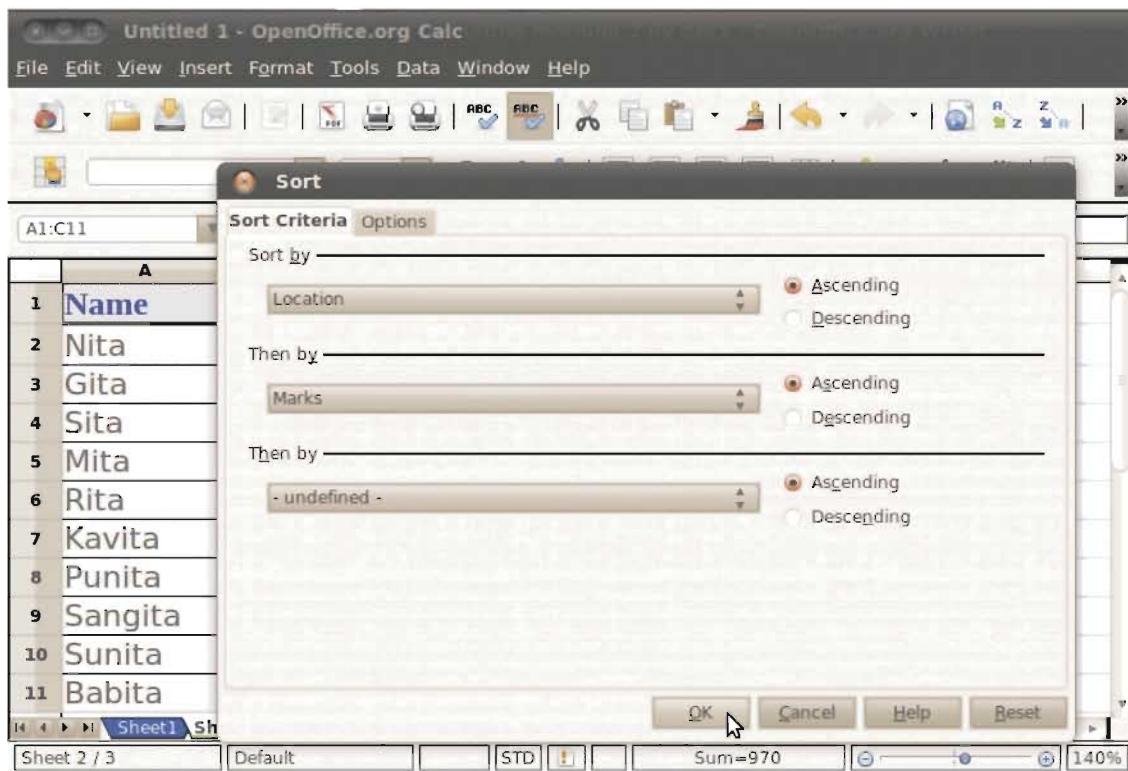


Figure 6.21 : Sorting options

Performing these steps will sort the selected data first by location and then by marks. That means, all Ahmedabad students are listed together in ascending order (lowest first) of their marks. As location is sorted in ascending order, the Ahmedabad location will come first and Vallabh vidyanagar will come at end. The sorted data will look like as shown in figure 6.22. You may add third level of sorting, if required.

	A	B	C	D	E
1	Name	Location	Marks		
2	Sunita	Ahmedabad	70		
3	Punita	Ahmedabad	105		
4	Mita	Ahmedabad	115		
5	Sangita	Anand	85		
6	Babita	Anand	90		
7	Gita	Anand	95		
8	Sita	Baroda	100		
9	Rita	Gandhinagar	135		
10	Nita	Vallabh vidyanagar	65		
11	Kavita	Vallabh vidyanagar	110		

Figure 6.22 : Sorted data

Data Filtering

Data filtering help to filter out unnecessary data and presents only those which you want to see. Consider the data about the students name and location, which we have just sorted in previous section. Let us add a filter on the data. Suppose we want to see details of students who scored 100 or more than 100 marks. To do so, we may add automatic filters as follows:

1. Select **Data → AutoFilter** as shown in figure 6.23. You may use the exiting worksheet, in which you have already entered the data;

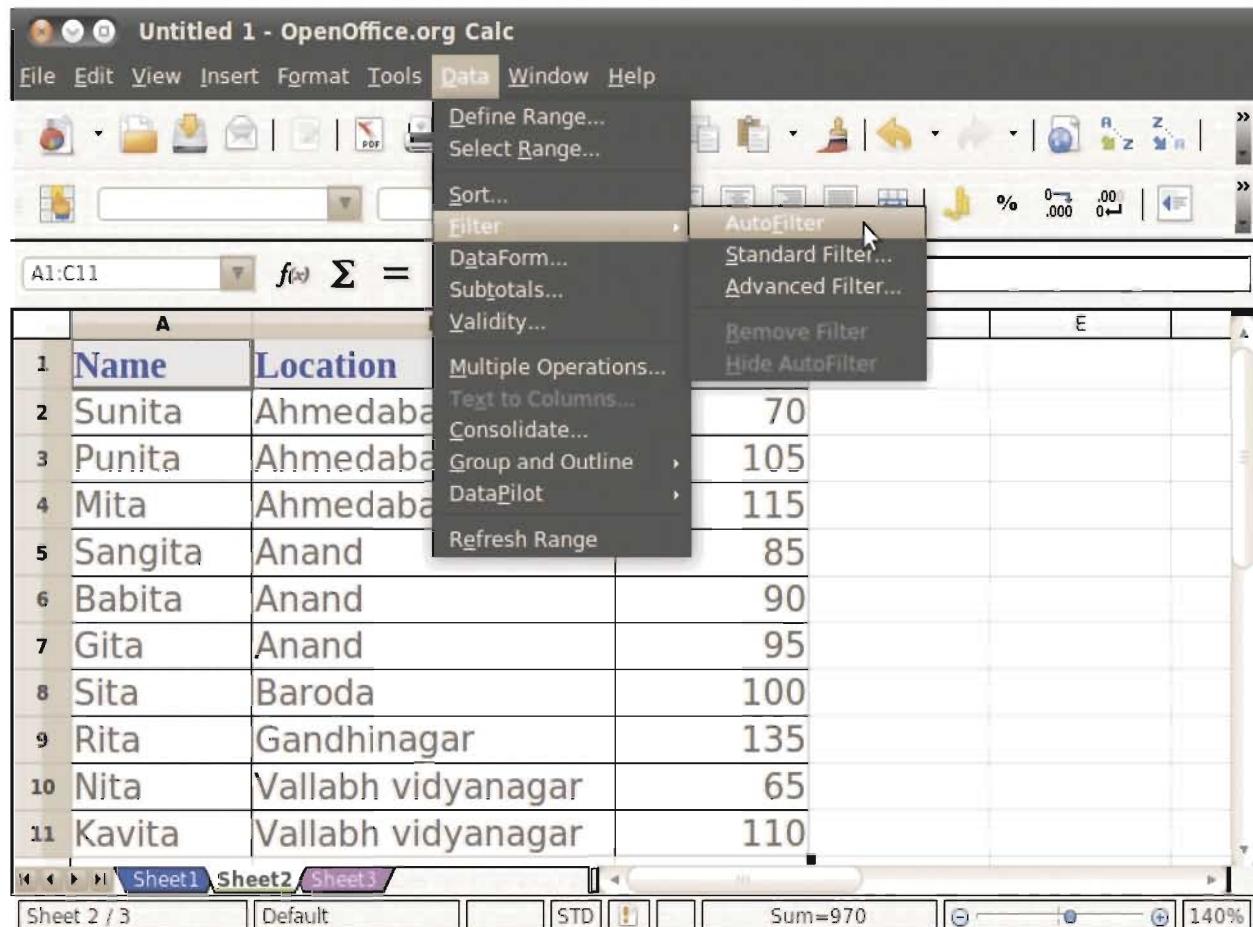


Figure 6.23 : Applying filters to the selected data

2. You will see dropdown arrows at the top of each column;
3. Click and hold down the arrow on particular column (here Marks) and select a value. You may choose from values you have already entered or phrases such as top 10 etc. If you require specific option, you may select standard filter and choose values from it as shown in figure 6.24.
4. You may see that the spreadsheet has filtered out student's data with marks less than 100. You may have noticed the changed colour of the arrow indicating that some data have been filtered out as shown in figure 6.25.

Choose **Data → Filter → AutoFilter** again to turn off the filter.

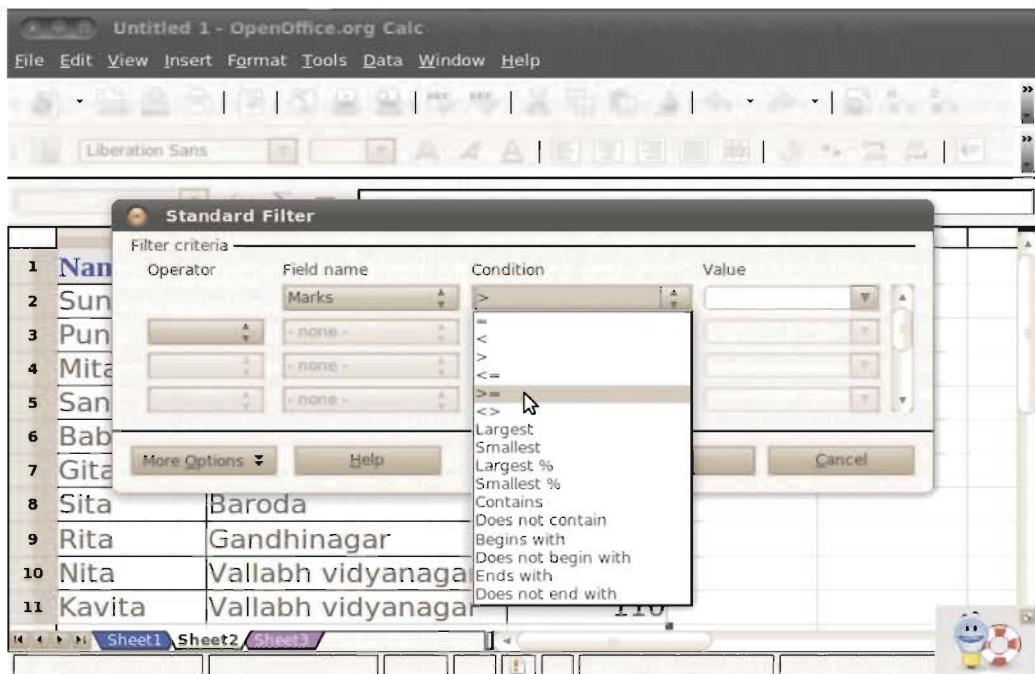


Figure 6.24 : Standard filter

Untitled 1 - OpenOffice.org Calc				
File Edit View Insert Format Tools Data Window Help				
A1:C11	f _{oo}	Σ	=	Marks
1	Name	Location	Marks	
3	Punita	Ahmedabad	105	
4	Mita	Ahmedabad	115	
8	Sita	Baroda	100	
9	Rita	Gandhinagar	135	
11	Kavita	Vallabh vidyanagar	110	
12				
13				
14				
15				
16				
17				

Figure 6.25 : Filtered data

Data Validations

As stated earlier, the spreadsheet packages are used for data analysis such as balance sheet preparation and financial analysis, merit list preparation and analysis of students' results. In this case the formula and functions you have used will output correct result provided valid data are given. However, one cannot make sure that the users of the system always enter valid data. To avoid errors in entering data we may use data validations. To validate cell contents select the cells and select **Data → Validity**. This will open a Validity dialog box as shown in figure 6.26.

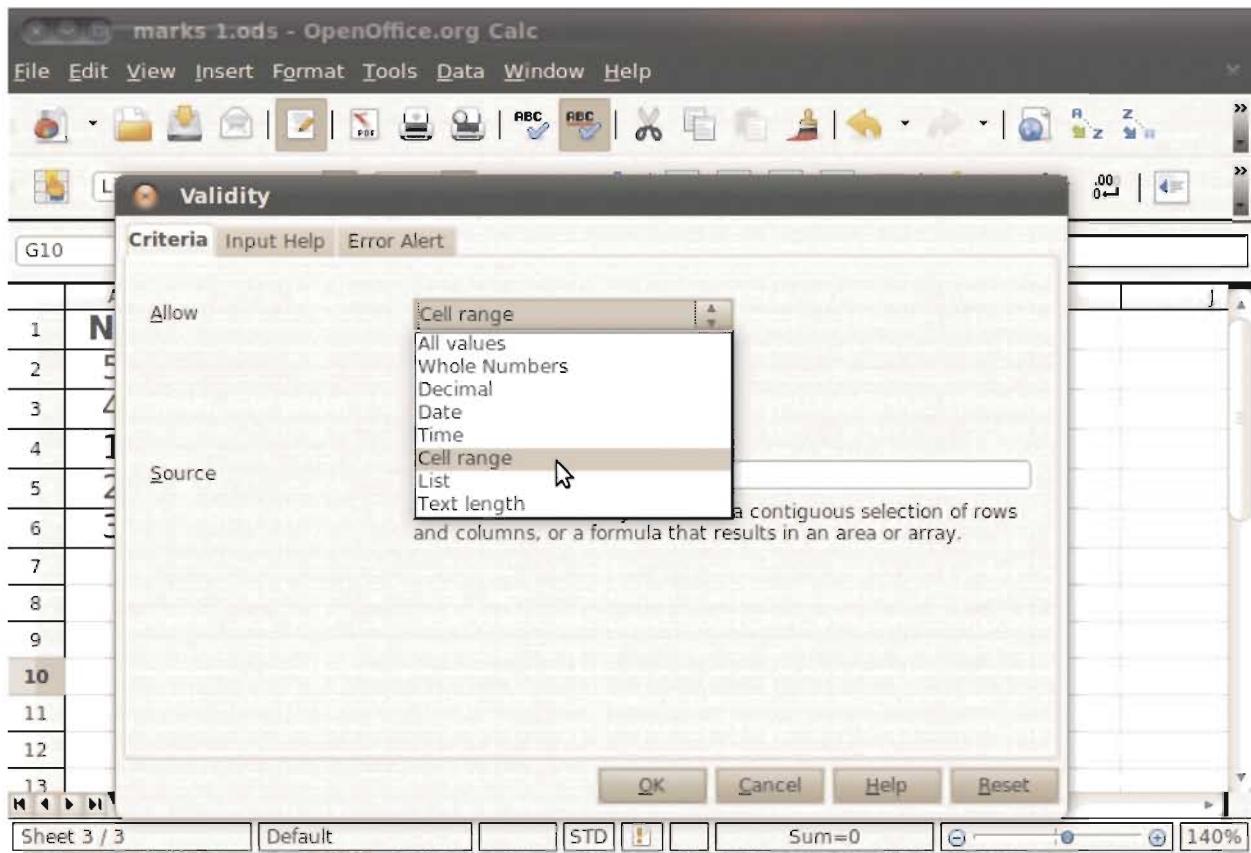


Figure 6.26 : Data validation options

You may now set different criteria for data which you expect from users. To control invalid data entry we may set some conditions here. These conditions include cell range, all values, only whole numbers to be entered, only date values to be allowed etc. We may do such setting with **Criteria** tab shown in the figure 6.26. The second tab is about **Input Help**, where we can provide help to users at the time of data entry.

There is one more tab, **Error Alert** to provide alert on error, if any. It is to be noted that if you select **Format** or **Delete All** from the **Delete Contents** window, then the validity options you have set will be removed.

Other Operations in Calc

Calc provide many other useful operations checking of spelling, finding and replacing text, printing the worksheet and many more. Some of these options are discussed in this section.

Spelling Check

Users while typing the contents in Calc worksheet generally make errors in spelling. Calc gives us a facility to check spellings. The words spelled incorrectly will be underlined with red line, provided the spelling check toggle button on the toolbar is turned on. As you type something wrong (with incorrect spelling), the word is highlighted with red underline. All you need to do is just a right click and select appropriate option from given choices. To check spelling the Calc uses a dictionary. If the word you have entered is found in the dictionary with the same spelling, the word is considered as correct word. Figure 6.27 illustrates possible option for spelling checking and correction.

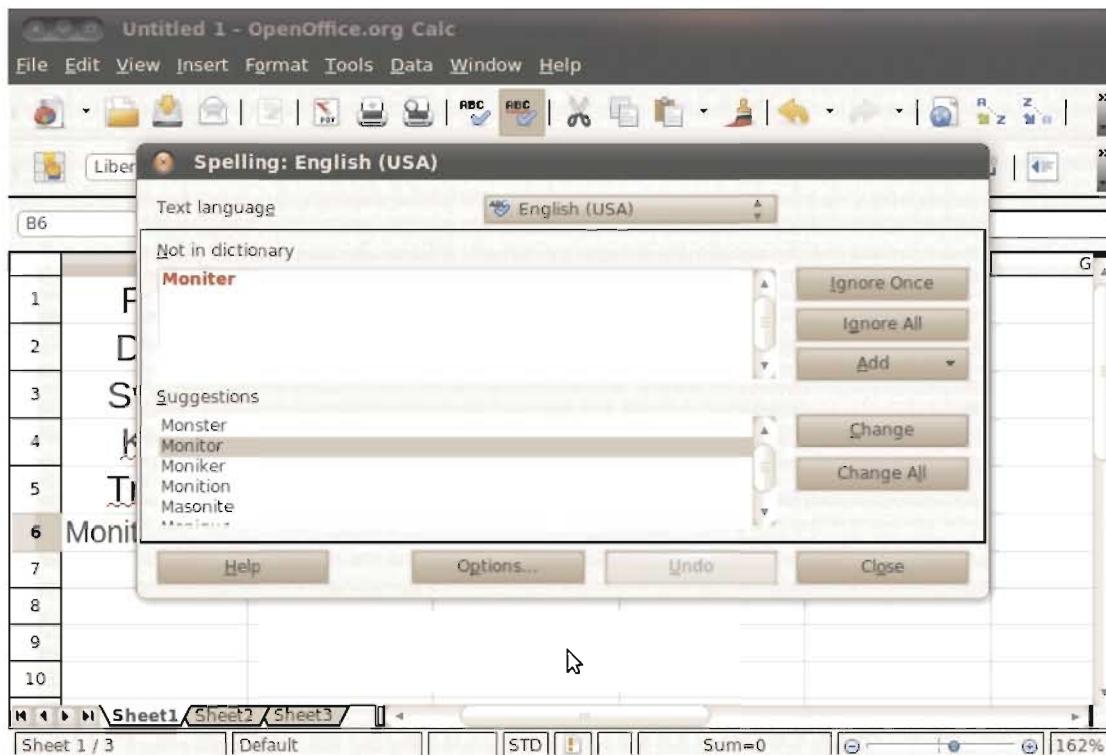


Figure 6.27 : Spelling check options

It may so happen that the word is correct, such as your name, but it may not be there in the standard dictionary. In that case, your name may have red underline too! To avoid this, you may add your name to the dictionary using option as "Add to dictionary". Now your name will not be highlighted as incorrect anymore. One simplest way to go for spelling check is just to press a function key F7. To avoid spelling check automatically, you may choose **Format → Cells → Font language=[None]**

Find and Replace

The commands Find and Replace are used to quickly find data in a worksheet and replace text or numbers in cells fully or partially. To do so, perform following steps:

- Select **Edit → Find & Replace;**
- Perform necessary actions as asked in the Find & Replace dialog box.

Printing

Printing the worksheet requires special care because of the size of the worksheet. Unlike the word processor, the size of a document of a spreadsheet application is not fixed. Multiple rows and columns you have created might not fit in a page. Moreover, some information will be truncated. Further, it is not required to print the whole worksheet. Therefore, before printing a document, it is advisable to use the print preview option. The print preview option provides a view of spreadsheet along with page breaks and margin. To preview a worksheet following steps can be carried out.

- Select **File → Page Preview;**
- At top of the computer screen, a bar similar to the one shown in figure 6.28 will appear. It shows possible actions that user can take to preview the document.



Figure 6.28 : Preview bar

By choosing necessary icons, we can carry out necessary actions. Labels shown along with the figure 6.28 describes the actions, which are listed in form of table 6.7.

Label	Description
1.	Turning of pages
2.	Jump to last page
3.	Zoom tool
4.	Full screen preview
5.	Leads to a dialog box for page formatting
6.	Makes margins visible
7.	Scaling
8.	Close preview

Table 6.7 : Preview options

Page preview can also be used to set page formatting values. To carry out page formatting, perform following steps :

- Select **File → Page Preview;**
- Click the **Format Page** button on the horizontal dialog box appear; it will open a rectangle dialog box called Page Style.
- Select the second tab entitled "**Page**" in the **Page Style** dialog box as shown in figure 6.29;
- Select required choices and save changes by clicking on **OK** button.

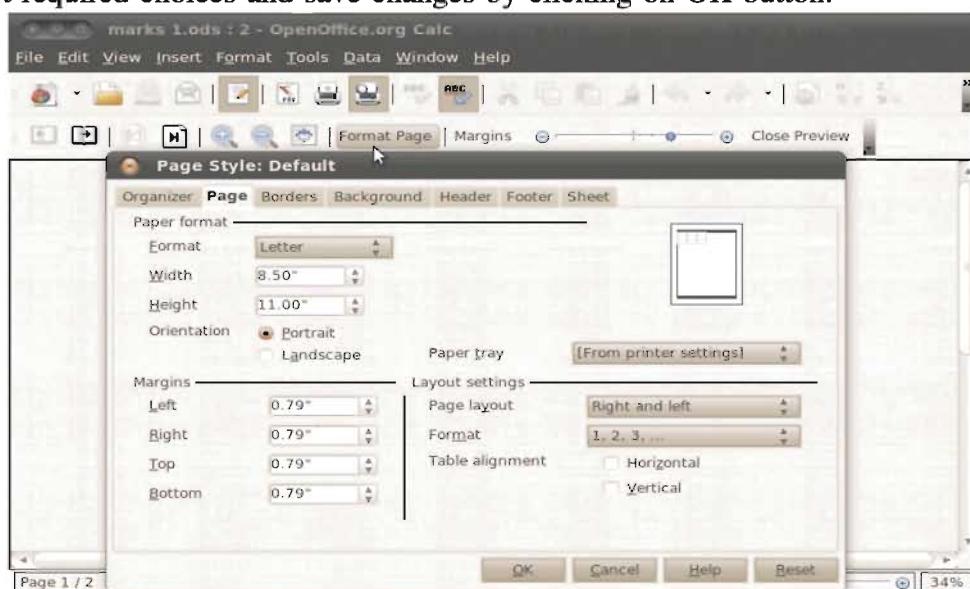


Figure 6.29 : Page formating options

You may choose paper size (**Format**) such as A4, legal or any custom size page. You can also set left, right, top and bottom margins using **Margins**. Beside these options, the other page formatting option you may experiment are **Paper Tray** option (specifying from where your papers comes to the printer; from upper tray or lower tray), and **Page Layout** if you are printing more than one pages on a physical page (single paper). The format choice is also used to specify the paper **Orientation** such as portrait or landscape orientations.

What if you want to print the cell grids (borders surrounding the cells) as well as the column and row headers? Just do the following things :

- Select **Format → Page**; a dialog box will appear.
- Alternatively, select **File → Page Preview** and click the **Format Page** button;
- Open the **Sheet** tab;
- Tick the **Grid** checkbox as illustrated in figure 6.30;
- Tick the **Column and row headers** checkbox;
- Close the dialog box.

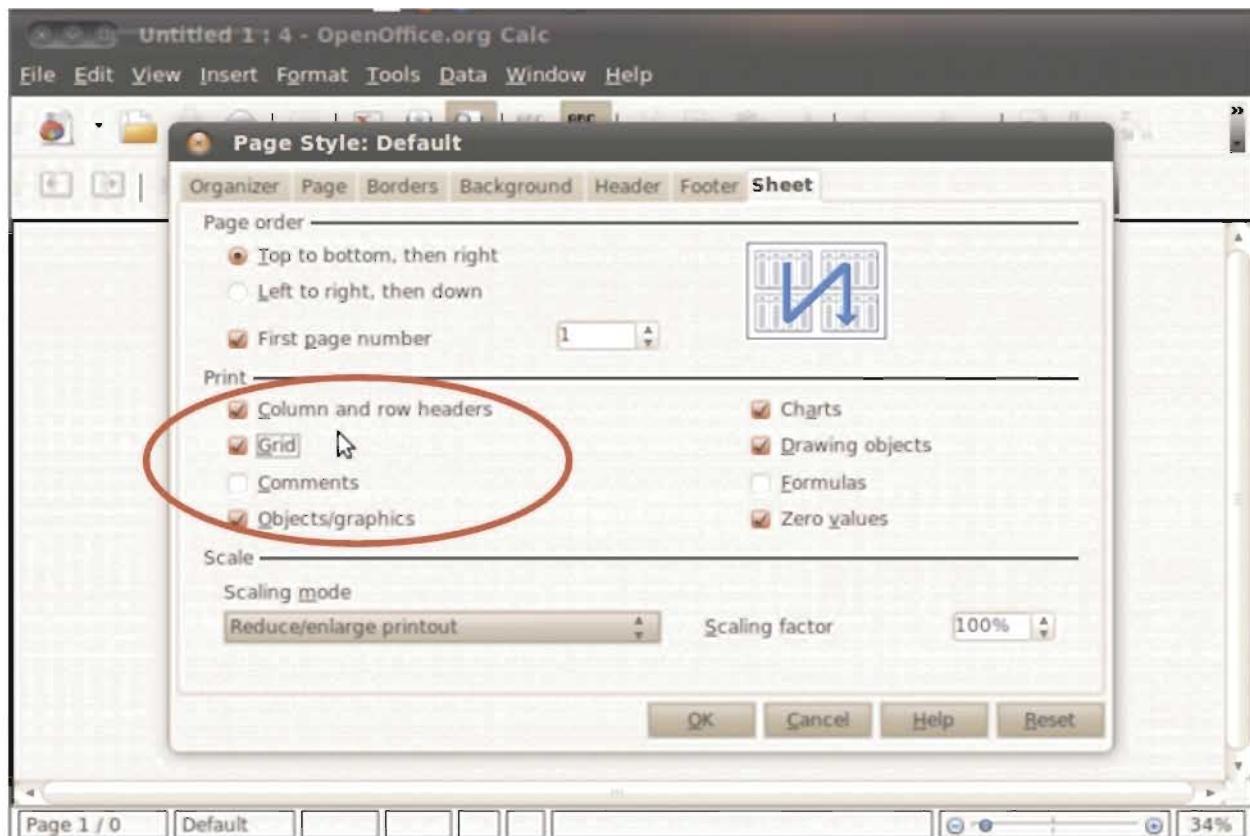


Figure 6.30 : Setting column and row headers as well as cell grids

You may also print colourful borders pressing the tab **Borders**. You may select border type, border colour and border thickness along with other options to set borders. Figure 6.31 illustrates options for setting borders to the cells. Once you set parameters click on **OK** button. Then you may preview the page. Figure 6.31 also shows a preview of data (blue rectangle indicated by block arrow) with red border as well as yellow background for the data entered. However, to see preview you have to go for Page Preview after setting the borders, cell headers and background.

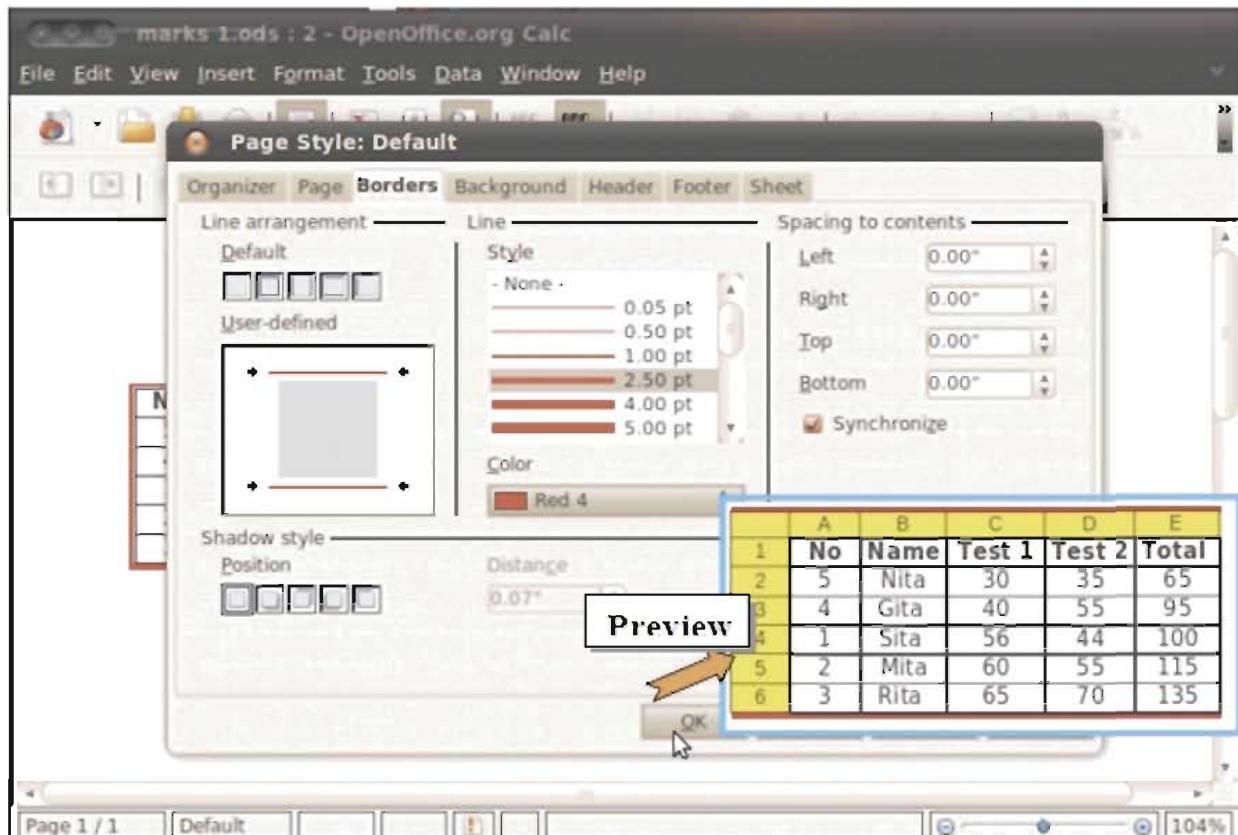


Figure 6.31 : Setting Borders and background

When every option and format is set, you may print the document just by clicking the printer icon at the top line.

Modifying Page Breaks

To see and modify page breaks, do the following:

- Select **View → Page Break Preview**;
- If you are happy with the layout, you may print it else change the page breaks by dragging the lines as per your requirement.

By performing the menu command **View → Normal** you will get the normal view of the page.

Header and Footer

At top of every page and at bottom of every page an area is kept reserved. These areas at top and bottoms are known as header and footer respectively.

To create the header do following.

- Select **File → Page Preview → Format**;
- Open the Header tab and click Edit and apply the changes needed.

The header and footer areas in Calc are divided in three areas each, called left area, centre area and right area. Whatever you write in these areas, will be automatically inserted into the document. Inserting header is illustrated in figure 6.32.

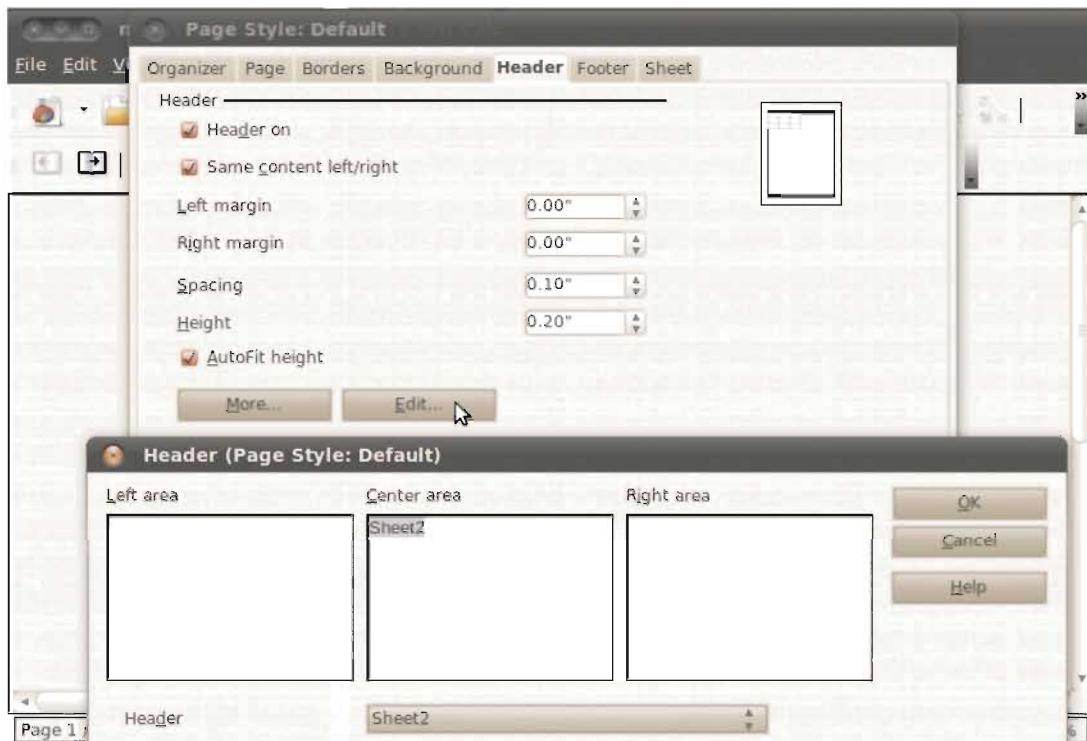


Figure 6.32 : Inserting Header

Getting Help

Calc has a built-in manual to provide help on different functions. To search required function, Calc provides a search function. All you need to do is just select **Help → OpenOffice.org Help**. Figure 6.33 shows a typical screen that appears when you call for help. The simplest alternative way is to press the **F1** function key on the keyboard.



Figure 6.33 : Getting Help

Summary

In this chapter we have seen data formatting, sorting and validating operations in Calc spreadsheet. We learnt spreadsheet level operations which facilitate opening a new or existing document, saving document in Calc as well as other format, other operations on file such as closing and searching documents. We also learnt sheet level operations such as re-naming and re-colouring sheet, inserting and deleting sheet, etc. in various ways. The cell level as well as row and column level operations also we have seen in this chapter. We learnt data sorting, data validation, and other miscellaneous utilities such as autofill operations, setting header and footer, printing documents, spelling check and getting help.

EXERCISE

1. Explain how to create a new document in Calc.
2. List how to rename and recolor the Calc worksheet.
3. Explain insertion and deletion of extra worksheets in Calc.
4. Write a short note on deleting cell content.
5. Explain working of autofill tool in Calc.
6. Explain relative and absolute address in Calc.
7. Write a short note on data filtering.
8. Write a short note on data sorting.
9. What is data validation? Why is it necessary ? Also explain how Calc does it ?
10. Explain how to add header and footer in a Calc worksheet while printing.
11. Choose the correct option from the following :
 - (1) Which of the following technique arranges the data set in a particular order such as ascending or descending ?

(a) Data formatting	(b) Data validations
(c) Data filtering	(d) Data sorting
 - (2) Which of the following techniques can be used to allow only date values in a cell ?

(a) Data formatting	(b) Data validations
(c) Data filtering	(d) Data sorting
 - (3) Which of the following options when selected deletes all data validations ?

(a) Delete formatting	(b) Delete all
(c) Delete formula	(d) Delete me

LABORATORY EXERCISE

1. Save your Calc document in other format using Save as option. Open it from other software in which you have saved it.
 2. Open a Calc document, add 7 sheets using Sheet tab, colour them as rainbow colour such as "Violet", "Indigo", ...etc. Make sure that "Indigo" sheet is re-coloured with indigo colour and so on.
 3. Try autofill tool with negative numbers.
 4. Try autofill tools with two dates. Write 15-08-2013 in a cell and in its adjacent cells write a successive date. Drag the content to next ten cells. Using this technique try to create a monthly calendar.
 5. Add your name in to the dictionary so that red line will not be displayed under it.
 6. Enter your friends list with their birth dates and sort the list according to City and then by names. Take columns headings as shown below.

Name	Address in one line	City	Mobile no	Email	Date of Birth	Month of Birth	Year of Birth
Khushi	1, Gokul Park	Anand	01	July	1995
....							

7. Filter the list of friends you have created in problem 6 of this exercise, so that it will display only those friends information whose birthdates are in August month.
8. Use help to find information about sorting and filtering. Prepare notes on these topics using the help.
9. Use your notes on data validations and also use the data you have entered in the problem 6 of this exercise, to validate the data in the date of birth column so that nobody can enter date such as 33. You have to make data validation in such a way that it will accept only 1 to 31 numbers as date of birth values.





Functions in Calc

Entering only data and simple mathematical expressions may not be sufficient for real life applications involving complex decision making operation and analysing large amount of data. To carry out such activities, spreadsheet programmes such as Calc provide built-in standard functions for mathematical, logical, statistical, date and time, financial and other calculations on numbers and text.

A function, like an expression generally begins with an equals (=) sign, followed by a function name and one or more function argument specified in its brackets. Sometimes, operators such as "+" may also be used with functions and formulas. However, it is a common practice to use '=' symbol as a prefix to avoid a function being simple text. The arguments can be a value, an address of a cell, text, a constant, or one or more functions. In Chapter 5, we have used the SUM function from the tool bar (shopping bill example). The SUM function, directly applied from the tool bar (indicated by symbol sigma - Σ , is a kind of shortcut. Systematically the same function can be written as **=SUM(A1:A10)** where SUM is a name of function, which specifies that summation procedure is called; and A1 and A10 are references to the Calc cells in the same worksheet. Together A1 to A10 (A1:A10) specify a range of cells containing 10 different cells. The content written in the brackets is known as argument of the function. Instead of the cell address or cell range, we can also write a value as **=SUM(23,25)**.

Similarly, you may also wish to try **=SUM(A1:A10)** function using ',' (comma) or ';' (semi-colon) instead of ':' (colon) in between the cell address A1 and A10. You may observe that just by using ',' or ';' one can sum up only two values at the cell A1 and A10 respectively. To sum all the values between A1 and A10 including content within them; we must use either ':' or individually all cell addresses as **=SUM(A1,A2,A3,A4,A5,A6,A7,A8,A9,A10)**.

Figure 7.1 illustrates the three ways to perform addition operation. You can see the status of selected cell having value (SUM=55) and the formula bar showing full summation expression. You can also see when A1 is referred in the formula, its colour is blue (as shown in the figure 7.1) and the colour of the cell A1 is also blue. This is really useful to cross check the entered formula.

The process of inserting a function is similar for all functions. The arguments specific to the function and the correct spelling of the function are mandatory. That is, to use a function we need to enter a correct spelling of the function along with necessary arguments. This is known as syntax of the functions. If we do not know the exact spelling of a function and the possible attributes, it will result in error.

The screenshot shows a Microsoft Excel spreadsheet with data in columns A and B. Column A contains numbers 1 through 10. Column B contains the sum of the first 10 natural numbers (55) and some descriptive text. The formula bar at the top shows '=SUM(A1:A10)'. Cell B1 contains '55' and 'Using Σ'. Cell B3 contains '=SUM(A1,A2,A3,A4,A5,A6,A7,A8,A9,A10)'.

	A	B	
1	1	55 Using Σ	
2	2	55 SUM(A1:A10)	
3	3	55 =SUM(A1,A2,A3,A4,A5,A6,A7,A8,A9,A10)	
4	4		
5	5		
6	6		
7	7		
8	8		
9	9		
10	10		
11			

Sheet1 Sheet2 Sheet3

Sum=55

Figure 7.1 : Three ways to perform addition operation

To enter a function in a cell, do the following :

- Select a cell in which result is to be displayed;
- Start entering function with '=' sign; if you do not write '=' sign (or any other valid operator such as "+"), the function will be considered as normal text).
- Enter the name of a function;
- Enter function arguments; and
- Press the Enter key;

Function Wizard

It is not possible to place many functions on the toolbar (generally top line of the screen), or to remember them with their correct syntax; hence most of the functions are entered through the function wizard.

Use the function wizard in the following way.

- Select a cell in which result is to be displayed;
- Select **Insert → Function**. You can also press **Ctrl** and **F2** to do the same from an opened spreadsheet document;
- Another alternative is to press the **Function Wizard** key in the formula toolbar as shown in figure 7.2;



Figure 7.2 : Function Wizard

- A dialog box will appear. Choose required function from the dialog box. Figure 7.3 shows typical dialog box.

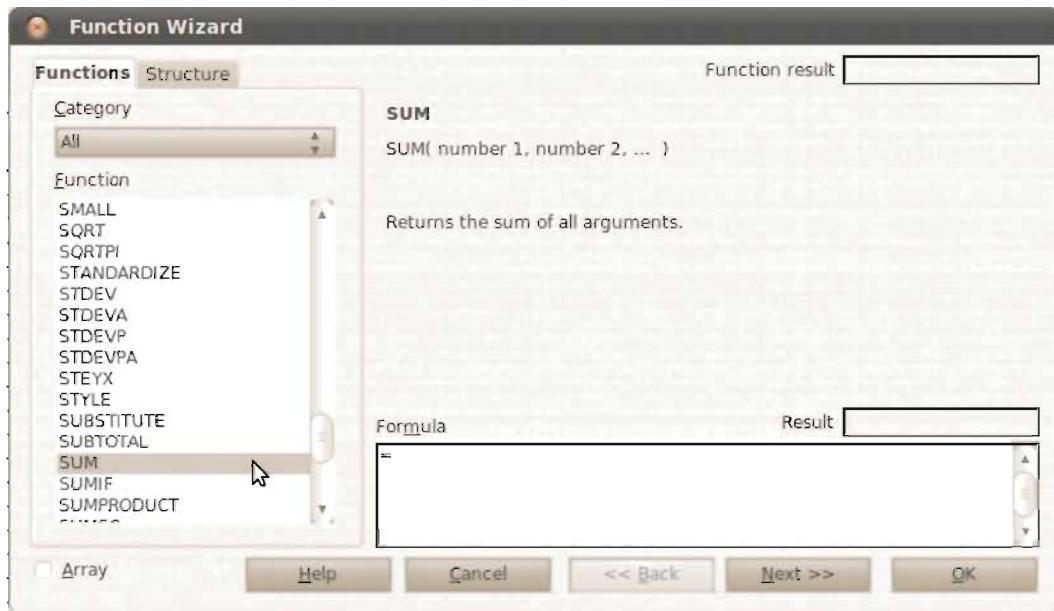


Figure 7.3 : Dialog box for function wizard

- Select the **SUM** function and select **Next** button;
- Enter values or select the function arguments from the worksheet. Here you can enter up to 30 arguments values, addresses (references) or ranges one by one. You may see the list of the arguments indicated by number 1, number 2, ... etc. As shown in figure 7.4;

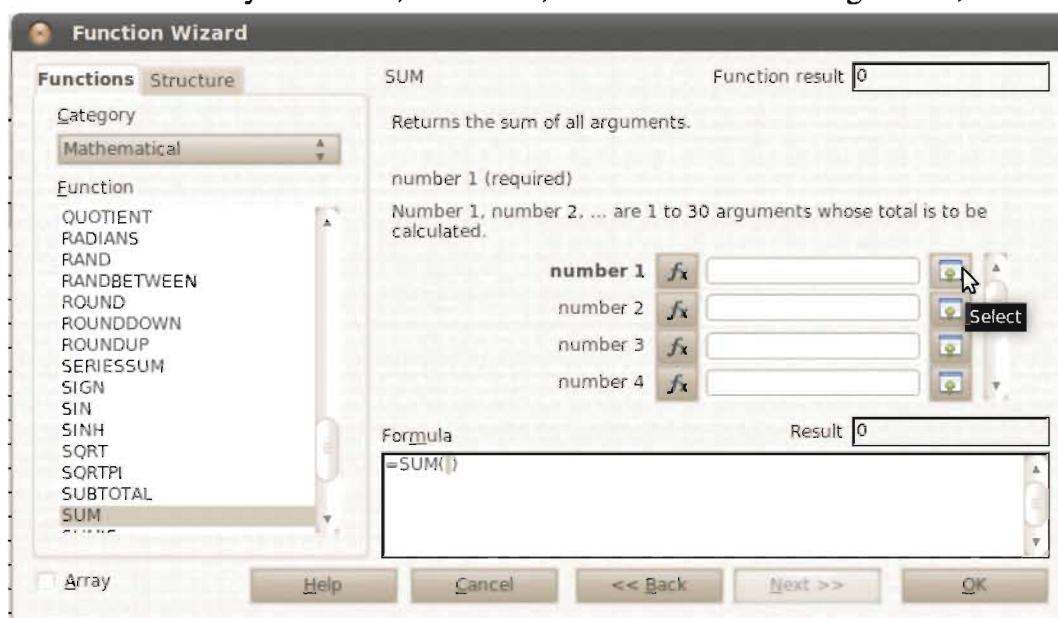


Figure 7.4 : Entering arguments of a function

- If you do not know, what is the value of a number and where it lies within the spreadsheet, you may wish to select it from the worksheet directly;
- To select values use the **Select** button as shown with the mouse arrow in the figure 7.4;
- When you click on the select button, it will lead you to the worksheet area. The whole dialog box for the function wizard is converted into a small toolbar type rectangle. Now you can see the worksheet area as well as the minimized function wizard. You are free to use mouse and select the cell or cell range which you want. Figure 7.5 indicates the minimised function wizard (rectangle outlined with blue line) and the worksheet area. If the data in which you are interested are not available on the same sheet, you may go to other sheet also. Verify data on the function wizard. It is **Sheet1:A1:A10** in our case;

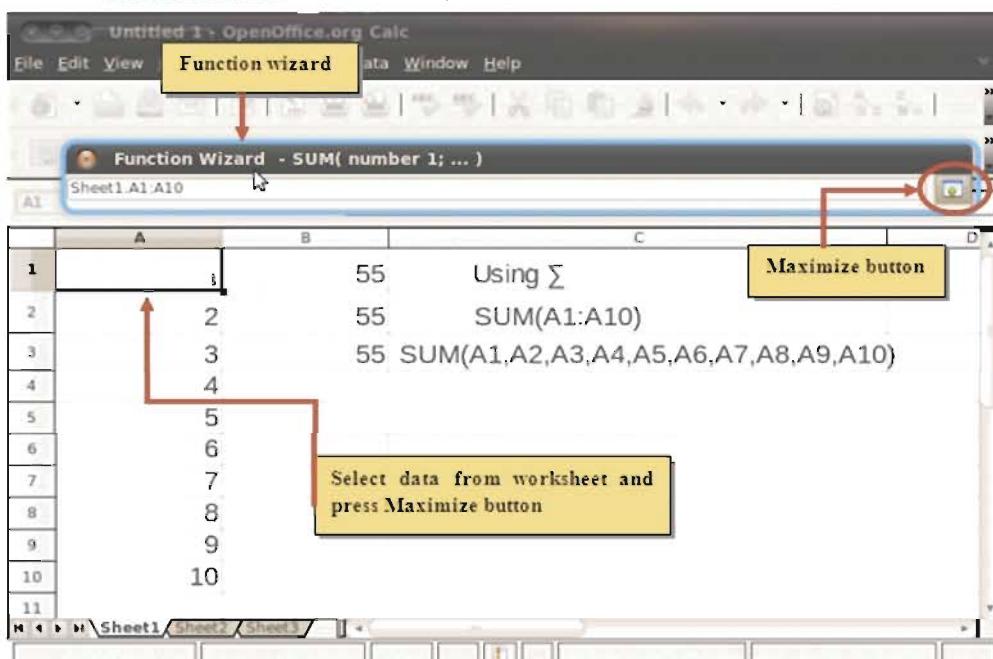


Figure 7.5 : Selecting a cell range using Select button

- After doing this, press **Enter** key or **Maximize** button (encircled in figure 7.5). Again full function wizard similar to the one shown in figure 7.5 will appear. Observe that the data range you have selected is visible in number 1 argument box.
- Click on **OK** button to finish and close.

Calc provides many built-in functions in various categories. The major categories of functions are mathematical, statistical, logical, and text formatting function. When you want to use another function as an argument, place the text cursor inside the original function's parentheses and choose the new function from the list.

Mathematical Functions

Math functions are very helpful when we work with numbers in Calc. Let us now see usage of some math functions.

Absolute Values of Numbers

The ABS function removes the minus sign (-) from a negative number and makes the number positive. It does nothing to 0 (zero) or positive numbers. The syntax is **ABS (n)** where n is a number. Figure 7.6 illustrates working of ABS function. You can also see the formula of the function in the formula bar (encircled).

1	Value of n:	-13
2	Function used:	=ABS(B1)
3	Value of ABS(n):	13

Figure 7.6 : Working of an ABS function

The ABS function is useful when you want only positive number. Consider that you are using an expression within another expression. The Calc first calculates inner expression and uses the inner expression value as argument of the outer expression. But the outer expression (such as square root) allows only non-negative (positive and zero) values. What if the inner expression returns a negative value? The negative value of the inner expression makes the outer expression invalid. To avoid this, we just take absolute value of the inner operation and send it to the outer expression.

Exponential Function

The EXP function returns the values of the exponential function e^x for the given number x where e is approximately equal to 2.718281828. The syntax is **EXP(Number)**. For example if we write =EXP(1) in a cell it will be considered as e1, which returns 2.72. Figure 7.7 illustrates this function. You can see the formula of function in formula bar (encircled).

1	Value of n:	1
2	Function used:	=EXP(B1)
3	Value of EXP(n):	2.71828182845904

Figure 7.7 : EXP function

Factorial Function

The FACT function returns the value of the factorial function for the given number (the product of all integers from 1 to the given number say n). The general form of the function is **FACT(n)** where n is an integer number.

For example if we write =FACT(3), the Calc will consider this as an expression ($1 * 2 * 3$) and return 6. Similarly FACT(6) returns multiplication of 1 to 6 numbers (that is $6*5*4*3*2*1$); which is 720. Figure 7.8 illustrates the factorial function with value 6.

	FACT	<i>f(x)</i>	X	✓	=FACT(B1)
A	B				
1	Value of n:		6		
2	Function used:		=FACT(B1)		
3	Value of FACT(n):		720		
4					

Figure 7.8 : FACT function

Natural Logarithm

The **LN** function returns the value of the natural logarithm (the logarithm base of e) for the given positive number. The syntax is **LN(Number)** where number is a positive number. For example, if we write =LN(8) in a cell it will return 2.08 (log_e8) as shown in figure 7.9.

	LN	<i>f(x)</i>	X	✓	=LN(B1)
A	B				
1	Value of n:		8		
2	Function used:		=LN(B1)		
3	Value of LN(n):		2.0794415417		
4					

Figure 7.9 : LN function

Logarithm Base 10

The **LOG10** function returns the value of logarithm base 10 for the given positive number. The general form is **LOG10(Number)** where number is a positive number. For example, LOG10(6) is considered as LOG106 and returns 0.77 as shown in figure 7.10.

	LOG10	<i>f(x)</i>	X	✓	=LOG10(B1)
A	B				
1	Value of n:		6		
2	Function used:		=LOG10(B1)		
3	Value of LOG10(n):		0.7781512504		
4					

Figure 7.10 : LOG10 function

Power

The POWER function returns the number raised to the given power. The general form **POWER(Number, Power)**. If we write =POWER (10,3); it will return 1000 (10³). It is equivalent to the notation 10³. See figure 7.11.

	A	B	
1	Value of n:	10	
2	Value of power:	3	=POWER(B1,B2)
3	Function used:		
4	Value of POWER(n, power):	1000	
5			

Figure 7.11 : POWER function

Product of Many Arguments

The PRODUCT function multiplies up to 30 arguments (x_1, x_2, \dots, x_{30}). Each argument can be a single value or a range of cells. The general form is **PRODUCT(n1,n2,n3,...,n30)**. Consider that we have entered some data in a worksheet as displayed in figure 7.12.

	A	B	C	D	E
1	10	1		3	
2	20	2		1	
3					
4	1200				
5					

Figure 7.12 : Product of many arguments using PRODUCT

Suppose we write =PRODUCT(A1:A2, B1:B2, D1:D2) in a cell. It will return 1200. Therefore, we can say that the product function considers all the values in a cell, a cell range or more than one cell range; and calculates its product. In figure 7.12 a total of 6 values are present in column A, column B and column D. Note that the column C is empty.

Square Root

The SQRT function returns the square root of a positive number. The general form of the SQRT function is **SQRT(Number)**. To find a square root of 100 =SQRT(100) is to be written simply in a cell. It will return 10 as shown in figure 7.13.

	A	B
1	Value of n:	100
2	Function used:	=SQRT(B1)
3	Value of SQRT(n):	10
4		

Figure 7.13 : SQRT function

Integer

The INT function rounds the given number down to the nearest integer. The general form of the function is **INT(Number)**. That is, if you write **INT(15.3)**, it will return 15 as shown in figure 7.14.

	A	B
1	Value of n:	15.3
2	Function used:	=INT(B1)
3	Value of INT(n):	15
4		

Figure 7.14 : Integer number using INT

Round

This function rounds the given number to the specified places. Many times a formula results in a number having lot many figures after decimal place. Example of such a number is 12.6546. If you want only two digits after the decimal point; obviously, you can not cut the number. You have to round it. The number 12.6546 is rounded to 12.65. Student result (total marks or percentage) is often rounded in such a way that it will result a complete number. That is, if a student gets percentages as 76.66, then it is considered that the student has secured 77 percentage. The general form of the ROUND function is **ROUND (Number, Places)**. Some examples of the round function are as mentioned :

=ROUND(76.6633,2) results in **76.6600**

=ROUND(45.1634,0) results in **45.00**

=ROUND(-56.5467,3) results in **-56.547**

In the second example listed above, we have rounded a number with 0 decimal place in order to generate a whole integer. Figure 7.15 illustrates the ROUND function.

	A	B	C	D
1	Value of n:	76.660000	45.560000	-56.547890
2	Decimal places:	2	0	3
3	Function used:	ROUND(B1,B2)	ROUND(C1,C2)	ROUND(D1,D2)
4	Value of ROUND(n,decimal places):	76.66	46	-56.548

Figure 7.15 : Examples of ROUND function

Just like ROUND, there are other two functions, ROUNDDOWN and ROUNDUP with same general form. Experiment these functions and identify their differences.

Truncate

Truncate function (TRUNC) takes two arguments; first argument is a number and the second argument is number of digits to be cut down from the number from its fractional part. The truncate function chops off the fractional part of a given number leaving some digits after decimal point, if specified. If there is no fractional part, leaving y digits after the decimal point (if the second argument is omitted), the TRUNC function cuts off the whole fractional part from the given number. The general form of the TRUNC function is **TRUNC(Number,Places)**.

See the following examples.

=TRUNC(1.239,2) returns 1.23 and digit 9 is lost.

=TRUNC(12.5) returns 12 and digit 5 is lost.

=TRUNC(-15.72) returns -15 and digits 7 as well as 2 are lost.

In the last example instead of TRUNC function try to use INT function and see the difference. You will find that the TRUNC function simply cuts off the specified part from the given number. It does not round the number. These examples are demonstrated in figure 7.16.

	A	B	C	D
1	Value of n:	1.239	12.5	-15.72
2	Decimal places:	2	--	--
3	Function used:	TRUNC(B1,B2)	TRUNC(C1)	TRUNC(D1)
4	Value of TRUNC(n,decimal places):	1.23	12.00	-15.00

Figure 7.16 : Examples of TRUNC function

See the examples mentioned in the ROUND function. Try to use the same examples with TRUNC function.

Statistical Functions

Besides typical mathematical functions Calc also offers you a variety of statistical functions on a series or range of values. Such statistical functions help in collection, analysis, explanation, and presentation of data in order to support forecasting, predicting as well as decision making activities. Let us see some popular statistical functions.

Average

The AVERAGE function returns the average of numbers given to it. The average is also considered as arithmetic mean. You have to sum up all the numbers in a list and divide the sum by the total count (size of the list). You may consider up to 30 such values separated with the semicolon sign (;). Instead of values, as you know, you may enter cell address or a valid cell range. The general form of the function is **AVERAGE(x₁,x₂,...,x₃₀)**. For example if you write **=AVERAGE(10,7,6,8,9,5)** in a spreadsheet cell, it will return 7.5. Figure 7.17 shows the average operation.

B3		f(x) Σ =	=AVERAGE(B1:G1)					
1	Numbers:		10	7	6	8	9	5
2	Function used:		AVERAGE(B1:G1)					
3	Average of Numbers:		7.5					

Figure 7.17 : Example demonstrating AVERAGE function

Mean

The GEOMEAN mean function returns the geometric mean of numbers given to it. The geometric mean of say n non-negative numbers is obtained by multiplying them all together and then taking the nth root. You may use up to 30 arguments (values, address or range). The general form of the function is **GEOMEAN(x₁, x₂, ..., x₃₀)**. For example if you write **=GEOMEAN(10,7,6,8,6,5)** in a spreadsheet cell, it will return **6.8**.

There is another type of mean function supported by the Calc, which is HARMEAN (harmonic mean). The harmonic mean has similar general form as **HARMEAN(x₁,x₂,...,x₃₀)**. Try to find harmonic mean of the same data using the above function.

Median

The median is defined as middle number of the group when they are ranked in order. If you can not find the exact one middle number (in case the total number of arguments in group is even), take two middle numbers and average them. The MEDIAN function returns the median up to 30 arguments given to it. The general form of the function is as **MEDIAN(x₁,x₂,...,x₃₀)**. For example if you write **=MEDIAN(10,7,6,8,6,5)** in a spreadsheet cell, it will return 6.5.

Mode

The MODE function returns the most common value in a data set from at most 30 arguments, which can be single values or ranges of cells. The general form is **MODE(x₁, x₂,...,x₃₀)**. For example, if you write MODE function as **=MODE(10,7,6,8,6,5)** it will return 6. Figure 7.18 illustrates use of different statistical functions.

	A	B	C	D	E	F	G
1	Numbers:	10	7	6	8	6	5
2	Function used:	AVERAGE(B1:G1)					
3	Average of Numbers:	7					
4							
5	Function used:	MODE(B1:G1)					
6	Mode of Numbers:	6					
7							
8	Function used:	GEOMEAN(B1:G1)					
9	Geo.Mean of Numbers:	6.822					
10							
11	Function used:	MEDIAN(B1:G1)					
12	Median of Numbers:	6.5					

Figure 7.18 : GEOMEAN, MEDIAN and MODE functions

CountA

The COUNTA function returns the total number of cells or arguments which have some value in it. You may give up to 30 arguments (values, address or range) separated with the semicolon sign (;). Some of the arguments may contain some value (of any type) some may be empty. The COUNTA function returns the total number of cells having some values in it. The empty cells will not be considered. The general form of the function is **COUNTA(x₁, x₂, ..., x₃₀)**. For example if you write **=COUNTA(10,7,6,8,6,5)** it will return 6.

Largest Value

The LARGE function returns the kth largest numeric value found in the given set of values x;

The general form is **=LARGE(x, k)** where x denotes the range of cells containing some numeric values; and k is the position.

This function considers the numbers specified in the range in a descending order and number at kth position is returned. For example, with set of values as **10, 7, 6, 8, 6 and 5** the LARGE function with value 2 returns value 8. The 8 number is 2nd largest value in the given set.

You can set the k argument to any integer value you like, but it must not be zero or larger than the number of elements in the given set of numbers.

Smallest Value

The SMALL function returns the kth smallest numeric value found in the given set of values x. The general form is **=SMALL (x, k)** where x denotes the range of cells containing some numeric values; and k is the position.

This function considers the numbers specified in the range in an ascending order and number at kth position is returned. For example, with set of values as **10, 7, 6, 8, 6 and 5**; the SMALL function with value 4 returns value 7 value, which is the 4th smallest value in the given set. See figure 7.19 illustrating examples of the LARGE, SMALL and COUNTA function.

	A	B	C	D	E	F	G
1	Numbers:	10	7	6	8	6	5
2	Function used:	COUNTA(B1:G1)					
3	Count of Numbers:	6					
4	Function used:	LARGE(B1:G1;2)					
5	Second Largest Value:	8					
6	Function used:	SMALL(B1:G1;4)					
7	Fourth Smallest Value:	7					
8							
9							
10							

Figure 7.19 : Examples of the LARGE, SMALL and COUNTA functions

Rank

The RANK function returns the rank of a given number in the given set of numbers. You need to provide a value (or address of the value) for which you need to find the position, the set of values (cell range) and order of the sorting; 1 for ascending and 0 for descending. The function will sort the values from the indicated set of values in ascending or descending order and returns the position (1st, 2nd, 3rd, and so on) of the given number.

The general form is **RANK(number, set, order)**. Suppose you have secured 67 marks in an examination. Your five friends have secured marks such as 47, 56, 78, 59, and 66. You know that among your friends, your rank is second; considering highest marks first, that is in descending order. This information you can find just by seeing the data; as you consider only your friends data; only six items. What if, there are so many items? Answer is simple. Use the Rank function. For example, to know your position (from the top, obviously!), you can use the Rank function =Rank (67, A1:F1, 0) it will return the position 2. Figure 7.20 illustrates RANK function.

Maximum

The MAX function can take up to 30 arguments (individually or as a range) and simply returns the maximum number within the arguments.

The general form is **MAX(x1, x2 ,..., x30)**. From the example mentioned earlier regarding students marks, to find out maximum (highest) marks you may use the MAX function as =MAX(67,47,56,78,59,66). It will return 78 as illustrated in figure 7.20.

	A	B	C	D	E	F	G
1	Numbers:	67	47	56	78	59	66
2	Function used:	RANK(67,B1:G1,0)					
3	RANK of 67:	2					
4	Function used:	MAX(B1:G1)					
5	Largest Value:	78					
6	Function used:	MIN(B1:G1)					
7	Smallest Value:	47					
8							
9							
10							

Figure 7.20 : RANK, MAX and MIN functions

Minimum

The MIN function can take up to 30 arguments (individually or as a range) and simply returns the minimum number within the arguments. The general form is **MIN(x1, x2 ,..., x30)**. From the example mentioned earlier regarding students marks, to find out minimum (lowest) marks you may use the MIN function as **=MIN(67,47,56,78,59,66)**. It will return 47 as illustrated in figure 7.20.

Calculation with Money

Calc also support other financial functions to aid calculation with money. Suppose one of your uncles comes to you and ask:

"Beta, just tell me using your computer, how many years should it take me to pay the money I borrowed from a bank, providing that I pay in equal amount of money every time and the interest rate is constant?"

You need not have to go for big calculation or to write a complex program. Your job is made simple by the Calc! You need to just write a function in a Calc cell as

=NPER(7.5%, -12000, 100000)

Here NPER is to calculate the period to repay your loan. The first argument (7.5%) is a constant interest rate. The second argument -12000 is the amount (in rupees) of annual instalment (it is negative, as we need to subtract this amount from the total amount borrowed every time an instalment is paid); here your uncle is paying 1000 per month and in a year he pays 12000 in total; and the third argument is total amount of loan. When you write this function in a Calc cell it will return 13.56 years ! This is what your uncle wants ! Figure 7.21 illustrates the same example.

	A	B	C
1	Total amount of loan taken:	100000	(Rs.)
2	Interest rate in %:	7.50%	(Constant rate)
3	Yearly total installment:	12000	(Rs. 1000 per month)
4	Function used:	NPER(B2, -B3, B1)	
5	Duration in years :	13.562227330178	(Years)

Figure 7.21 : Time to repay loan using NPER

The general form of the NPER function is **NPER(Interest, Instalment, Loan, Future, Type)**. The Future argument is used to enter the amount of money that remains after the last payment is made. Usually we repay whole amount of loans. The Type argument indicates if payments are made at the beginning of each accounting period or not. The Type argument value is equal to 1 when payments are made at the beginning of each accounting period else 0. The future as well as type arguments are optional and you may see that we have not used them in our example as we want to repay all our debts and each instalment we normally pay in the beginning of the account period; failing to which we have to pay more interest.

There are more such functions supported by Calc which deal with money management. Examples are interest calculation, net present value, future value, and time to maturity. Go to Calc help and explore them.

Making Decisions

The functions so far we have discussed are simply considering some arguments, process them and return some values. They really work well for us; however such functions are still not smart. A function is called smart if it is able to take some decision. For this, we need conditional functions that can make decisions based on a set of pre-defined conditions. A classic example is your result. Consider the following scenario discussing some predefined conditions about grading your performance.

If you get marks $\geq 35\%$ and $< 48\%$; you get pass class;

If you get marks $\geq 48\%$ and $< 55\%$; you get second class;

If you get marks $\geq 55\%$ and $< 60\%$; you get first class;

There are many such situations which need this type of descriptive logic and pre-defined conditions. Calc offers a set of such logical functions that simplifies the job. Let us have a look at some popular such functions.

Making decisions with IF

The prominent of all conditional functions used in the Calc is IF. It takes three arguments called Test, True and False. The Test expression/argument indicates the condition you want to test for. Calc calculates the expression and if result is true, it returns value indicated by True part; else False part.

Consider the following example and try it in a Calc worksheet.

- Write different values at cell A2 and B2.
- Go to cell C2 and write the following function.
`=IF(A2>B2,"Value at A2 is greater", "Value at B2 is greater")`
- Change values at the cell A2 and B2 and see what happens to the cell C2.

You can write a small formula to decide who the winner, of each game; is when yours and your friend's score are entered in two different cells as illustrated in figure 7.22.

	A	B	C
1		Me	My friend
2	Score of a game	13	17
3			
4	Result of the game:	My friend wins	

Figure 7.22 : Who is the winner ?

You can see the formula we have used in the formula bar (encircled). In the case of student's result example mentioned above it is required to build multiple if functions. One of them is given below.

Considering total marks of a student in percentages is stored in the cell A1. You may write If function as :

=IF(A2>=60,"First",IF(A2>=55,"Second",IF(A2>=48,"Third","Fail")))

In the above function, the first IF statement checks value of the cell A2; the entered marks of students in percentages. If value at the cell A2 is greater than or equal to 60, then the expression holds true and it returns the value indicated by the first expression; which is "First". If the percentages are less than 60, it leads to the second expression, which is another IF statements. You may go on adding many such If within Ifs; hence called nested if. Figure 7.23 explains it in an illustrated way. In the figure we have written multiple student names and their percentages to illustrate various possibilities of the IF statement. However, the formula is written once only (see encircled formula bar), and it is dragged (copied) for other students as shown in figure 7.23.

Figure 7.23 : Example of IF function

When it comes to bigger formulas, it is possible to loose the grip on it and easy to introduce errors. To avoid this, you may take help of function wizard. One advantage of using function wizard is that you need not have to type the exact name of the functions. Once you see and experiment the function wizard, you may select the **Structure** tab on this and see the structure of the formula you have written as shown in figure 7.24; which graphically displays the structure of the function creation. If there are errors, they are shown as red spots on the list.

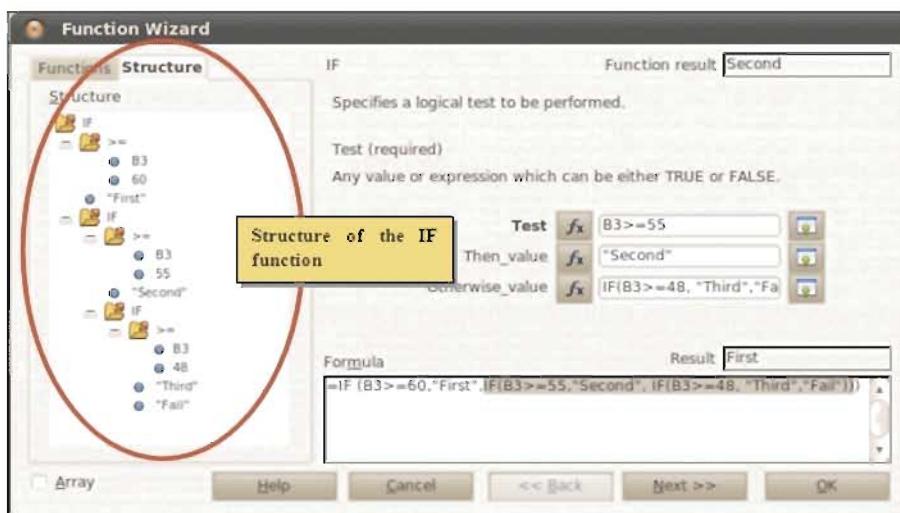


Figure 7.24 : Structure of IF function

Relational Operators

In the above example discussing IF statement, we have seen operators such as \geq . Such operators are known as the relational operators. Table 7.1 lists relational operators that you may need.

Relational Operator	Symbol	Description
Equal	=	Both the operands are (say A1 and A2) are equal Example:=IF(A1=A2, "A1 is equal to A2", "A1 and A2 not equal")
Less than	<	First operand is less than the second one Example:=IF(A1<A2, "A1 is less than A2", "A1 is not less than A2")
Greater than	>	First operand is greater than the second one Example:=IF(A1>A2, "A1 is greater than A2", "A1 is not greater than A2")
Less than or equal to	\leq	First operand is less than or equal to the second one Example:=IF(A1<=A2, "A1 is less than or equal to A2", "A1 is not less than or equal to A2")
Greater than or equal to	\geq	First operand is greater than or equal to the second one Example:=IF(A1>=A2, "A1 is greater than or equal to A2", "A1 is not greater than or equal to A2")
Different from	\neq	Both the operands are not equal, the first operand is not equal to the second one Example:=IF(A1<>A2, "A1 is different than A2", "A1 is not different than A2")

Table 7.1 : Relational operators

Logical Functions

The Calc support some more functions that support logical decision making. These functions are AND, OR, and NOT. These functions are used to build complex formulas and functions. This section describes the three logical functions mentioned above.

TRUE

This function returns the logical value TRUE. The general form of this function is **TRUE()**
FALSE

This function returns the logical value FALSE. The general form of this function is **FALSE()**
NOT

The general form of NOT is **NOT(expression resulting in logic value)**. NOT negates logical values, so FALSE becomes TRUE and TRUE becomes FALSE. See the following examples.

=NOT(6<>6) returns TRUE.

=NOT(1=1) returns FALSE.

Can we write NOT(TRUE) and NOT(FALSE)? Try this. Figure 7.25 illustrates some examples of TRUE, FALSE and NOT functions.

	A	B	
1			
2	Logical function	Value	
3	TRUE()	TRUE	
4	FALSE()	FALSE	
5	NOT (TRUE)	FALSE	
6	NOT (FALSE)	TRUE	
7	NOT (6<>6)	TRUE	
8	NOT (1=1)	FALSE	
9			

Figure 7.25 : Examples of TRUE, FALSE and NOT functions

AND

This function is used to compare results of up to 30 (thirty) arguments. The result of the AND function is TRUE if all of the conditions /arguments results in TRUE as well. The general form is **AND(Cond_1,Cond_2,...,Cond_30)**.

Here the arguments of the AND function are conditions, each capable of resulting itself in either TRUE or FALSE; that is Boolean values. Consider this example. If we need to check whether content of three cells A1, A2 and A3 are equivalent or not, we need to check if A1 and A2 are equal and then A2 and A3 are equal. If so, we may conclude that all three are equal. That is two conditions need to be formed using relation operator "="; which are respectively $A1=A2$ and $A2=A3$. AND function can be written as =**AND(A1=A2,A2=A3)**. Figure 7.26 illustrates an example as discussed above.

C5	f(x)	Σ	=	=AND(A1=A2,A2=A3)
1	A	B	C	
2	12			
3	12			
4	6			
5	Are the above values same? :			FALSE
6				

Figure 7.26 : Example of AND

OR

The general form is **OR(Cond_1,Cond_2,...,Cond_30)**.

The OR function is used to compare results of up to 30 (thirty) conditions. The value of OR is TRUE if at least one of the conditions tested is TRUE. For example, consider the formula we have used in the logical AND function described above. Let us change slightly the formula. Earlier we wanted all the three arguments A1, A2 and A3 must be same. Now we want to proceed if any two of them are equal. This can be written as =**OR(A1=A2,A2=A3,A1=A3)**.

The above function returns TRUE value in following conditions :

- if A1 is equal to A2;
- if A2 is equal to A3;
- if A1 is equal to A3; or
- If A1, A2 and A3 all are equal.

The function returns FALSE if any of them is not matching. That is in following conditions the function returns FALSE value.

- If A1 is not equal to A2; if A2 is not equal to A3; and if A1 is not equal to A3; and this happens altogether then only the formula is FALSE.

That is, OR function returns FALSE value only if all the conditions provided are FALSE. Figure 7.27 shows some examples of OR function while Table 7.2 illustrates some cases of logical function.

A4		$f_{(x)}$	$\Sigma =$	$=OR(A1=A2, A2=A3, A1=A3)$			
	A	B	C	D	E		
1	12	12	12	6	6		
2	12	12	6	12	12		
3	12	6	12	12	31		
4	TRUE	TRUE	TRUE	TRUE	FALSE		

Figure 7.27 : Example of OR

Cases			Expression Values	Result
Value of A1	Value of A2	Value of A3		
12	12	12	A1=A2 → TRUE A2=A3 → TRUE A1=A3 → TRUE	TRUE
12	12	6	A1=A2 → TRUE A2=A3 → FALSE A1=A3 → FALSE	TRUE
12	6	12	A1=A2 → FALSE A2=A3 → FALSE A1=A3 → TRUE	TRUE
6	12	12	A1=A2 → FALSE A2=A3 → TRUE A1=A3 → FALSE	TRUE
6	12	31	A1=A2 → FALSE A2=A3 → FALSE A1=A3 → FALSE	FALSE

Table 7.2 : Cases of the logical function $OR(A1=A2; A2=A3; A1=A3)$

String Functions

Joining Two Strings of Text

The CONCATENATE function can join up to 30 strings of text. The general form of the function is **CONCATENATE(text1,text2,...,text30)**.

You may use '&' sign instead of the function name. That is, to join two strings such as "ice" and "cream". You may write a function as shown below.

```
="ice"& "cream" or  
=CONCATENATE("ice","cream")
```

Both the above functions return a string "icecream". In case you want space between them, you may add a blank within any of the string as =**"ice "& "cream"** or =**"ice"& " cream"**.

If the strings to be joined are stored in the separate cells (say A1 and A2), you could also use the function as =**A1&A2** or =**CONCATENATE(A1,A2)**.

Perform the following exercise.

- Go to cell A1, write your first name.
- Go to cell A2, write your middle name.
- Go to cell A3, write your surname.
- Go to cell A4, write a blank as " ".
- Use & function in cell C3 as =**A1&A4&A2&A4& A3**

Figure 7.28 demonstrates some concatenation examples.

	A	B	C
1	ice		
2	+ cream		
3			
4	icecream		
5			
6	Use of & operator		
7	"ice"&"cream"	icecream	
8	"ice" & " " & "cream"	ice cream	
9	B1 & " " & B2	ice cream	
10	B1 & B2	icecream	

Figure 7.28 : Examples of CONCATENATE function

LOWER

This function turns all given characters into lowercase. The general form of the function is **LOWER(text)**. For example =**LOWER("School")** returns "school".

Upper

This function turns all given characters into uppercase. The general form of the function is **UPPER(text)**. For example =UPPER("School") returns "SCHOOL".

Proper

This function turns first letter of all words into uppercase and remaining alphabets in lowercase letters. The general form of the function is **PROPER(text)**. For example =PROPER ("I love my scHool") returns **I Love My School.**

The string functions lower, upper and proper are normally used for formatting the documents. A properly formatted document has good readability and clarity.

Roman and Arabic

The Roman function converts the number provided into its equivalent Roman number. The general form is **ROMAN(number, mode)**.

Please note that Roman numerals can only represent numbers from 0 to 3999; therefore the number argument you provide must be within this range. As you know a Roman number can be written in many ways. Suppose you want to write number 8 in Roman. You may write it as VIII; however, the number 8 can also be written as IIX. The second argument of the function is about such mode of representation, which used to simplify the R the resulting Roman number. The Mode operator can take values 0, 1, 2 or 3. TV and movie industries use Roman numerals to encode their film strips and other material.

The Arabic function is reverse of the roman function, it considers a roman number as its arguments and convert the number into an Arabic number.

See the example given below.

=**ROMAN (125)** which returns **CXXV**.

Find out what the function =**ARABIC("MMXI")** returns ? Is it **2013** ?

Figure 7.29 shows some examples of ROMAN and ARABIC functions.

	A	B	C
1	Function	Value	
2	ROMAN(125)	CXXV	
3	ARABIC("mmxiii")	2013	
4	ARABIC(ROMAN(2013))	2013	
5			
6	Also consider following examples to see how mode works		
7	ROMAN(999)	CMXCIX	
8	ROMAN(999,0)	CMXCIX	
9	ROMAN(999,1)	LMVLIV	
10	ROMAN(999,2)	XMIX	
11	ROMAN(999,3)	VMIV	

Figure 7.29 : Examples of ROMAN and ARABIC functions

Trim

This function removes extra spaces from strings. That is if you write **TRIM("I Love My School!")**, it will return **I Love My School !**

The general form is **TRIM(text)**.

Comparing Strings

EXACT compares two strings and returns 1 if they are exactly the same or 0 if they are not. For example =**EXACT("Blue", "Blue")** returns 1, while =**EXACT("Blu", "Blue")** returns 0.

The general form is **EXACT(s1, s2)**.

Substring

There are two functions in Calc, one for left side substring and another for right side substring.

The **LEFT** function returns the first character from the string text, or a specified number of n characters starting from the left.

The general form is **LEFT(Text, n)**. That is to extract first 4 left characters from a string you may write =**LEFT("Ram Dhashrathbhai Patel", 4)** returns "Ram".

Similarly the **RIGHT** function returns the last n characters from the end (right side) of the string text. That is if you write =**RIGHT("Ram Dhashrathbhai Patel", 5)**, it will return "Patel". The general form is **RIGHT(Text, n)**.

Calc also provides utility to find middle of a string. The general form is **MID(Text, Start, n)**.

That is to extract middle name from the arguments given in the function; we need to write function as =**MID("Ram Dhashrathbhai Patel", 5, 13)** which returns "Dhashrathbhai".

Length of a String

The **LEN** function returns the length of the given string. The general form is very simple, which is **LEN(Text)**. That is, if you write **LEN("Ram")** it will return 3. Figure 7.30 displays some examples of string function.

	A	B
1	Function	Value
2	=TRIM("I Love My School")	I Love My School
3	=EXACT("Blue", "Blue")	TRUE
4	=EXACT("My School", "New School ")	FALSE
5	=LEFT("Ram D Patel", 3)	Ram
6	=RIGHT("Ram D Patel", 5)	Patel
7	=MID("Ram D Patel", 5,1)	D
8	=LEN("Ram D Patel")	11

Figure 7.30 : Examples of string functions

Time and Date Functions

Calc offers functions for time and date calculations. For calculation that involve shorter period (such as hours, minutes, or seconds), you may use time functions and for bigger period (such as weeks, months, or years), you may use date functions.

Today's Date

The TODAY function returns values for the today's date stored in your computer. It is function without arguments. Suppose today is 11th July, 2013; then the today() function returns 11/07/2013 or 11. Jul. 2013 (as per the format set in your computer).

Date

The DATE function returns the valid date for the given numbers for year, month, and day. That is if you experiment =DATE(2013, 1, 1), it will return 1/1/13.

Day

The DAY function is used to convert a date given in any recognizable format into a day of the month. For example, today's date is 11th July, 2013; then =DAY(today()) returns 11.

We may also write the function as =DAY(DATE(2013,7,11)). The general form is DAY(Date).

To ensure the argument provided to a date type function is in the date format, you may use DATE function as we have used. The DATE function will first convert the numbers into a valid date format and then returns the month number of the date.

Weekday

The WEEKDAY function is used to convert a valid date given into a day of the week. Days are numbered from 0 (Sunday) to 6 (Saturday). For example, if you write =WEEKDAY(DATE(2013,7,11)), it will return 5.

The general form is WEEKDAY(Date, n). Here date is any valid data and n is a number that indicates week starts from which day. You may use following values for n.

N=1 : week starts on Sunday (day 0).

N=2 : week start on Monday (day 1).

If you do not specify the argument, it will take value 1, that is, the days are numbered from 0 (Sunday) to 6 (Saturday) as shown in the example above.

Here are some more examples.

=WEEKDAY(DATE(2013,7,11),0) returns 3

=WEEKDAY(DATE(2013,7,11),1) returns 5 (default); which is equivalent to
=WEEKDAY(DATE(2013,7,11))

=WEEKDAY(DATE(2013,7,11),2) returns 4

Month

The MONTH function is used to convert a valid date into the month number. For example, if you write **=MONTH(DATE(2013,7,11))** in an Calc cell, it will return 7.

The general form of the function is **MONTH(t)**.

Year

The YEAR function is used to convert a valid date into the year number. For example, if you write **=YEAR(DATE(2013,7,11))** in an Calc cell, it will return 2013.

The general form of the function is **YEAR(t)**.

Number of Days in a Year

The DAYSINYEAR function returns the number of days in the year presented by given date within the function. Try following function in a spreadsheet cell.

=DAYSINYEAR(DATE(2013,7,11))

The function returns 365.

The general form of the function is **DAYSINYEAR(t)**.

Earlier you might have written a complex formula or a special program to check whether a given year is leap year or not. Instead of this, you may just check no of days in a given year.

Difference between Two Dates

The DAYS function calculates the number of days between two dates. Note that the name of function is DAYS instead of DAY. The earlier (DAY) function returns a day number of month. The DAYS function provides number of days between two valid dates.

=DAYS(DATE(2012,7,11), DATE(2013,7,11)) returns 366. As the year 2012 is a leap year containing 366 days in it !

Use this function in this way.

=DAYS(A1, "NOW())

Here A1 is a cell address where you have to write your birth date. The second argument is **NOW()**; which will return current date stored in your computer along with time. Guess what will it return? Yes, you are correct, it will return your age in number of days! With this you can exactly find out who is the eldest in your class!

The general format of **NOW()** function is the same; **NOW()**. It is a function without argument.

Note that when second date is subtracted from the first date, many times the function returns a negative value. To avoid this, you may use the ABS function. Figure 7.31 demonstrates some date and time functions.

	A	B
1	Function	Value
2	TODAY()	11. Jul. 2013
3	DATE(2013,7,11)	July 11, 2013
4	DAY(DATE(2013,7,11))	11
5	WEEKDAY(DATE(2013,7,11))	5
6	WEEKDAY(DATE(2013,7,11),0)	3
7	WEEKDAY(DATE(2013,7,11),1)	5
8	WEEKDAY(DATE(2013,7,11),2)	4
9	MONTH(DATE(2013,7,11))	7
10	YEAR(DATE(2013,7,11))	2013
11	DAYSINYEAR(DATE(2013,7,11))	365
12	DAYS(DATE(2012,7,11), DATE(2011,7,11))	366
13	WEEKS(DATE(2011,7,11), DATE(2012,7,11),0)	52
14	YEARS(DATE(2011,7,11), DATE(2013,7,11),0)	2

Figure 7.31 : Date and time functions

Number of Weeks Between Two Dates

If there are two functions called DAY and DAYS; Similarly the WEEKS function calculates the number of weeks between two dates.

=WEEKS(DATE(2011,7,11); DATE(2013,7,11),0) returns 52.

The function uses two DATE functions to convert the arguments given into valid date formats. Beside this, the WEEKS function also uses a '0'; which indicates a week type. The '0' value indicates weekly intervals. Instead of '0' you can use calendar weeks indicated by '1'.

The general form of the WEEKS function is **WEEKS(t1, t2, Week type)**. Just as DAY and DAYS are different, similarly WEEK and WEEKS are different functions.

Number of Years Between Two Dates

The YEARS function calculates the number of years between two valid dates, either in yearly intervals (indicated by 0) or calendar years (1). For example, if you write:

=YEARS(DATE(2011,7,11), DATE(2013,7,11), 0), it will return 2.

You may find out how old you are using this function. Just find out number of years from your birth date and current date!

Functions in Other Spreadsheet Packages

Most of the spreadsheet packages support similar functions. If you take example of Microsoft Excel, it also supports most of the functions discussed in this chapter. You may have to change some minor syntax, such as little difference in spelling or use of ';' (semicolon) or ',' (comma) between the arguments of a function.

Package like Google Spreadsheets [doc.google.com] also supports the similar functions. As you know, it can be useful on the Internet platform, mobile phones and any machine that supports the standard infrastructure (hardware/software) such as Android operating system.

In addition to these, the Google packages also offer you free spreadsheet templates for various jobs such as creating students' schedule, grade reports, project planner, attendance of students and employees, managing personal budget, wedding list, managing invoices, etc. These templates have ready framework such as sections, row & column headings, formulas and script required for necessary calculations. Applications such as Google spreadsheet are becoming popular because of platform independence and mobility.

Summary

In this chapter we learnt different mathematical, statistical, string, date & time and formatting functions. We have also seen that how the functions can be entered through functions wizards. Moreover, the function wizard will also be helpful in pointing and correcting errors while working with functions.

EXERCISE

1. Explain working of function wizard in brief.
2. List any three mathematical functions of your choice with proper example of each.
3. List any three statistical functions of your choice with proper example of each.
4. List any three decision making functions of your choice with proper example of each.
5. List any three time and date functions of your choice with proper example of each.
6. List any three string functions of your choice with proper example of each.
7. Explain If function in Calc in detail with suitable example.
8. List the three logical functions with proper example of each.
9. Choose the correct option from the following :
 - (1) In which of the following ways we can enter a function in Calc ?
 - (a) Directly typing function name in a cell
 - (b) Function wizard or selecting from tool bar
 - (c) A and B both
 - (d) Depends on functions
 - (2) A function can start with which of the following options ?
 - (a) '=' sign
 - (b) Alphabets
 - (c) Numbers
 - (d) Any of these
 - (3) Which of the following is not a logical function?
 - (a) OR
 - (b) AND
 - (c) NOT
 - (d) PROPER
 - (4) Which date and time function of Calc you will use to find out whether the given year is leap year or not ?
 - (a) DATE
 - (b) TIMESTAMP
 - (c) YEARS
 - (d) YEARDIFF

(5) Which is the possible mechanism to enter a function in Calc cell ?

- (a) Through function wizard
- (b) By manually entering
- (c) Both (a) and (b)
- (d) Depends on function

(6) How many arguments one can use with a Calc function ?

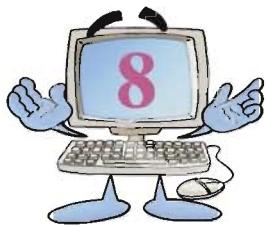
- (a) One
- (b) Two
- (c) Three
- (d) Depends on the function

(7) What can be a function argument ?

- (a) Value
- (b) Text
- (c) Other function
- (d) All of these

LABORATORY EXERCISE

1. Consider your marks of different subjects and prepare a simple mark sheet containing school name, student's number, student's name, standard and marks. Also find percentages and class (first class, pass, fail etc.) from the marks provided. You may use your last year school report card.
2. Simulate the electricity bill of your house using Calc.
3. Ask your family member or neighbour about the loan information, if any. Use NPER function to calculate the time to repay the loan.
4. Write three numbers in three different cells. Use values in such a way that it forms a valid date. Use date function to convert these values into a valid date.
5. Take a date and find out that whether the year is leap year or not.
6. Try to find out information spreadsheets packages for mobiles, iphones and ipads. Find out name of the package/spreadsheet that offers this facility.



Charts in Calc

With the advancement of technology and the increased need of technical analysis, the use of charts has increased. Utility of chart is one of the factors that make the spreadsheet packages widely popular. Charts are illustrations of a business situation with ability to display a significant amount of information in attractive way. A chart is also considered as a graphical presentation of numerical data.

To prepare a chart, one needs to have basic knowledge about various charts, concept underlying the chart patterns and their applications. That is, just by selecting a chart type and creating a chart through a spreadsheet package will not contribute much in business. You must have understanding of the chart concepts such as how the chart would be helpful in your business. One must also know when to use a particular type of chart, what information it conveys, what support it provides and which type of decisions can be made using such chart.

Type of Charts

According to the nature of applications and requirements, a chart type should be selected. Charts are used for various purposes such as telling history; evaluate alternatives, presenting trends or find-out exceptional cases. That means, an incorrect choice of chart can lead to poor representation of the concept and generates misunderstanding. On the other hand, correct choice of chart can lead to right and faster decisions. Popular reasons why we should go for a chart are mentioned below :

- For comparisons;
- For demonstration of distribution;
- For understating of situation;
- For analysis of trend over the time;
- For investigating deviations; and
- For identifying and understanding the relationship between entities;

Follow the steps given below to prepare chart in any situation :

- Make it clear what you want to say;
- Collect and arrange data;
- Remove invalid data;
- Determine the best chart type; also check the data you have collected for the chart are sufficient or not;
- Prepare the chart; and
- Format the chart.

As stated, the very first step about the chart making after identifying its objective; is to prepare data. Once data are entered in a spreadsheet document, you can view it in a graphical manner.

Inserting a Chart

To insert a chart in a worksheet do the following.

- Select the data range in a spreadsheet;
- Select **Insert → Chart**;

Consider you have given some surprise quizzes and class tests from January to April in an academic year and obtained marks as indicated table 8.1.

	Quiz marks	Test marks
January	35	27
February	28	30
March	37	42
April	31	33

Table 8.1 : Marks of a student

Open a spreadsheet document and enter the data as shown in table 8.1. Do not change the order of the data, otherwise when we plot a chart, it will give different result. Also, save the data frequently.

Once you have entered all the data, select the data range. Here the data ranges from cell A1 to cell C5. Keep the data range selected, now choose option **Insert**. A vertical submenu appears. Select **Chart** option from it. The operation sequence is shown in figure 8.1.

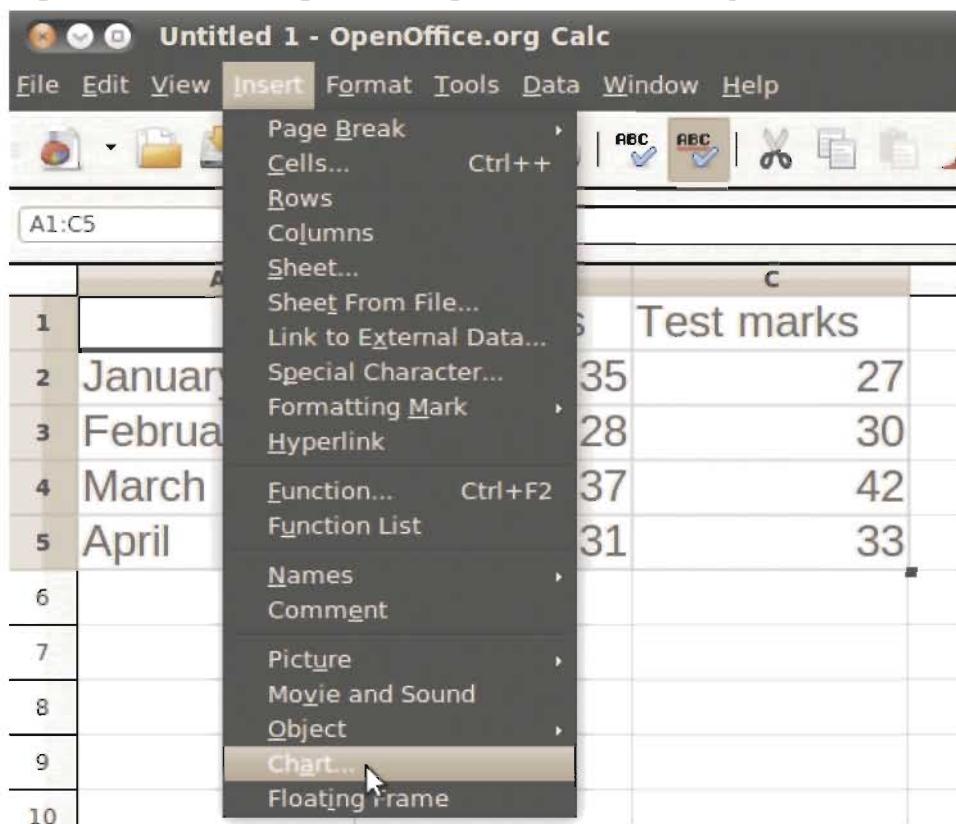


Figure 8.1 : Inserting a chart

When you select the option for inserting a chart, you can see a chart wizard. The chart wizard of Calc is actually a set of simple and user friendly steps to perform charting operations in the spreadsheet package. What one has to do is just select the data arrange and invoke the chart wizard. Rest of the things are taken care by the wizard. The chart wizard dialog box is as shown in figure 8.2.

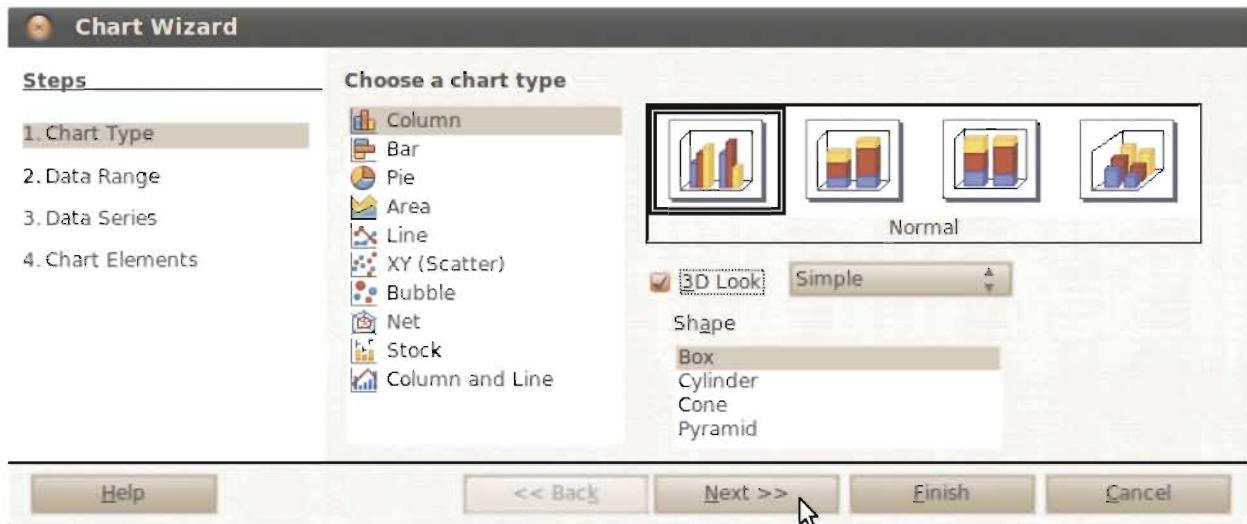


Figure 8.2 : Chart wizard

The Chart wizard as shown in figure 8.2 lists 4 steps seeking information about chart type, data range, data series and chart elements. These steps are discussed below:

Selecting Chart Type

As shown in figure 8.2, select a column chart. You can also see preview of how a typical column chart looks. The preview updates every time you select a different type of chart, and provides a good idea of what the selected chart will look like.

The chart wizard has the following main components.

- List of steps involved in setting up the chart;
- List of chart types;
- Preview of chart types;
- Options for each chart type; and
- Options to proceed further or going back and modify the choices.

You may see other chart types along with their respective preview in the chart wizard. The chart types are Column chart, Bar chart, Pie chart, Line chart, Area chart, XY chart, Bubble chart, Net chart, Stock chart and Column chart. Calc offers a choice of 10 basic chart types. Once you select a chart type, you may choose further options such as look of a chart and shape of the chart. The Calc also offers a few options for each type of chart. Options vary according to the type of chart you pick. At any time you can go back to a previous step and change selections.

There is a 3D view box also. If you select the 3D box (by clicking on it), you may see the changed preview. Only those types which are suitable for 3D (such as column chart, bar chart, pie chart, and area chart) give you an option to select a 3D look. You may also select shape of the chart such as "Box", "Cylinder", "Cone" and "Pyramid" by selecting appropriate choice in the Shape box. Appropriate preview is shown as and when you made choices.

For our example, we have selected following choices :

- The chart type is Column chart;
- The chart look is **3D** with **Simple** view;
- The chart shape is **Box**.

At the end, click the Next button. You may observe that the Calc has prepared a default chart in the background of the chart wizard. At this stage, if you select the Finish option, then you see the default chart, which you may modify or format afterwards. You may choose Back button to go back and modify your choices.

Data Range

Once you have selected the chart type, you need to provide necessary data. One possibility is that you have already selected a range of data before clicking the Insert → Chart option; as we did. If not so, we can still select it by clicking on the Data Range option as shown in figure 8.3.

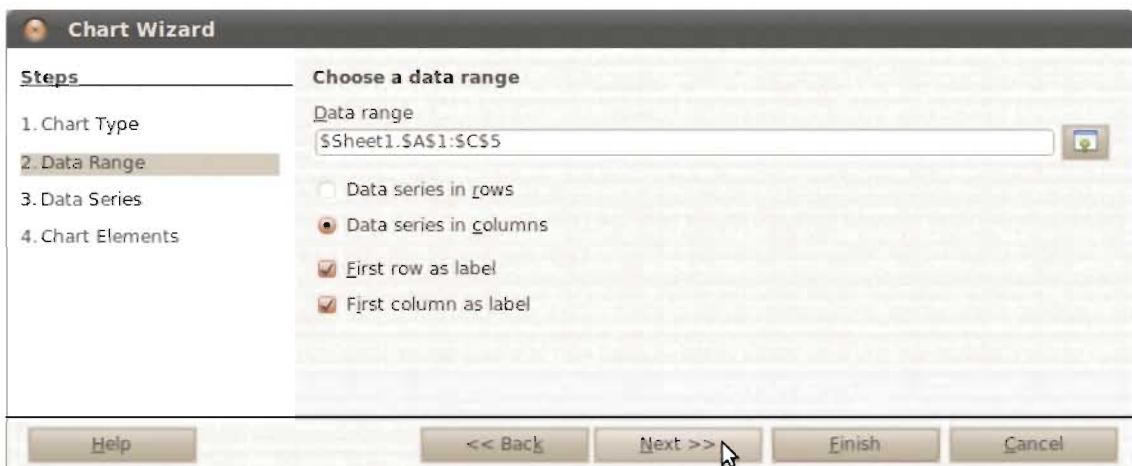


Figure 8.3 : Data range selection

Calc is smart enough to identify data range even if the part of the data range is selected instead of the complete data range prior to insertion of a chart. That is, suppose you have selected only a cell such as B3, and you choose **Insert → Chart**, even though, Calc automatically guesses the data range. However, it is a good practice to verify the data range. You may use already existing data from other spreadsheet documents or other compatible documents as mentioned below.

Calc charts can be based on the following data :

- Spreadsheet values from Calc cell ranges.
- Cell values from a Writer table. You may create the data in packages such as Writer, Draw, or Impress, and you can copy and paste them into the Calc spreadsheet.
- Any properly formatted text file or table.

In our case the data are coming from the same sheet and its range is from A1 to C5. Note that the Calc has considered the data range as \$Sheet1.\$A\$1:\$C\$5. That is, the Calc converts the range into an absolute address. Hence, if you move your chart to any other location, the data range remains same (absolute).

The next button is asking about the location of data series, in rows or as columns. Here we have

entered our marks in columns. Hence, we can select option **Data series in column**. Also, we need to select the option **First row as label**. This option is useful when the data contains multiple columns and each is having a heading. The first row makes all heading for every column. By mentioning the first row contains all headers, we can specify labels of columns. In our example of student's marks, the first row contains headings such as **Quiz Marks** and **Test Marks**.

Similarly, when every row has a heading, you may select **First column as label** option. You can select both the checkboxes as well.

Data Series

The chart wizard also allows you to customise data ranges for the chart preparation. Here we use values selected automatically by the chart wizard. Figure 8.4 shows the default data series considered by the chart wizard.



Figure 8.4 : Data series selection

You may add or remove the data series using this dialog box. To add data series, such as final exam marks in the current example, use **Add** button. To remove any data series, select the series, which you want to delete and choose, **Remove** (see mouse arrow in figure 8.4). The dialog box also provides facility to enter range name and range values. Here range name is \$B\$1, having value "Quiz marks". You can use **Select** button (encircled with red line in figure 8.4). It will minimize the chart wizard and lead you to the worksheet area. You may select appropriate cell. When you complete the selection, press the **Maximize** button of the chart wizard. This will be helpful if you do not remember the location of the cell(s).

Similarly, you can select values of the data series. In our example, the data series values are the quiz marks located within cells B2 to B5. You may write it as \$Sheet1.\$B\$2: \$B\$5.

The data series phase of the chart wizard also lets you enter the Categories of the data. In our example, data series categories are the month names located within cells A2 to A5. You may write it as \$Sheet1.\$A\$2: \$A\$5. Here also we can use the **Select** (encircled with blue line in figure 8.4) button to minimize the chart wizard and select the values directly from the spreadsheet. In the end, choose Maximize button.

Data Elements

Chart wizard lets you to enter values of different chart elements such as title, subtitle, X-axis captions, location of legends and options to display the grids. Figure 8.5 shows a screen of chart wizard that allows us to set data elements.



Figure 8.5 : Chart elements

The chart elements such as title of the chart, sub title of the chart and information about axis and legends are provided through this phase of the chart wizard. As shown in figure 8.5, the first option within the **Chart Elements** option is **Title**. Here you can provide main title of the chart. The second option is **Subtitle**, which lets you enter the subtitle of the chart. The title and subtitles are placed on top of the chart within the chart area. Once you are done with the title and subtitle, you may provide information about the axis. Title information regarding the three axes namely X axis, Y axis and Z axis can be provided here. You can also choose to display legend or not at this stage using **Display legend** option. If you have opted for **Display legend** option, you need to provide further choice about where you want to display the legends; left, right, top or bottom of the chart. By default, the legend is visible and placed right side of the chart within the chart area.

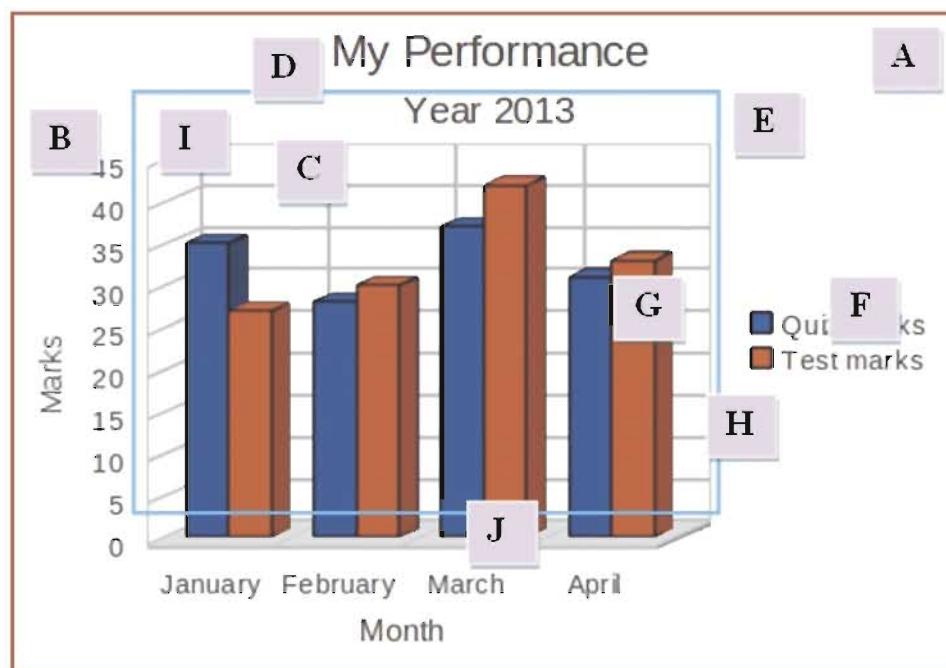


Figure 8.6 : Simple chart and chart elements

Once the correct options are given, you may click on Finish button to create chart. The chart is now placed in the current sheet. You can see the chart as shown in figure 8.6 with labels to the chart's elements. The elements of the charts are discussed in next section.

Elements of a Chart

A chart generally has components such as chart area, chart title and subtitles, chart axis with values, chart data area axis, chart data area, legends for the chart axis, data series, and chart floor. We have already created a simple chart showing a student's performance in quizzes and tests. To know about the specific component of a chart, you may just put the mouse arrow on it. It will display the component name. Let us examine some elements in the Calc chart. Figure 8.6 also shows some labels indicating elements of chart. These labels are described below :

A: Chart area : The chart area is the rectangular area in which the chart is embedded.

B: Y axis with values : The Y-Axis values are taken automatically according to the data we have provided. Here we have given marks not more than 50, the Y axis values are considered as 0 to 45.

C: Data area axis : The data area axis is the axis within the data area. By choosing this option, you may see vertical or horizontal (or both) grid lines on the background of the chart. The main utility of the axis within the data area is to identify value, if the chart does not display it.

D: Title and Subtitle : The title of the chart is a name given to the whole chart. Here we have given the title as My Performance. Along with the title, we may also give subtitle. The purpose of the subtitle is to indicate additional information or the purpose and use of the chart. Here we have used Year 2013 as subtitle.

E: Data area : The data area within the chart area actually holds the chart. It is also known as Chart. The chart is a core part of the chart area, hence usually placed in center of the chart area. The area surrounded by blue coloured line boundary in figure 8.6 is known as chart or data area.

F: Legend : The chart is a kind of drawing. It must have some text to indicate about its data and graphics. The chart shown in figure 8.6 has columns of blues and reds. From title, X-Axis and Y-Axis labels you can learn that the chart is about marks of various tests in different months. However, it is not clear which type (colour) of the columns represents which marks. To accommodate such information in a chart, legends are used. Here, in figure 8.6, we have used legend indicating blue colour for Quiz marks and red for Test marks.

G: Data series column : Chart may contain more than one data series. For every data series a colour is chosen and chart is build. For example, for a column chart, for each data series a colour is picked up and many columns of that colour are developed representing various values at different period of time/stage. Here the marks of quizzes are considered as a data series and a blue colour is chosen automatically. Calc prepares many columns of blue colour, each representing a mark in a month. These blue colour columns in figure 8.6 actually represent a data series for Quiz marks. Similarly, red colour columns in figure 8.6 actually represent a data series for Test marks.

H: Chart floor : The chart floor is the horizontal plane which provide base to the chart elements. For particular type of chart, such as bar and column charts, it is quite useful. You can see in figure 8.6 the chart floor is filled with grey colour.

I: Chart wall : The chart wall is the vertical plane which provide base to the Y axis of the chart. As shown in figure 8.6 it is not filled with colour. However, you may modify that to fill a colour of your choice.

J: legends on X-Axis : Consider a simple chart is given to you. The X-Axis of a chart has only values on it. Just like in figure 8.6, we have used month names on X-Axis, such as "January", "February", etc. These numbers are self explanatory. One can easily know that we are talking about

months of the year. What if we want to mention speed of a vehicle ? Of course, we can write 10, 20, 30 etc. But is there any way to specify these are kilometre per hour; or meter per minutes? These units and categories of the items on X-Axis can be specified with the legend on X-Axis. Similarly, we can specify the legend for the Y-Axis too.

An Alternative Way to Create a Chart

Alternatively, you can select the chart icon from the standard tool bar. If you cannot see the standard toolbar, you can make the toolbar visible by View ' Toolbars options as discussed earlier. Figure 8.7 shows the chart icon (encircled) on the toolbar.



Figure 8.7 : Chart Icon

You need to perform just three steps.

1. Enter data into a sheet.
2. Select it and press chart icon as mentioned above.
3. Select **Chart type** and click **Finish**. You may not give other options at this stage.

When you click the Finish button, your chart is ready and placed in the current sheet. Your chart may not be having all titles and legends. However, if you like the chart, you can always format it to make it more effective and attractive. The following section explains ways of formatting and modifying charts.

Formatting and Modifying Charts

Many times you need to customize the chart according to your requirement. That is, you may wish to modify the chart according to your style. Let us discuss different formatting options of the chart.

Modifying the Place and Size of the Chart

When a chart is created it assumes the default appearance and placement, usually in current worksheet. Sometimes you do not like the location of the chart where it is placed by default. Sometimes you do not like its size. You can move the whole chart by selecting it with just a left click and dragging it (by its handle with mouse) to a desired position. You may copy it or cut it and paste at a new location. Similarly, you can resize the chart by selecting the chart and dragging it using mouse through its handle. See figure 8.8.



Figure 8.8 : Modifying Chart Size

At times after the chart is prepared you may need to change some values in the data, feel free to change the values. The chart will be automatically changed. After a chart is inserted or selected in a spreadsheet using mouse, the toolbar of the Calc package changes to show chart buttons and commands. When the corresponding data in the cells are changed; the chart changes as well.

It is possible not only to format the chart, but also modify the chart. The following section describes various ways to format or modify a chart.

The first thing you may notice about the chart, besides its location and size, is its type. Say, from the column type of chart, you want a line chart now. The first thing you must remember is to bring the chart into edit mode. To do so double (left) click in a **Chart Area**. A grey border appears around the chart as shown in figure 8.9. You may right click the chart area to modify its various parameters.



Figure 8.9 : Chart Area

Modifying Chart Type

To modify the chart type, perform the menu command **Format → Chart Type**. Alternatively, you may press the **Chart Type** icon on the chart formatting toolbar as shown in figure 8.10.



Figure 8.10 : Chart formatting toolbar

The chart formatting toolbar appears when you select the chart. The chart formatting toolbar is used to format the typical chart elements such as chart area, chart data series, chart axis, chart titles, etc. In figure 8.10 the third icon encircled with a blue colour line is about chart type. When you click the encircled icon, it will display a dialog box. The options in the dialog box allow as changing the chart type and look of the chart such as 3D chart and shape of the chart such as Box or Pyramid.

You may directly right click on the chart to modify its type. When you right click, a small vertical menu appears with possible operations you can perform at the current stage.

Formatting the Chart Area

As mentioned above, chart area is an element of a chart. When you click on a chart, the chart toolbar is activated as shown in figure 8.11. The first icon in the toolbar is presenting a list of all the elements you can format. You can select any particular chart element which you would like to format from this list. For this, use small triangular up and down arrow keys shown with the elements box as shown in figure 8.11. See the encircled portion with red line in figure 8.11. Once you select the appropriate element then choose the next option **Format Selection**. See blue colour circle in figure 8.11. In the same figure (8.11), we have selected **Chart area**. However, you can see the list of other elements of the chart. After selecting the **Chart area**, next step is to select **Format selection** button.

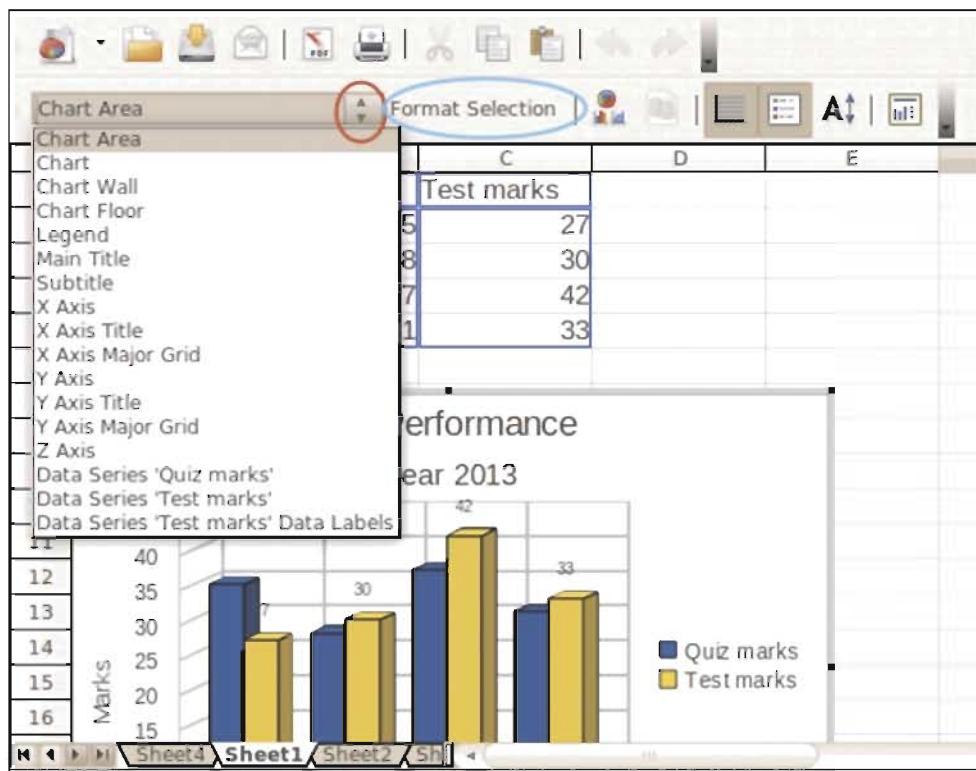


Figure 8.11 : Formatting chart area

This will allow us to format the selected chart element by providing suitable dialog box. The dialog box shown in figure 8.12 allows us to format border of the area, Area of the chart and transparency of the filled colour. The borders are by default invisible, however, you can make them visible. Further, you can also select border style such as continuous line, dotted line, thick line, etc and border colour. The area of a chart can also be filled with various colours, gradients or a bitmap image with different transparency levels. If you do not want to fill uniform colour (with or without transparency) in a chart area, you may wish to give gradient to the colours in linear, radial or axial directions.

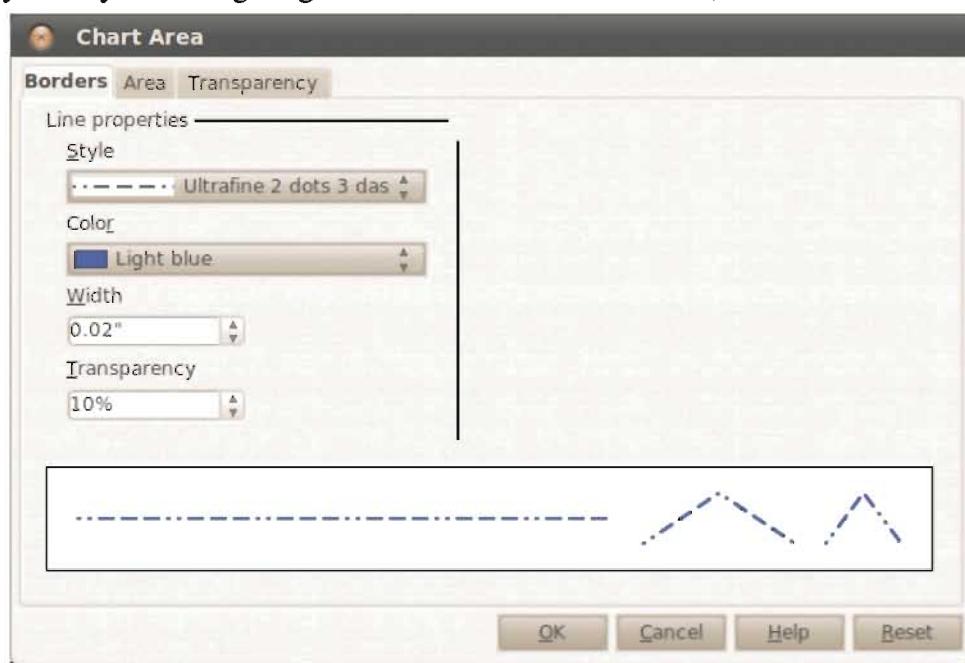


Figure 8.12 : Modifying the chart area

Formatting Titles to the Axis

To add titles to X-Axis and Y-Axis for better understanding of the chart, you can perform following actions.

- Select the chart.
- You can see the chart formatting toolbar as shown in figure 8.11. Select X-Axis title as an element to be formatted from the toolbar.

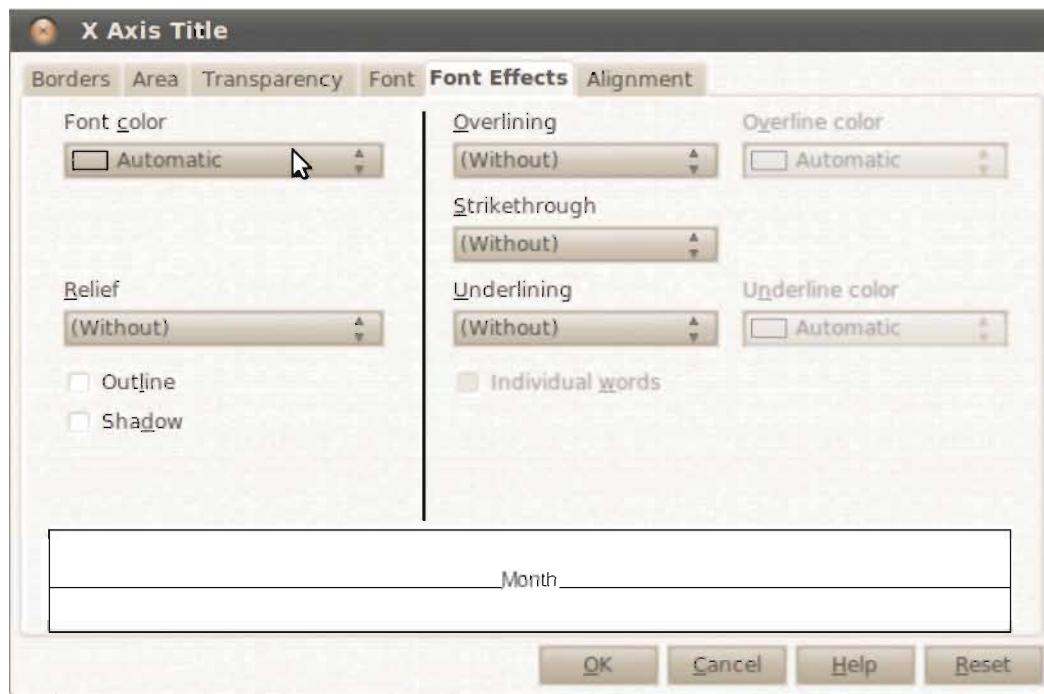


Figure 8.13 : Formatting X-Axis titles

- Press **Format Selection** button.
- You will see a dialog box called **X-Axis Titles**. This dialog box is shown in figure 8.13.
- Fill the necessary fields in the dialog box.

Using this facility of formatting, you can provide information about various attributes of the X-Axis titles such as borders of the area of title of the axis, title fonts, font effects and alignment.

This dialog box provides facility to set options for setting borders, filling the area of title and assign level of transparency to the filled area. As stated earlier, you can also format fonts and change font effects for values of the axis. The **Font** tab lets you select the appropriate fonts. If you have suitable fonts in your computer, you may display labels in regional language like Gujarati and Hindi also. You may include or exclude font's effects such as font colour, overline title fonts with different colour lines, strikethrough fonts, underline title fonts with different colour lines. You can also apply shadow and outline effect to the title fonts to look fancy and attractive. Further, you may choose text flow and text alignment (orientation) of labels by choosing the last tab of the dialog box named **Alignment** in the dialog box shown in figure 8.13. Similarly you can format the Y axis title also.

Formatting X-Axis

You can also format X-Axis using the chart formatting toolbar. Select appropriate element, here select **X-Axis**; and press **Format Selection** button. You will see a dialog box. Using this dialog box,

you may set options to set scale of the axis, which is generally automatic. Positioning of axis, that is where the axis crosses at start, end or other places with other axis. Along with the positioning of the axis, you can also mention where you want labels of the axis. You may set labels of the axis near it, outside the axis, starting of the axis or ending of the axis.

Many times labels are very descriptive, to manage the size of the chart, you may want to decrease their size or fonts. If you do not require labels at all, there is an option available called **Show labels**. You can tick mark and deselect the option, which make labels disappear for that axis. Similarly, you can format the Y-Axis also.

Formatting the Chart Wall

The chart wall can be modified by selecting options from the tool bar and selecting **Format Selection** button. By doing this a dialog box appears, which is similar to figure 8.11. The dialog box allows you to change border properties such as invisible or visible border, type of border, colour of border, thickness (weight) of the border, etc. Besides border properties, the dialog box also provides you an opportunity to change area colour and degree of transparency of the wall colour.

Formatting the Chart Floor

The chart floor can also be modified by selecting options from the chart formatting tool bar and selecting **Format Selection** button. You may select the chart floor formatting option by selecting the chart, right clicking on the chart floor and selecting the **Chart floor** option from the popup menu. By doing this, similar dialog box appears which allows you to change property of borders, area and degree of transparency of the floor of the chart.

Formatting Chart Legends

If you have totally forgotten entering the legend or skipped it at the time of building the chart, you may add chart legends by right clicking on chart and choosing option **Insert Legend**. Once the chart legends are visible, you can think for formatting them. The chart legends can be formatted using the chart toolbar shown in figure 8.11. As usual, when you opt for formatting chart legends from the chart formatting toolbar and select the Format Selection button, a dialog box appears. If you have noticed, the chart legends are also placed in a rectangular area with invisible borders. Using the dialog box, you can set borders of the legend area, format area, change degree of the transparency, etc. These options behave in a similar way as explained earlier. In addition to these options, other options such as font, font effects and position of the legend in the chart area are also provided by this dialog box. These options are required as the legend of chart also has some text besides some graphics. Further, we need to mention its location that is where we want to put the legend text. The default location is the right side of the chart; however, you can place the legend on top of the chart, bottom of the chart or left side of the chart. You can select any suitable fonts, its size and appearance such as bold, italics or underlined.

Formatting Data Series

To modify a series of data you may select the chart and right click on it. A vertical menu appears to you. Select format data series option and format it. Another method to do same is selecting the data series element from the chart formatting toolbar (as shown in figure 8.11), and select Format Selection button. This will offer an opportunity through a dialog box to set parameters for the data series.

There is another way to format data series as given below :

- When the chart is ready, click on any column representing the data series. If you have clicked on a single red colour column, you may notice that all red colour columns are activated. This represents a data series.
- See figure 8.14 showing data series for Test marks having red colour columns in selected manner.
- Formatting chart toolbar appears on the top of the sheet. See encircled area with red line. It shows **Data series 'Test Marks'**.
- Choose **Format Selection** button. You can see another rectangle dialog box along with the chart. See rectangular area with blue coloured outline.
- You can select appropriate tab, such as layout, area filling, transparency level, etc.
- In figure 8.14, you can see that we are changing the appearance of the area by changing the fill colour from red to yellow.

Similarly, you may format the other data series.

Inserting Data Labels

Another similar operation is to enter the data labels along with every data column in the chart. Suppose you want to enter the actual value along with the chart component such as columns, then just activate the data series and press right click. After that you can select **Insert data labels**. In figure 8.14 you can see that the data labels are already inserted. See encircled number with blue line in figure 8.14. Similarly you may format the other data series.

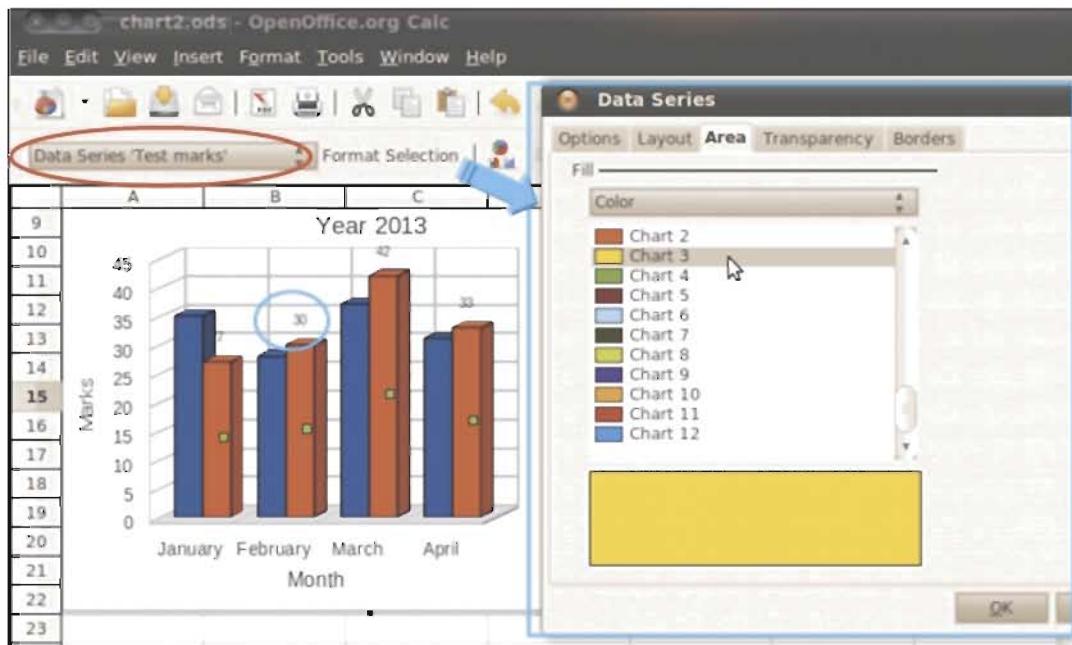


Figure 8.14 : Formatting data series

Working Example

Let us create one more chart of different type. Consider the data as follows. Let us consider a data about movies. You all might like to enjoy movies. However, everybody has got different choice and liking. Some of you like action movies and some like comedy movies. There is a special audience for science fiction films. Consider data given in table 8.2 representing the similar situation.

Movie type	Action	Comedy	Drama	Science-Fiction
Value in %	30	35	20	15

Table 8.2 : Liking of movies

Let us draw a chart using the information shown in the table 8.2. Perform following steps.

- Enter data shown in the table 8.2 into a Calc worksheet and save it by giving appropriate name.
- Check that the data range should be A1 to E2.
- Select **Insert → Chart**.
- Select the chart type as **Pie** chart;
- Choose **3D** look and **Simple** chart.
- Press **Finish** button.
- You can see the chart as shown in figure 8.15.
- However, you cannot see the data labels in your chart as well as the Drama category is also not pulled out. To add the data labels and to pull out the Drama category (represented by yellow colour), perform remaining steps as given below.
- Select the chart. You may see the grey border surrounding the chart.
- Right click the chart, which causes a vertical menu appears to you.
- Select **Insert Data Labels**.

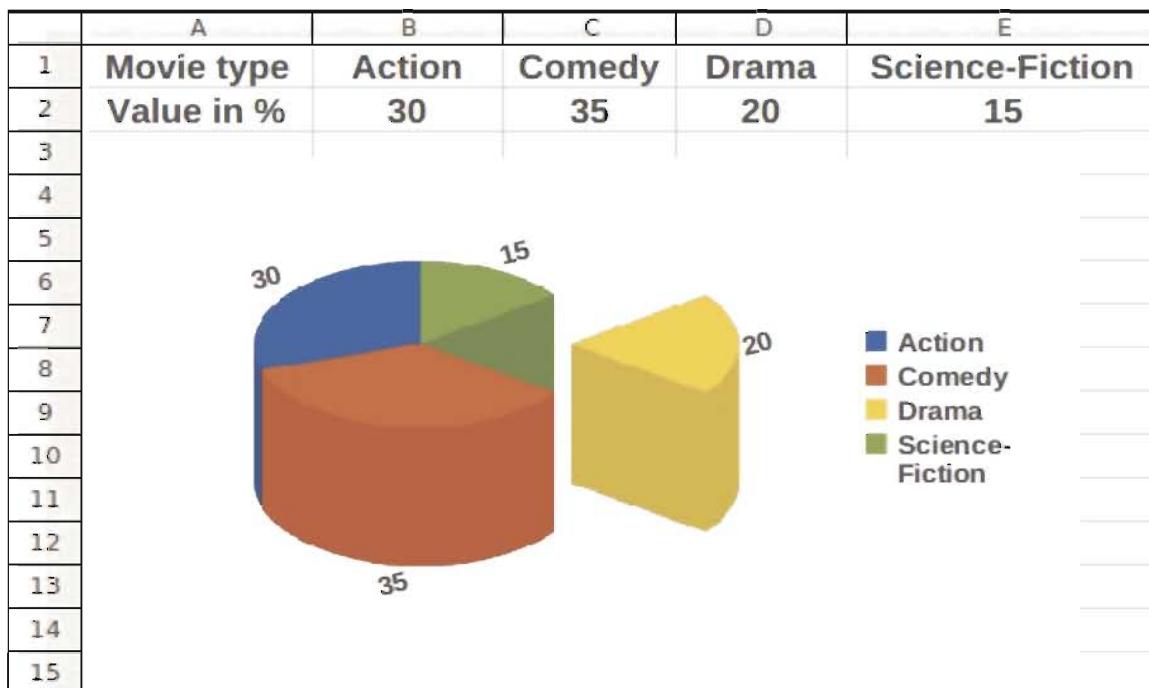


Figure 8.15 : Pie chart

- You can see the values are displayed along with the chart.
- To pull out a portion of the pie chart, just select the area using mouse key. Using the mouse, drag the area at the desired position.

- If you want to do further formatting, select the chart. The chart formatting tool bar appears to you. Select appropriate element to be formatted and choose **Format Selection**. Using this facility, you can change chart type, chart area, data series, font, font effects, appearance of axis, legends, grids, chart floor and wall, etc. Here we can make the fonts bold and increase the font size. We also have rotated the data labels with 15 degree.
- Similarly, you can format the legends of the chart and make the fonts of the legend bigger.

Adding Hyperlink to the Chart

Hyper link is a reference to data that can be followed by selecting it. As you know, a hyperlink may points to a whole document or part of a document. The text which embeds such links within it is called a hypertext. Once a chart is created, you can set hyperlink to the whole chart or to a component of the chart. If user clicks on the hyperlink, it will lead the user to predefined destination. You may set hyperlink to a document of the Web, document within your local area network/computer or to a new document.

For any type of hyperlink, you must have a chart. You have to create a chart first. Follow the below given steps to create hyperlink to the Web (Internet) document.

- Enter data and create a chart of suitable type.
- Verify the chart and select it using mouse keys.
- Select **Insert → Hyperlink**,
- It will offer you four options regarding the destination of the link. That is, you can hyperlink the chart (or selected entity) to the Internet, Mail & News, Document or New Document. Figure 8.16 shows these four ways (encircled with red oval) for linking the chart.

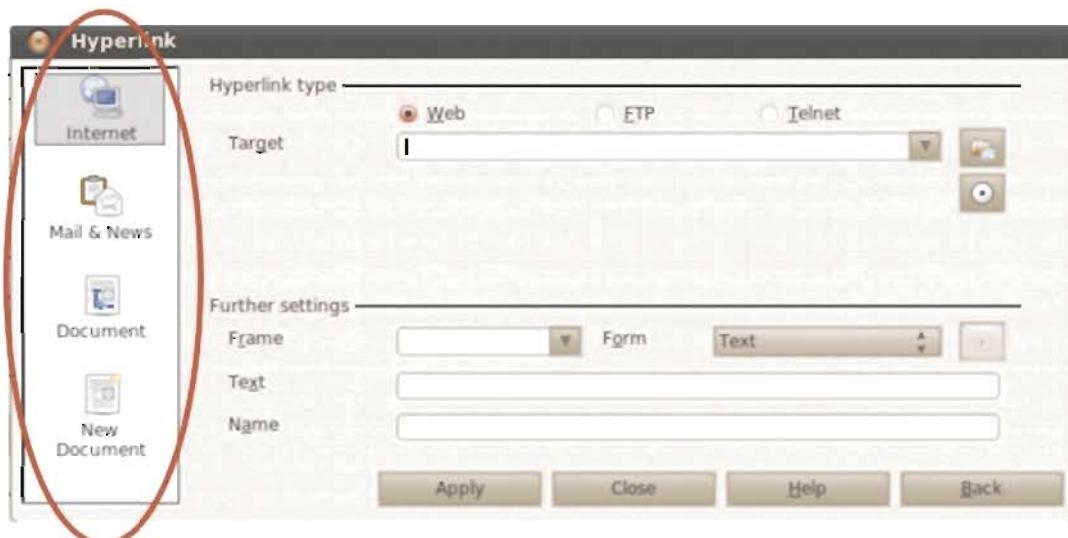


Figure 8.16 : Hyperlink options

By default, the first option "**Internet**" is highlighted and as a **Target** you may give the destination URL such as *Google.com*. You are supposed to have an Internet connection, if you want to experiment the link. If everything goes right, the chart will lead you to a Google's main page when clicked. Alternatively, any type of hyperlink is followed by pressing **Ctrl + Enter**.

Once you enter all necessary information, you may choose **Apply**. Use **Close** button to close the dialog box.

Linking a Chart to a Mail

If you want to connect the chart to the mail and news, you need to select Mail & News option. You will see different dialog box. Figure 8.17 shows the options used when linking to Mail & News.



Figure 8.17 : Options for mail and news in hyperlink

You need to provide information about the receiver and subject of the mail. The chart will be queued for mailing to the receiver's address when you select the **Apply** option. However, this process remains offline and requires a mail client set at background. Similarly, for **News** option, you need to configure news server. Choose **Close** button at the end.

Linking a Chart to a Document

To link your chart to a document, you have to choose the third option Document. You may see screen as shown in figure 8.18.

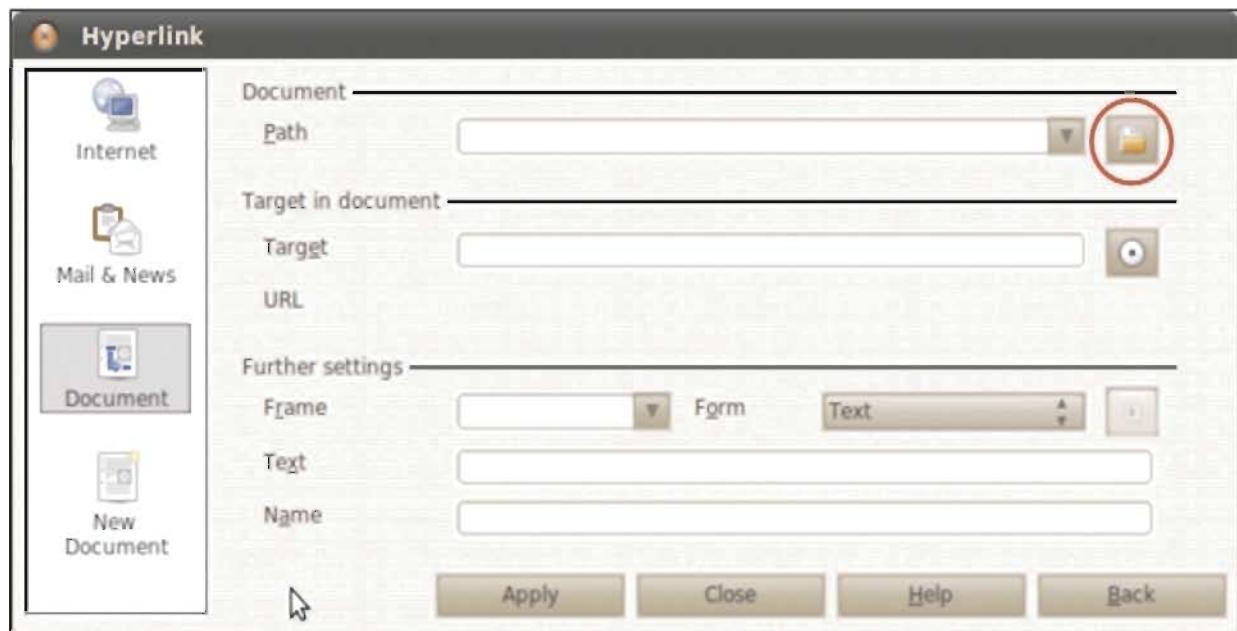


Figure 8.18 : Linking chart to a document

This option is useful when you have a small chart prepared, which is further explained in detail in another document. In this situation, for further explanation, you may link your chart to other document. Any way, if you think that a particular document is related to the chart, you may link it with the chart using this facility. The full path mentions the location of the destination document. You can browse the document, if you do not remember the complete path of the document using the Browse button (encircled with red line) given beside the path text box. You can also provide target within the destination document using the **Target** option.

At last choose **Apply** option. At the end use the **Close** button to complete the procedure and close the dialog box.

Linking a Chart to a New Document

To link the selected chart to a new document, you can choose the fourth option **New Document**. You can provide **File name** and **File type**. Spreadsheet, word processor, drawing etc. type of files you may create using this option. The file once created, is now available to edit, as **Edit now** option is selected by default. If you do not want to edit it, you may choose **Edit later** option. Figure 8.19 shows the above mentioned scenario.

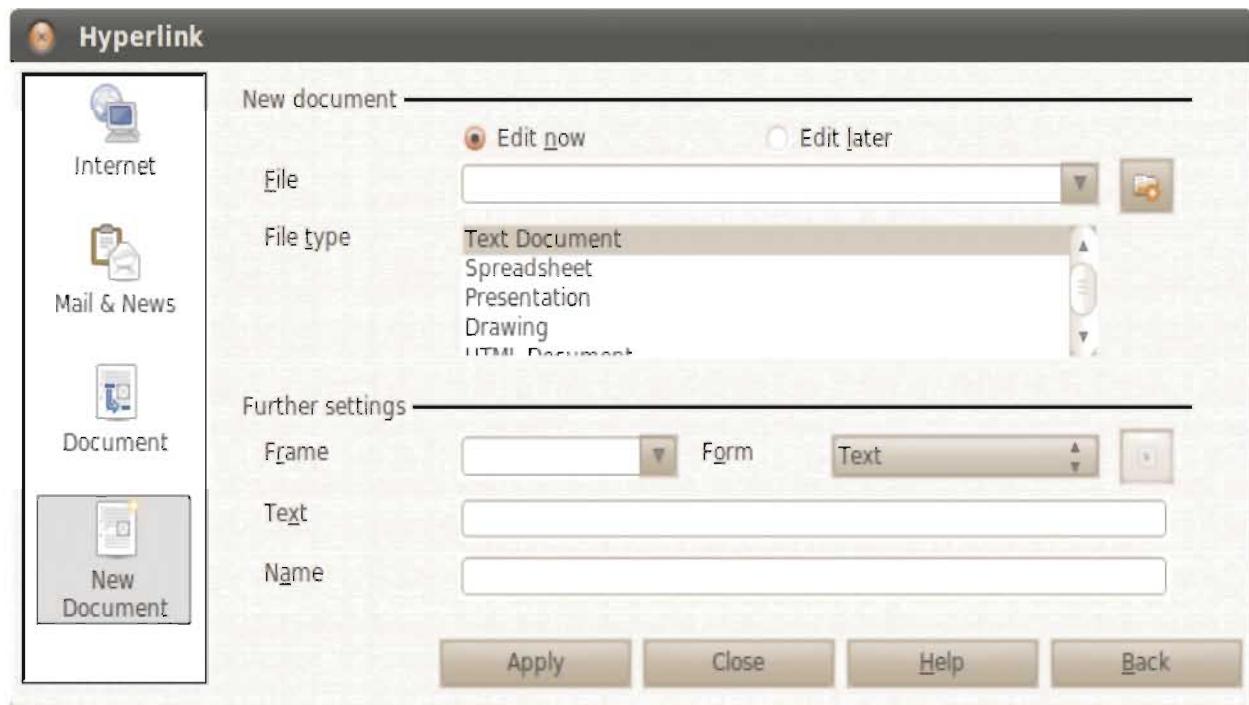


Figure 8.19 : Linking a Chart to a New Document

At the end, choose **Apply** option. Use **Close** button to close the dialog box.

Exporting the Chart and other Operations

You can export your document containing chart as portable document file having PDF extension. To do so, perform following steps:

- Select **File → Export**. You may see the screen as shown in figure 8.20.

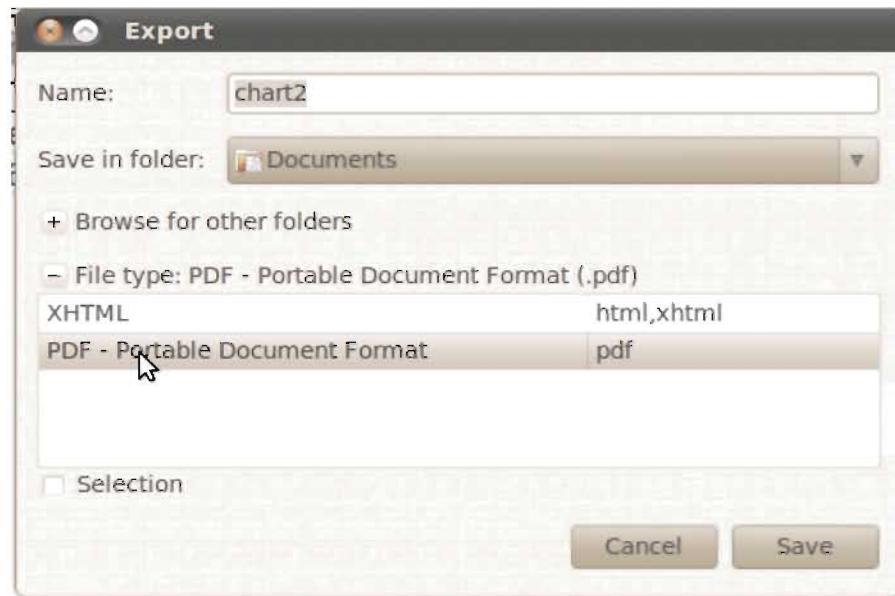


Figure 8.20 : Exporting the document

You will see two options such as export the file as XHTML (web page) or PDF (Portable Document Format).

- Choose **PDF**.
- Save the file in a desired folder.

When you give your choice as PDF; you will see a dialog box asking various parameters about the resulting PDF file as shown in figure 8.21. Using the dialog box shown in figure 8.21; options such as number of pages, format of images, (if it contains any) etc. can be set. If you have selected a range of pages that you wanted to export; content of only the given number of pages will be exported into the resulting PDF document. The default range is all the pages of the document.

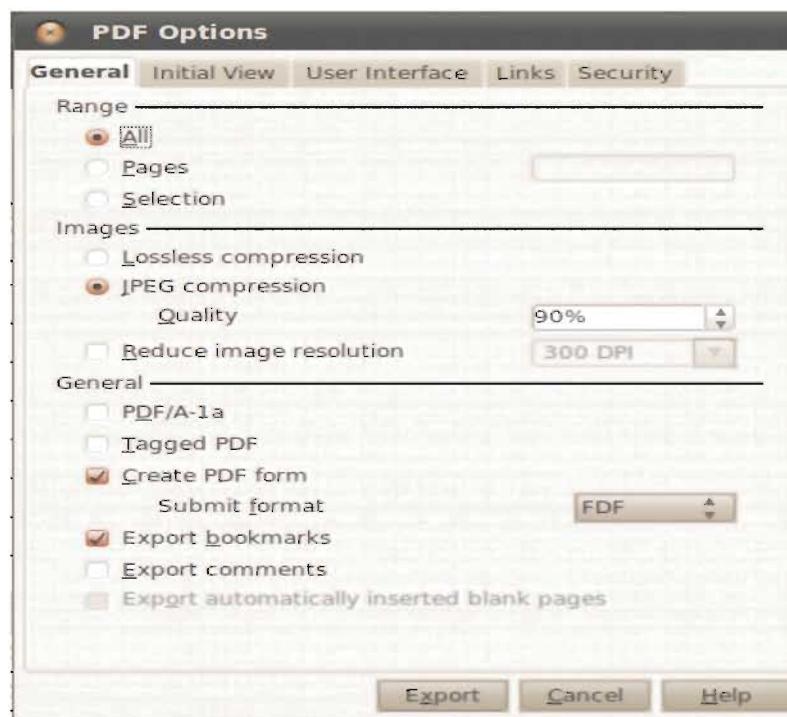


Figure 8.21 : Exporting document in PDF form

Further, the images of the documents are saved in jpeg format. Here you have options of lossless image compression as well as JPEG image compression with given percentage of quality. Select the Create PDF option, if it is not selected. Generally, these options are default options and automatically selected by the system. You may not disturb the default general option. You may choose Export button directly. By doing so, your document with the chart is exported in PDF form.

Copying the Chart

For copying chart to writer, impress and other docs you have to select chart, copy it and paste it as and when require. Perform following simple steps to copy the chart.

- Choose the chart with mouse keys.
- Right click and choose **Copy** or **Cut** option.
- You may go to **Edit** menu and choose required options also.
- Go to the destination document. And use **Paste** command.
- Save the destination file.

Deleting the Chart

To delete the chart, simply select the chart and press Delete button.

Printing the Chart

Many times you may require printing a chart and the worksheet content. Perform following simple steps to print the chart.

- Choose **File → Print**. You will see dialog box as shown in figure 8.22.
- Choose required printer.
- Click on OK button.

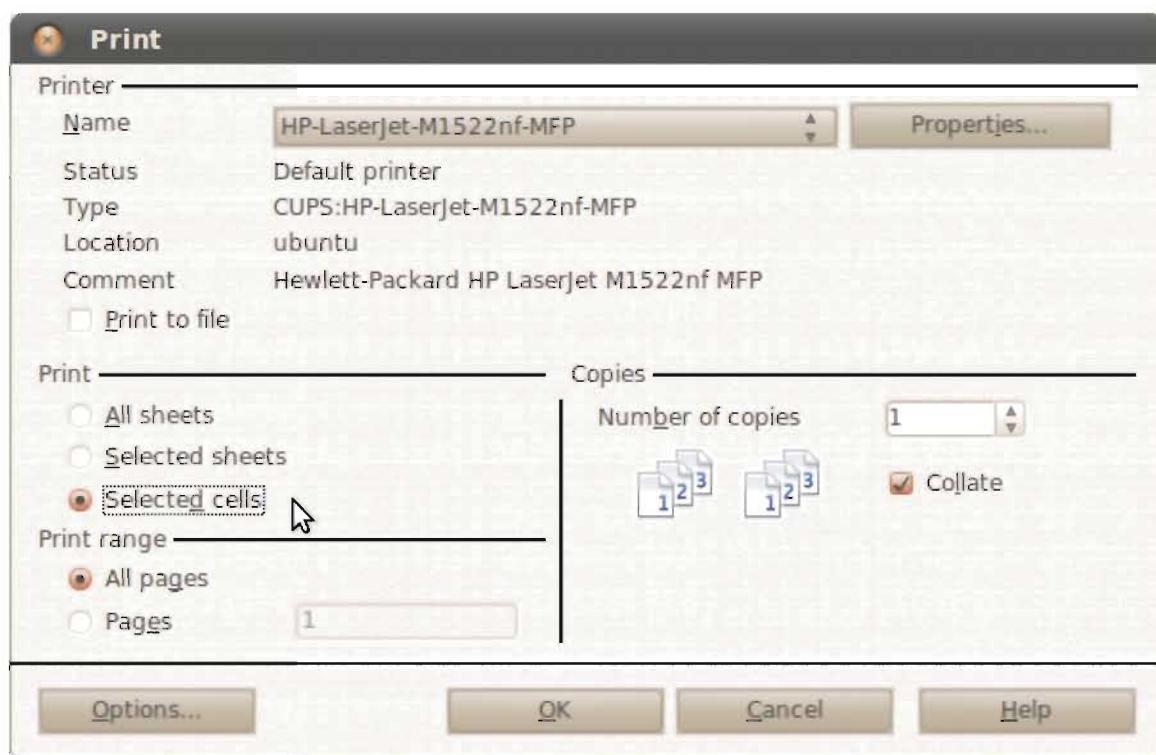


Figure 8.22 : Printing a document

Note that it is important to check the page preview before you print the document. This will give you a clear picture of how your document is going to be printed.

Inserting Other Multimedia Objects

Besides the charts, you can insert any multimedia object into a worksheet. The multimedia objects include objects of multiple media such as pictures and camera clips, video and movie clips, animated clips such as GIF files and 3D texts. For this, we normally use simple Insert menu commands. That is, to insert a camera clip or clip from your mobile, first you need to connect the device with your computer. You may use wired connection (if you have cord with your mobile or camera with USB connector) or Bluetooth technology. Once your computer has access of the object such as your photograph; select **Insert → Picture** to insert the desired picture. Similarly, you can insert a movie and an animated clip (or GIF file) into the Calc document.

You can add a 3D text in a worksheet also. The 3D text in Calc is known as graphical text art or Fontwork. An example is provided in figure 8.23.



Figure 8.23 : Fontwork in Calc

To create a Fontwork object follow the steps given below.

- On the Drawing toolbar or on the **Fontwork** toolbar, click the **Fontwork Gallery** icon.
- In case, the Drawing toolbar or the **Fontwork** toolbar is not visible choose **View** menu and choose **Toolbars** option to enable the toolbar (**View → Toolbars**).
- In the **Fontwork Gallery** dialog, select a Fontwork style and click OK. You can use style from **Fontwork Gallery**, you can edit shape from Fontwork **Shape**, you can change height of letters, alignment of text, and character spacing of the text. You may use drawing toolbar to edit some more properties such as line colour, fill colour and fill style.
- The Fontwork object is inserted into your document. Double-click the object to enter your own text.
- Press Esc to exit text edit mode.

Once the Fontwork object is inserted; you can edit it by clicking it. The Fontwork toolbar is displayed. Edit the text. At the end, press Esc key when you finished the editing.

Summary

In this chapter we learnt about various charts, their importance and their purposes. We have also seen how charts are inserted and modified in the Calc spreadsheet package. There are several components of a chart. This chapter considers most common components of a chart and explains it in detail. We have also learnt how to refine and format such chart components for more effective presentation. The chapter has discussed two examples of chart namely: a column chart and a pie chart. We have also seen how a hyperlink can be set to a chart. The chapter also discussed mechanisms to print chart, to copy a chart, to delete a chart and to export a chart as portable document (PDF). At last, we have also seen how to insert a 3D and multi- media object in a spreadsheet document.

EXERCISE

- 1.** Explain advantages of drawing a chart in Calc.
 - 2.** Explain various types of charts with one line description of each.
 - 3.** Explain how a chart can be inserted in Calc.
 - 4.** List the components of a chart wizard.
 - 5.** List elements of a chart with one line description of each.
 - 6.** How can we change the chart type once the chart is created ?
 - 7.** How to format chart axis?
 - 8.** Can we add hyperlink to the chart? How ?
 - 9.** Choose the correct option from the following :
 - (1)** Which of the following option is used to print a chart ?

(a) Insert → Chart	(b) File → View
(c) File → Print	(d) View → Chart
 - (2)** The charts in Calc may have how many of the following axis ?

(a) Two	(b) Three	(c) Two or three	(d) Four
---------	-----------	------------------	----------
 - (3)** Which of the following is used to enter 3D text in Calc ?

(a) Fontwork	(b) Artwork
(c) Drawing work	(d) Graphwork
 - (4)** Which of the following is used to provide a link to the chart ?

(a) Activelink	(b) Hyperlink
(c) Drawinglink	(d) Connectionlink
 - (5)** Which of the following shows us the preview of the chart ?

(a) Save as XHTML	(b) Page preview
(c) Export chart	(d) Any of these
 - (6)** To fix the column or row in a data range - to make it absolute, which symbol is used ?

(a) #	(b) \$	(c) &	(d) %
-------	--------	-------	-------
 - (7)** To which of the entity can a Calc chart be linked ?

(a) To an existing document	(b) To a new document
(c) Web page	(d) All of these

LABORATORY EXERCISE

- 1.** Find out your own marks in main subjects such as English, Science, Maths, etc. Also collect class highest marks in respective subjects. Draw a column chart comparing your marks with the highest marks in every subject.
- 2.** Make a small survey in your neighbourhood about who is reading which newspaper. Collect data, arrange them in a clean form and prepare 3D pie chart based on this.
- 3.** Consider a typical mathematical function such as $Y = 2 \times X$. Consider values of the X as given in the table given below.

Value of X	Value of Y
1	?
3	?
5	?
8	?
6	?

Do the followings :

- (a) Enter these values in a Calc worksheet.
 - (b) Use formula to find out values of Y for each given value of X.
 - (c) Plot a line chart of the X and Y values together.
 - (d) Format chart as you like.
- 4.** Consider following data related to household budget.

Category	July
Provision	1000
Milk	500
Vegetables	500
Servant	800
Electricity	1500
Gas	500
Fees	1500
Newspaper	100
Cable	200
Mobile	200

Do the followings :

- (a) Enter these data into a spreadsheet and develop a pie chart.
- (b) Once chart is ready format it as per your choice; e.g. change colour of any component, change font style, add/remove legends, add data labels to the chart and pull out a few segments.
- (c) Export the chart as XHTML file. Open a web browser and verify how it looks.



Problem and Problem Solving

People in their day to day life solve lot of problems. Let us discuss one such process, consider that you are playing 'Hide and seek' with your friends. In this game our problem is to find the location where our friend has hidden. This can only be done only if we can predict the place of his/her hiding.

Let us take another problem, assume that we have to solve a mathematical equation to find the value of x , if we have been given equation $2x + 4 = 0$. To solve this problem we need to rearrange the equation as $2x = -4$, and then say that $x = -4/2$, the answer here is -2 .

Let us take another example. We want to find the meaning of the word "Eloquent". We will need an English Language dictionary for the same. A dictionary contains thousands of words and its meaning. We are still able to find the desired meaning very quickly. Have you ever thought how we are able to do it so fast? We actually take advantage of the sequential order of words given in dictionary. We quickly eliminate the options that do not start with alphabet "E" and start looking for the words that start with alphabet "E"; we again use this elimination method to find words with second alphabet as "l". This elimination method is continued till we find the exact word that we are looking for.

The above example gave us a hint of what a problem can be? Problem solving can be a mental or machine (any systemic procedure) process that leads to the desired outcome required by the user. The examples discussed above also indicate that result of some problems is exactly predictable while for others predicting exact outcome may become difficult. Thus the problems can be classified into two types; well defined and ill-defined problems. In the above cases problem 2 and 3 are well defined problems, while problem number 1 is ill-defined. Note that well defined problems have clear goals and hence we can clearly define solution steps. It is fact that computers solve well defined problems only and therefore we will discuss only well defined problems in this chapter.

In the field of computers, solution of a given problem is a sequence of instructions given to a computer. Computer can solve variety of problems from the easiest to the most complex ones. To solve a problem it needs to be given a complete set of instructions. These instructions tell the computer what is to be done at every step. Remember one thing, computer does not solve a problem; it merely assists in solving the problem. We can solve any problem using the steps mentioned:

1. Define the problem.
2. Identify the input, output and constraint of the problem.
3. Identify different alternatives of solving the problem.
4. Choose the best possible alternative from the above list.
5. Prepare detailed stepwise instruction set of the identified alternative.
6. Compute results using this instruction set.
7. Check correctness of the answer obtained.

Steps 1 to 5 are performed by person who needs the solution, while step 6 and 7 are performed by a computer.

Assume that we need to find whether a given number is odd or even. The following set of instructions can be used to solve this problem.

1. Accept the number
2. Divide the number by 2 and find the remainder
3. If the remainder is 1, the given number is odd otherwise the number is even

The generalized solution to the problem is obtained using three different techniques mentioned below.

1. Pseudo code
2. Flowchart
3. Algorithm

The three steps given for finding whether a number is odd or even is known as Pseudo code. Pseudo means fake or simulated, we should consider the second meaning as it is more appropriate in our context. We can say that the steps mentioned to solve the problem is simulated code. Let us now discuss the other two techniques.

Flowchart

A flowchart is a technique in which we use pictorial representation of every action that we perform within the machine process that solves a problem. A set of symbols, showing different actions, is used to represent a flowchart. The symbols are also called components of flowcharts. We have a unique symbol corresponding to each action within a process. Let us discuss some commonly used components and its associated symbols.

Start and End : The Start and End components are used to show the beginning and the end of a flowchart. It is represented by an oval shape as shown in figure 9.1. The symbol is also called terminal symbol.

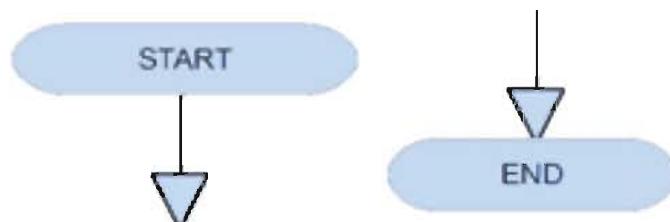


Figure 9.1 : Terminal Symbol

This symbol is used twice in a flowchart as it is used in the beginning as well as at the end of a flowchart.

Input-Output : Every problem needs an input, it then processes this input and generates an output. Consider the problem number 3 discussed above, here we need to input one number, dividing this number by 2 and finding the remainder is the process and the decision whether the given number is odd or even is our output. Thus we need one symbol to show the input and one to show the output. The input and output in flowchart is represented by a parallelogram as shown in figure 9.2.

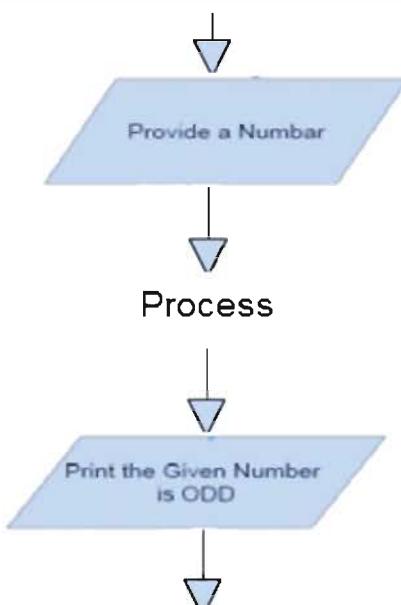


Figure 9.2 : Input-Output Symbol

Arrow : Observe that in both figure 9.1 and 9.2 we have shown arrows coming in and going out of the symbols. An arrow is used to show the sequence of the actions that are to be performed. It generally starts at one symbol and ends at another symbol. Thus there will be one arrow coming into the symbol and one arrow going out from the symbol.

Note that the start symbol will only have arrow going out, while the end symbol will have arrows coming in.

Process : Process is the core part of any solution procedure. A process is actually, a sequence of actions. To represent a process we use rectangle symbol as shown in figure 9.3.

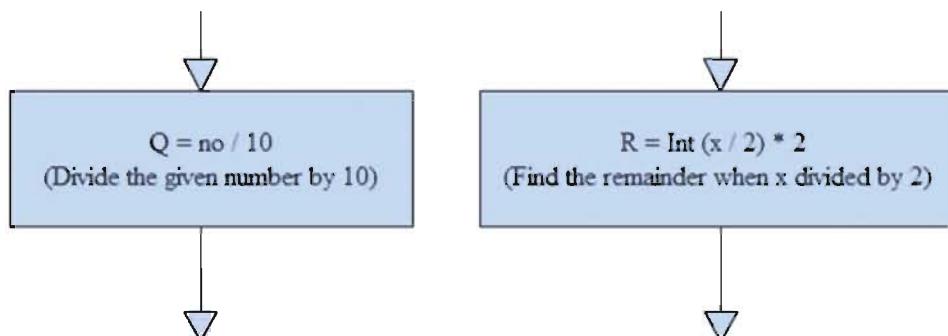


Figure 9.3 : Process Symbol

Normally a process in computers is either arithmetic or logical operation. The arithmetic operation we mean addition, subtraction, multiplication or division. Logical operations generally help in decision making, when used it answers in the form of true or false. For example we may ask a question such as is 10 greater than 5? The answer to this question is yes or true.

Decision : A logical decision of a process is represented by a diamond shaped symbol as shown in figure 9.4. It is also called a test symbol. A decision box is used when we want to alter the normal sequence of the solution (see in figure 9.4) or when a specific statement needs to be executed based on the result of decision.(See figure 9.5).

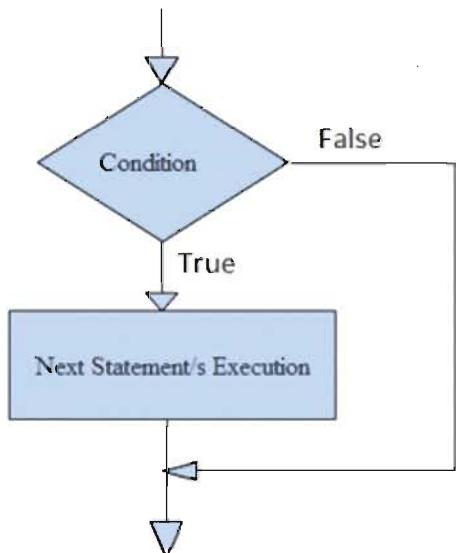


Figure 9.4 : Decision Symbol

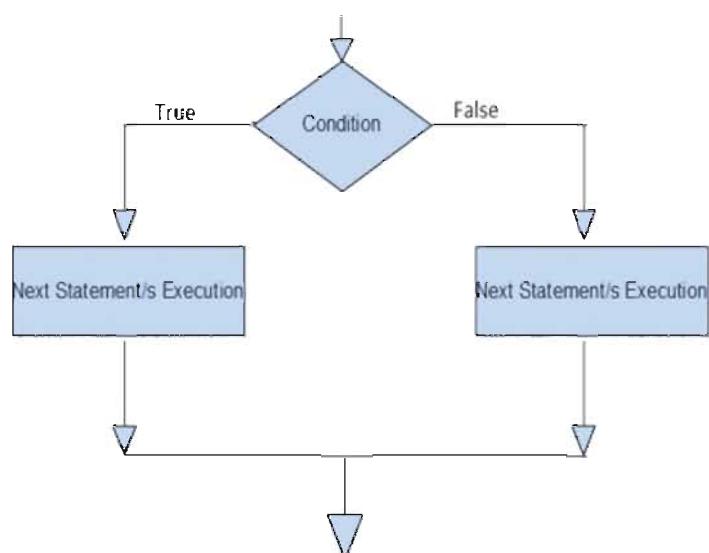


Figure 9.5 : Execution of different results

Sometimes we may require more than two alternatives to take a decision. In such a situation more than one decision box can be combined to form the required alternatives as shown in figure 9.6.

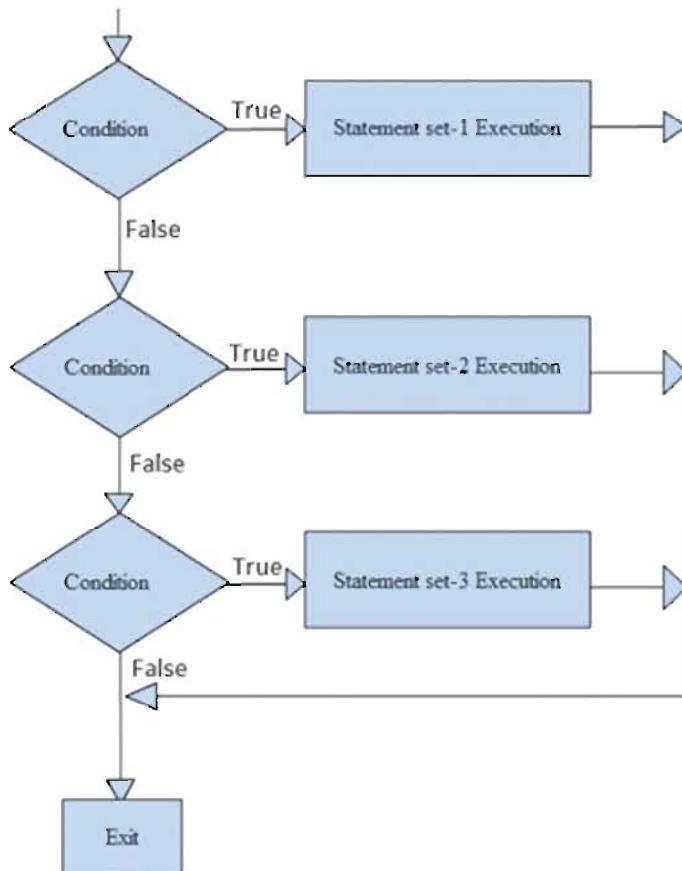
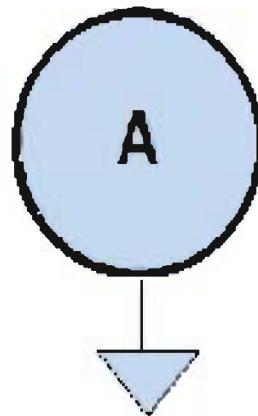
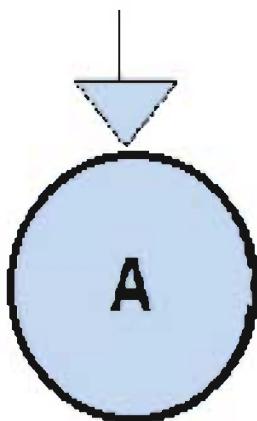


Figure 9.6 : Combining more than one decisions

As shown in the figure 9.6 first the condition will be checked; if it results into true, then first set of statements will be executed. In case of result being false next condition will be checked. This process will be repeated for all the given conditions. Note that when any condition results into true, the conditions following it will not be checked.

Connector: A circle symbol is used to represent a connector. At times it is possible that a flowchart becomes too large to fit into a single page or it may not be possible to use an arrow to link two processes. In such cases a connector can be used to join the two parts. Minimum two circles are required for a connection; one with an arrow coming into a circle and other with an arrow going out of the circle as shown in figure 9.7. To distinguish different connectors, different characters are used with a same character (alphabet) in a pair.

Break From One Flowchart



Joining Another Flowchart

Figure 9.7 : Connector

Examples of Flowchart

Till now we have learnt what a flowchart is and also looked at some of the symbols used in it. Let us now solve some problems by creating flowcharts.

Problem 1 : The fitting charge of tiles is Rs. 10 per square feet. Now assume that the problem is to find the total cost of fitting tiles on the rectangular shaped town hall floor.

Solution :

To find the total cost of fitting the tiles we have to first need to find the area of the floor. Area of a rectangle, as we know, is product of length and breadth. Hence if we know the length and breadth we can find the area of a rectangle. The formula to compute area of the floor can be given as $\text{Area} = \text{Length} * \text{Breadth}$.

Next to find the total cost of fitting the tiles we multiple cost of fitting per square fit of tiles with the area of the floor. The formula to compute total cost thus is $\text{Cost} = \text{Area} * 10$.

To solve this problem we require four variables Area, Length, Breadth and Cost. Before we continue further let us first define the term variable as it will be used in all flowcharts.

Variable : An entity whose value can be assigned and changed during the execution of the process.

The flowchart to find cost of fitting the tiles on the floor is shown in figure 9.8.

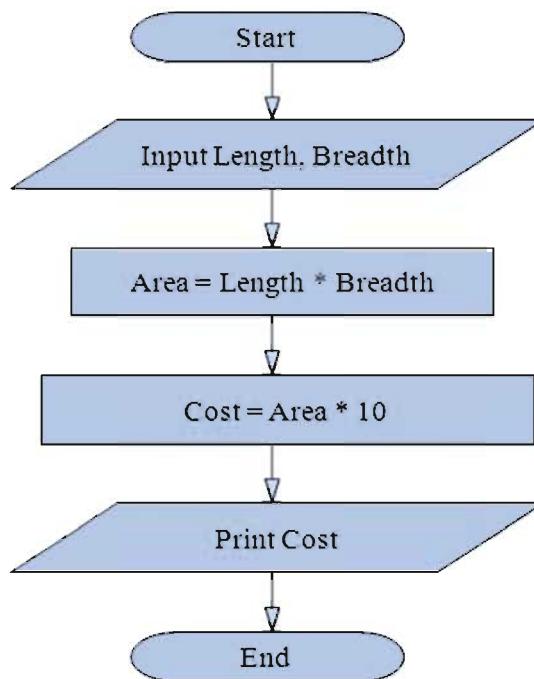


Figure 9.8 : Flowchart to find cost of tiles

Observe the value 10 in expression $\text{Cost} = \text{Area} * 10$. Here 10 is known as a constant. It is an entity whose value once assigned remains fixed and cannot be changed during the entire process. The solution that we have created in flowchart shown in figure 9.8 can only be used in the cases where the cost of fitting one square feet tile is Rs. 10. If the cost of fitting the tile changes, the above solution will not work. A generalized solution the above problem is given in figure 9.9.

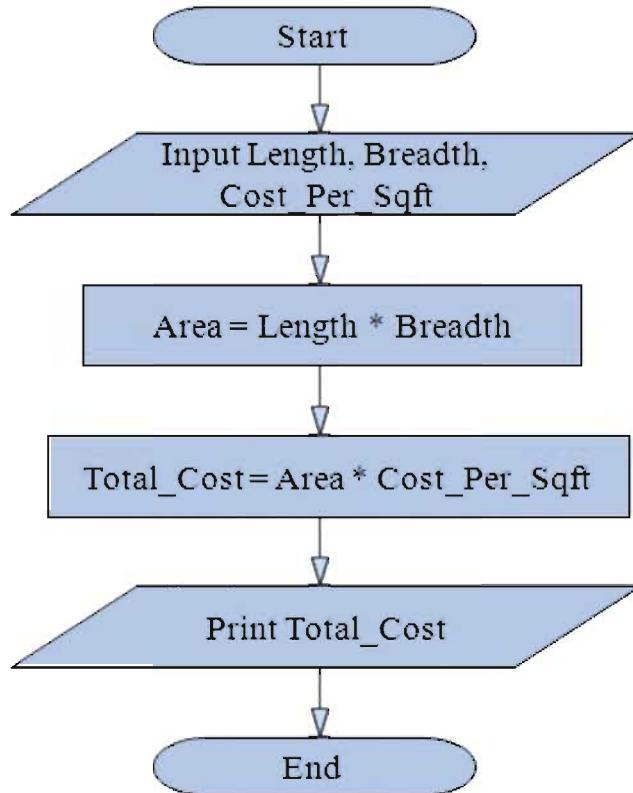


Figure 9.9 : Modified flowchart to find cost of fitting tiles

Observe the flowchart in figure 9.9; here we have used five variables instead of three. The variable Cost_Per_Sqft is now used to store the value of cost for fitting one square feet of tiles. With every run now we may change the value, and we will now be able to get the output. Table 9.1 lists some such outputs.

Length	Breadth	Area=Length*Breadth	Cost_Per_Sqft	Total_Cost=Area*Cost_Per_Sqft
50	50	2500	10	25000
25	75	1875	10	18750
45	35	1575	20	31500

Table 9.1 : Output of flowchart shown in figure 9.9

Problem 2 :

Suppose you have a very nice and round cricket ground at the city sports center. The authority wants to make fencing surrounding this ground. They also want to cover the total surface with a lawn. The authority wants to know how many meters the total fencing will run into? They also want to know the total surface area of the ground that needs to be covered with lawn.

Solution :

To solve this problem we need to find out the area and perimeter of the ground. The formula to find area of the circular ground is $\text{Area} = \pi * R^2$, similarly $\text{Perimeter} = 2 * \pi * R$ (here value of π is 3.14 and R refers to radius of the ground).

Thus to solve this problem we need three variable Radius, Area and Perimeter. The formula that we will use in the flowchart to calculate area and perimeter are given below:

$$\text{Area} = 3.14 * \text{Radius} * \text{Radius}, \text{Perimeter} = 2 * 3.14 * \text{Radius}$$

The flowchart for the said problem is shown in figure 9.10.

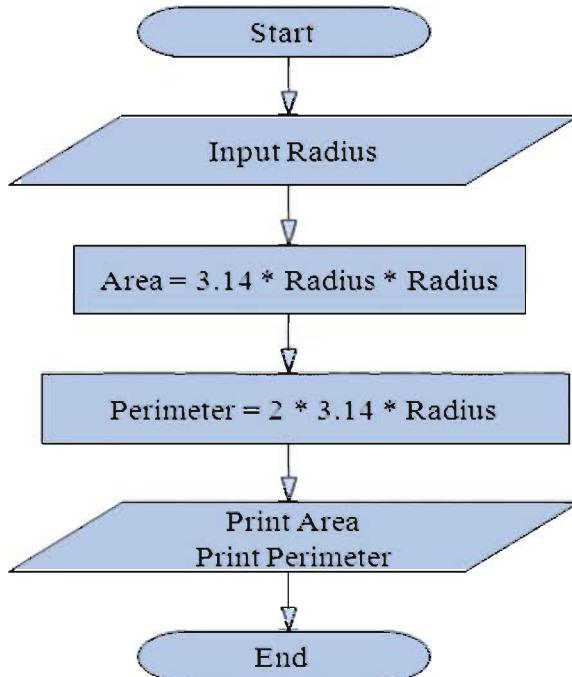


Figure 9.10 : Flowchart to find area and perimeter of a circle

Problem 3 : Ms. Jhanavi has borrowed a loan of Rs. 35,000 at the rate of 11.5 percent from a bank for 6 years. Calculate the simple interest that she will have to pay to the bank.

Solution :

The formula to compute simple interest 'I' on a loan of principle amount 'P' at an interest rate of 'R' for 'N' years is given as $I = (P * R * N) / 100$.

To solve this problem we require four variables I, P, R, N and a constant value 100. Figure 9.11 shows the flowchart to solve the problem.

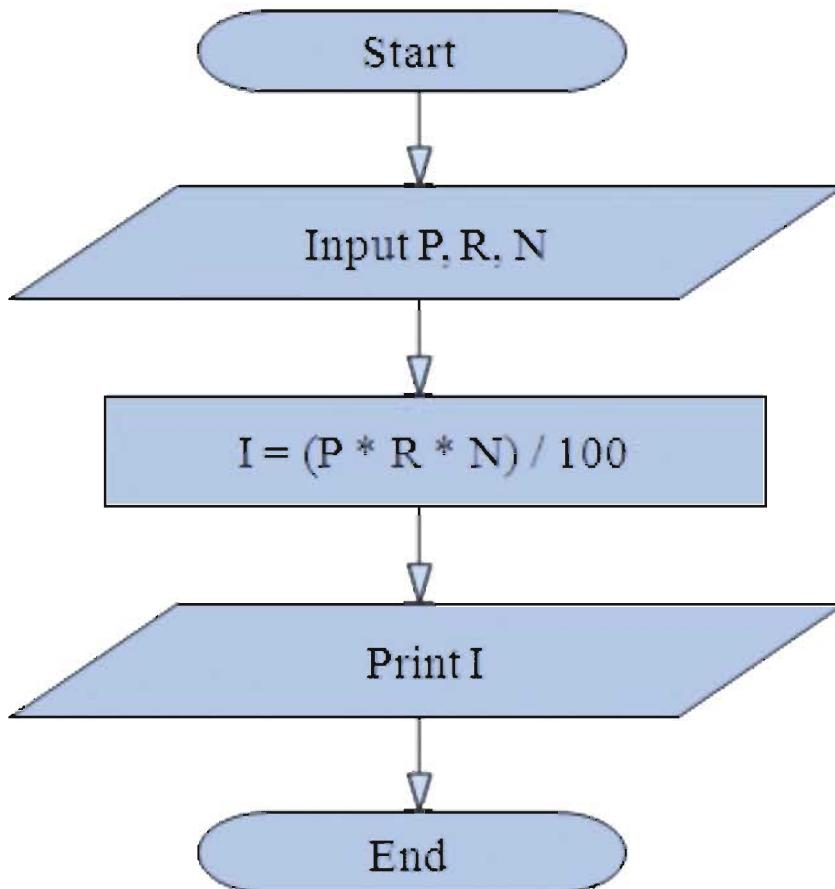


Figure 9.11 : Flowchart to find simple interest

Problem 4 : Assume that you want to find out the youngest student amongst two students. The input in this case will be the age of the student.

Solution :

The solution to the problem can be obtained by performing the steps mentioned below :

Take the ages of both the students; let it be assigned to variables Age_1 and Age_2. Now compare the value of Age_1 with Age_2. If both values are same, then we have two students with same age, hence both can be considered as youngest. If the value of Age_1 is less than Age_2 then first student is youngest. Otherwise second student is youngest. Figure 9.12 shows the flowchart to solve this problem.

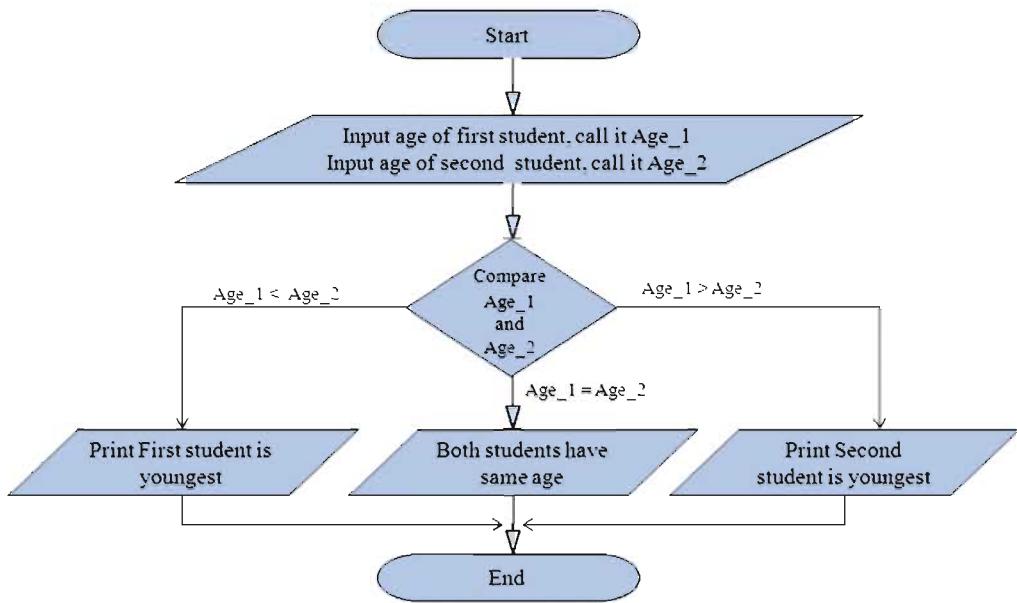


Figure 9.12 : Flowchart to find youngest of two students

Problem 5 : Let us now try to find youngest amongst three students.

Solution :

Take the ages of three students; let it be assigned to variables Age_1, Age_2 and Age_3. Now compare the value of Age_1, Age_2 and Age_3. If value of all the ages is same then, we have three equal aged students. Hence we can assign all three as youngest.

Otherwise compare Age_1 and Age_2. If the value of Age_1 is less than Age_2; then compare Age_1 and Age_3 if still Age_1 is less than Age_3, Then first student is youngest.

In the comparison made above, if Age_2 is less than Age_1, then compare Age_2 and Age_3 if still Age_2 is less than Age_3, Then second student is youngest. Otherwise third student is youngest. Figure 9.13 shows the flowchart to solve this problem.

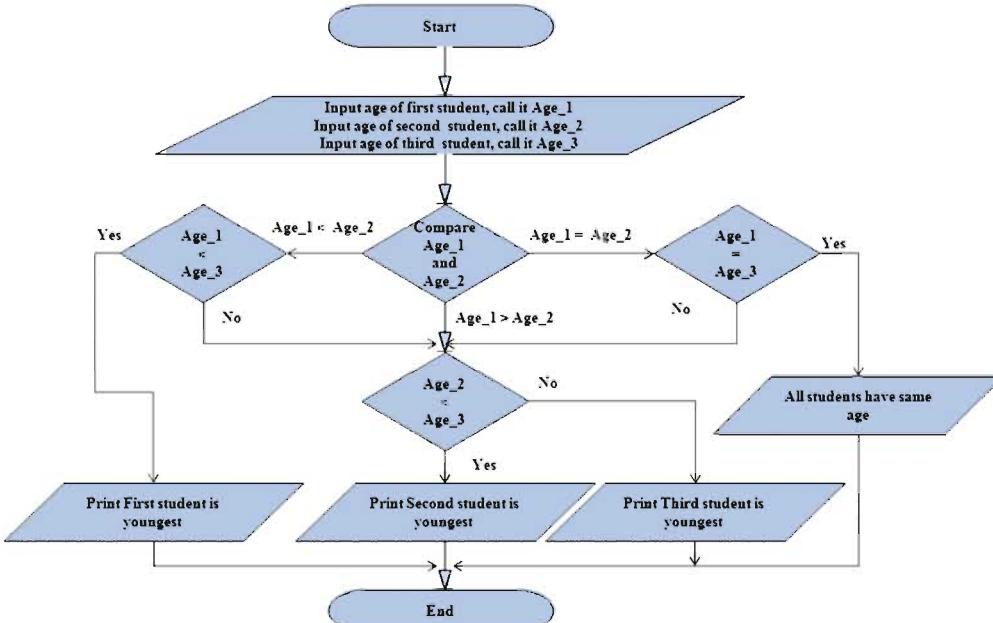


Figure 9.13 : Flowchart to find youngest of three students

Problem 6 : Let us further modify problem 5 to find the out youngest student from a class with any number of students in it. The steps mentioned below can be used to solve our problem.

Step 1: Choose first student, whatever his age, assume that it is MINIMUM.

Step 2 : Now choose another student and compare his/her age with the one that has been mentioned as having minimum age.

Step 3: If new student chosen is younger, we now make his/her age as MINIMUM and discard the old student's age.

Step 4: If both the students have same age then we discard the new student's age and assume that old student's age is still MINIMUM.

Step 5: We repeat step 3 and step 4 till we get the student that has youngest age. When we repeat the same set of statements for more than one time, it is called loop.

Figure 9.14 shows the flowchart to solve this problem.

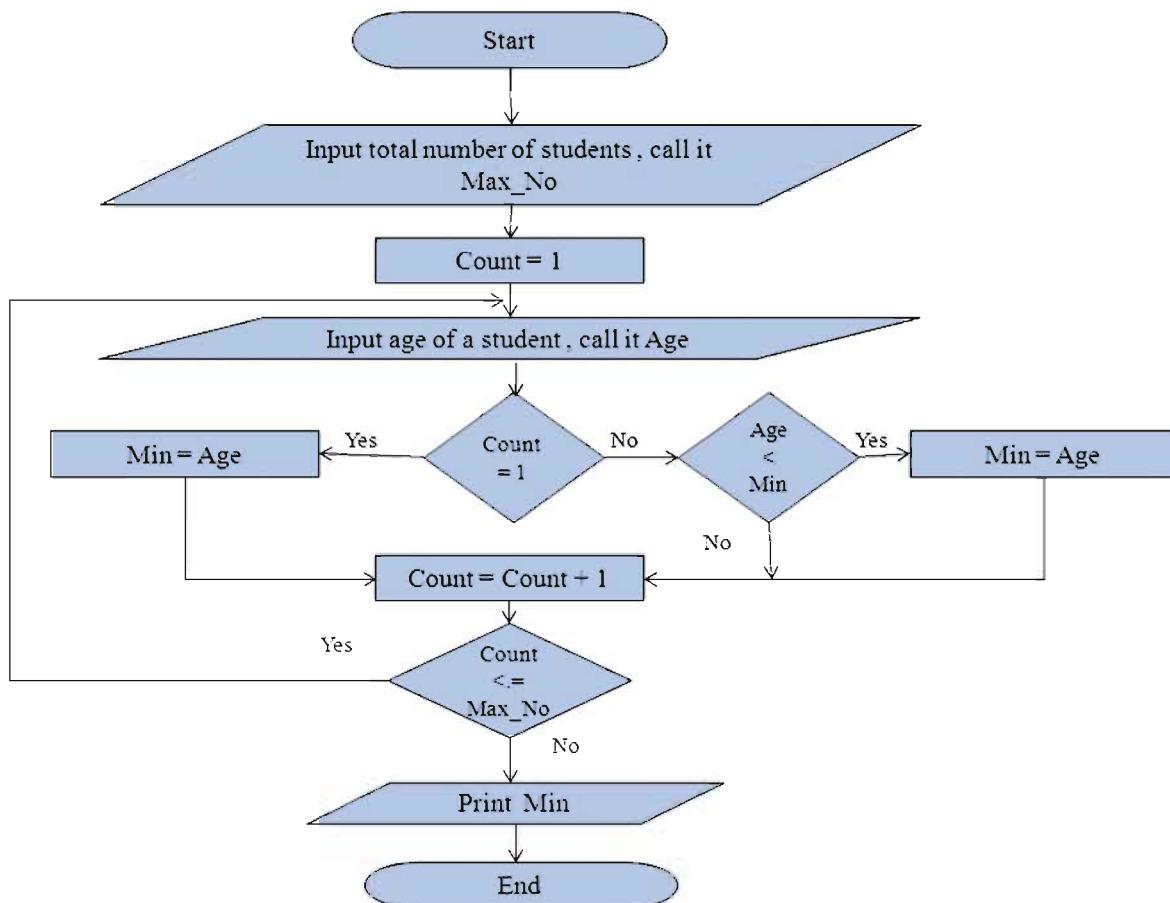


Figure 9.14 : Flowchart to find youngest of any number of students

Problem 7 : Let us now solve one mathematical problem. Assume that we want to find out sum of first 50 odd numbers. That is we want to calculate the sum of $1 + 3 + 5 + 7 + \dots + 99$.

Solution :

Assume that you have two boxes, one in your left hand and other in your right hand. Now put one marble in the left box, and then empty it in the box of your right hand. Now again put three marbles in box in your left hand and transfer the marbles in the box of your right hand. What will

happen ? The box in your right hand contains a total of $1 + 3$ marbles. If we repeat this process with $5, 7, \dots, 99$ marbles. We will get sum of $1 + 3 + 5 + 7 + \dots + 99$ marbles. Note that we had assumed that the box in our right hand was initially empty.

With similar analogy, we have taken two variables, numb and sum. The variable numb is initialized with value 1 and the variable sum is initialized with 0 (similar to an empty box). The variable numb is incremented by 2, at iteration of the loop, and is added to sum. Thus the value of variable numb becomes 1, 3, 5, till 99. The statement $\text{sum} = \text{sum} + \text{numb}$ keeps on adding the values of variable numb into sum till the loop ends. Figure 9.15 shows the flowchart of the problem.

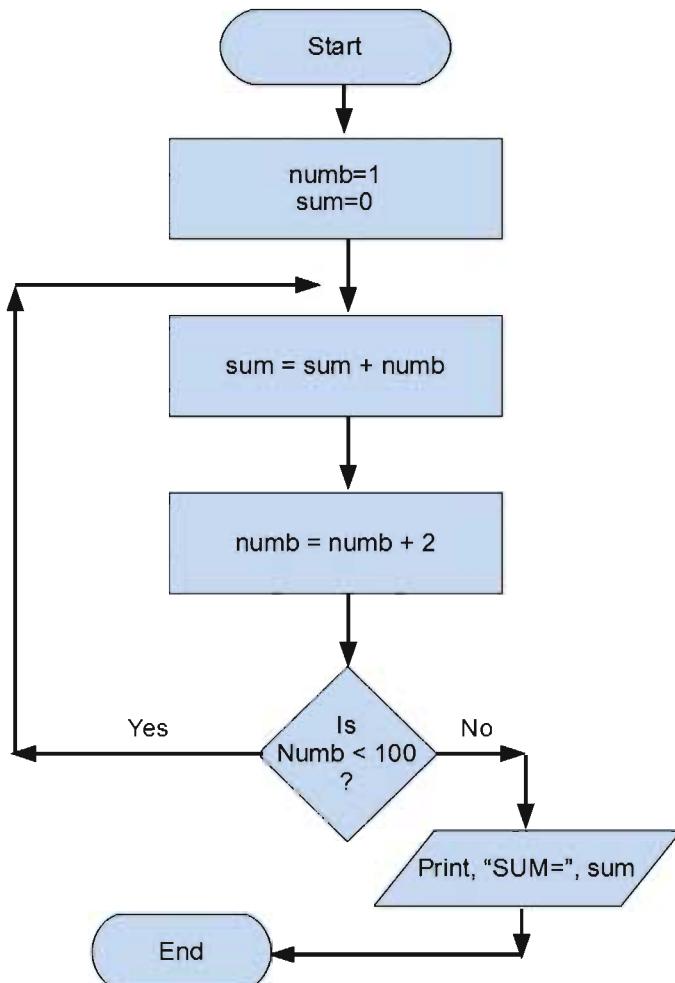


Figure 9.15 : Flowchart to sum of odd numbers till 99

Problem 8 : Let us now have a look at two different ways of exchanging values of two variables.

Solution :

Take two variables say A and B. If we replace the value of either A or B by any other value, the original value stored in them will be lost. Therefore we have used an extra variable C. To make sure that value of A is retained we first transfer it to C. Then value of B is transferred to A and then saved value of A (saved in C) is transferred to B. Figure 9.16 shows the flowchart of this solution.

Figure 9.17 gives a flowchart of the same problem, but it does not use an additional variable. Rather it performs addition and subtraction to interchange the values.

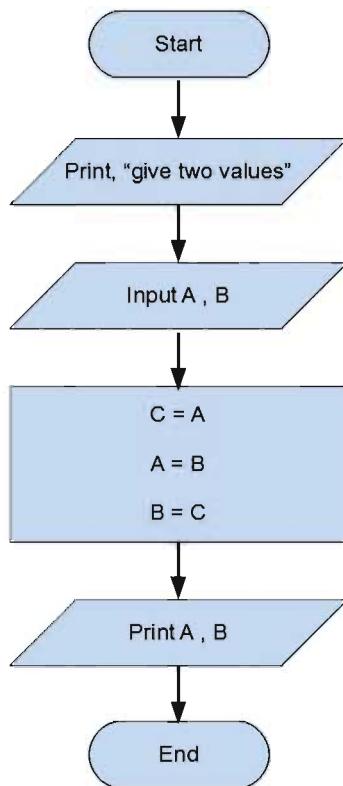


Figure 9.16 : Swapping with extra variable

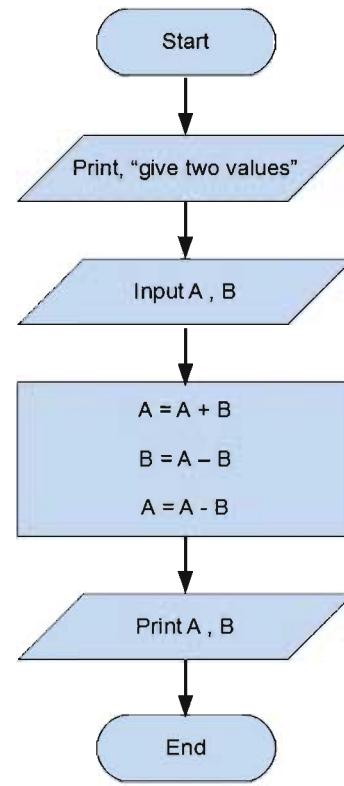


Figure 9.17 : Swapping without extra variable

Advantages and Disadvantages of Flowchart

Whatever tools we use they have some advantages and disadvantages. Flowchart is also not an exception. It also has some benefits and drawbacks.

Advantages of Flowchart :

1. As it provides pictorial representation of the problem solution, it is easy to understand.
2. It provides a convenient aid for writing computer instructions.
3. It assists the programmer in reviewing and correcting the solution steps.
4. It helps in discussion of different methods for solving a problem. This in turn provides us a facility to compare them accurately.

Disadvantages of Flowchart :

1. Drawing a flowchart for a large and complex problem is time consuming and laborious.
2. As flowchart consists of symbols, any changes or modification in the program logic will most of the times require a new flowchart to be drawn.
3. There are no standards specifying the details of data that should be shown in a flowchart. Hence flowchart for a given problem although with similar logic may vary in terms of data shown.

Algorithm

Your friend has reached your home and is now interested in coming to your school. He/She calls you and asks for directions to reach your school. To instruct him/her on how to reach the school you will first need to have some route locations in mind. Assume that you have a route map as shown in figure 9.18 to assist you.

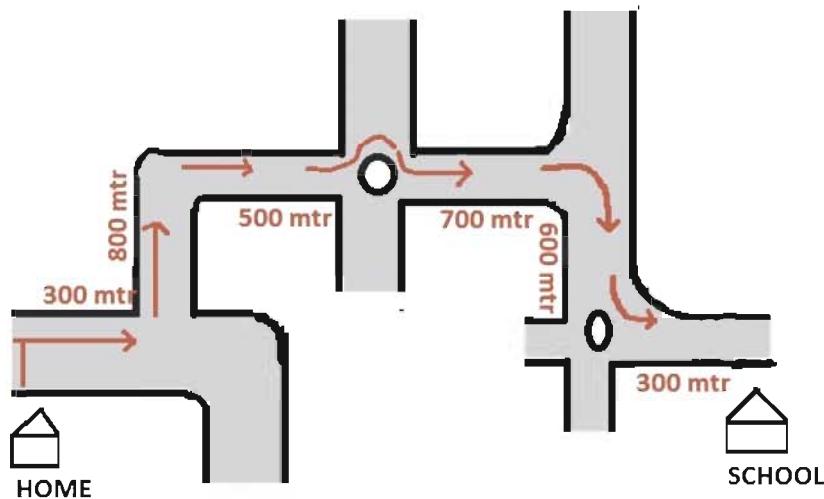


Figure 9.18 : Route Map

Solution :

Using the route map given in figure 9.18, you will be able to guide your friend step by step as mentioned below:

- Step-1** : Start from my home and take right and go for 300 meters.
- Step-2** : Now take a left turn and go straight for another 800 meters.
- Step-3** : Take a right turn and walk for 500 meters, you will encounter a four cross roads with a circle.
- Step-4** : Do not take turn at circle; instead keep on walking in the direction you were walking till another 700 meters.
- Step-5** : Now take a right turn; after 600 meters you will reach another cross roads with an oval shaped fountain in the middle of cross roads.
- Step-6** : Take a left turn from at this junction and walk for 300 meters.
- Step-7** : You will be able to see my school on the right side of the street.

From the above example we can say that the problem of reaching your school is solved by following a sequence of steps. In computer science, an algorithm refers to a step by step procedure for solving a particular problem. An algorithm is written in a natural language like English. Algorithms once written then can be easily converted into computer programs. Let us now learn to write some algorithms.

Problem 1 :

Write an algorithm to find summation of numbers divisible by 11 within the range of 1 to 100.

Algorithm :

- Step-1** : Start
- Step-2** : Take two variable Num and Sum.
- Step-3** : Initialize Num = 1 and Sum = 0.
- Step-4** : Check whether Num is divisible by 11? To do this we have to find remainder when Num is divided by 11. Store the remainder in another variable say R.

$$R = \text{Num} - \text{int}(\text{Num}/11) * 11$$

Step-5 : If R is not equal to 0 go to Step-7.

Step-6 : Sum = Sum + Num

Step-7 : Num = Num + 1

Step-8 : If Num < 100 go to Step-4.

Step-9 : Print Sum

Step-10 : End

Explanation :

To find the remainder, we first divide the number stored in Num by 11 and then we take its integer part. For example if the value of Num is 39 then dividing it by 11, we will get 3.54545455. If we take its integer part, it will be only 3. The portion after decimal point will be removed. When the integer part is multiplied by 11, we will get 33. If this 33 is subtracted from the original number that is 39, we will get a value 6. This value is the remainder. Note that we have assumed a process called int which gives integer part of the number.

Step number 1 to 4 is self explanatory. Step-5 checks the remainder using the technique mentioned above. If the remainder is not equal to 0, it means that Num is not divisible by 11 and we will proceed to take the next number. If Num is divisible by 11, we will jump to step-6. We add the value of Num to Sum and then proceed to step-8. In step-8 we check whether Num is less than 100 or not. This check is used to exit from the loop. We will thus continue till value of Num is less than 100. Finally we will print the value of Sum.

Problem 2 :

Write an algorithm to find compound interest for given, principal amount, time period and rate of interest, when interest is calculated every year.

Algorithm :

We know that the compound interest can be calculated using the formula

$$CI = P * (1 + R/100)^N - P$$

Here P is Principal Amount, R is rate of interest and N is time period in years.

Step-1 : Start

Step-2 : Take four variables P, N, R and CI.

Step-3 : Assign values to P, N and R.

Step-4 : Calculate compound interest using $CI = P * (1 + R/100)^N - P$

Step-5 : Print value stored in CI

Step-6 : End

Problem 3 :

Find out the total weekly pay to be given to an employee based on the number of hours he/she has worked. To get a solution here we need to know the pay per hour that is to be given to the employee.

Algorithm :

We want to find weekly pay, for this we need two inputs, number of hours worked and salary per hour.

Step-1 : Start

Step-2 : Take three variables No_Of_Hrs_Worked, Pay_Per_Hour and Weekly_Pay.

Step-3 : Assign values to No_Of_Hrs_Worked and Pay_Per_Hour.

Step-4 : Calculate weekly pay using formula

Weekly_Pay = No_Of_Hrs_Worked * Pay_Per_Hour

Step-5 : Print value stored in Weekly_Pay

Step-6 : End

Summary

In this chapter, we have learnt about the two most used techniques of problem solving, namely, flowchart and algorithm. We saw different symbols that can be used to draw a flowchart. The user needs to decide between using either a flowchart or an algorithm to solve a problem.

EXERCISE

1. Which are the basic components of a flowchart ?
2. Which kind of problem is solved by computers ?
3. Which types of operations are parts of the computer process ?
4. Which are the operations constitute Arithmetic operation ?
5. Decision box is used for which type of operation ?
6. How do you define a variable ?
7. What do you understand by algorithm ? How it is differ from flowchart ?
8. Choose the correct option from the following :
 - (1) Which of the following symbol is used to begin a flow chart ?



- (2) Which of the following refers to a list of instructions in a proper order to solve a problem called ?
- (a) Algorithm (b) Flowchart (c) Sequence (d) Roadmap
- (3) Which of the following symbol is used to test conditions in a flowchart ?
- (a) Diamond (b) Circle (c) Arrow (d) Square
- (4) Which of the following symbol is used to represent output in a flowchart ?
- (a) Square (b) Circle (c) Parallelogram (d) Triangle
- (5) Which of the following is the standard terminal symbol for a flowchart ?
- (a) Circle (b) Diamond
- (c) Rounded Rectangle (d) Square
- (6) Which of the following refers to the purpose of Algorithm and Flowchart ?
- (a) Know the memory capacity
(b) Identify the base of the number system
(c) Direct the output to the printer
(d) Specify the problem completely and clearly
- (7) Which of the following is not a problem solving technique ?
- (a) Pseudo code (b) Flowchart (c) Algorithm (d) Sequence
- (8) Which of the following is a pictorial representation of a problem solving technique ?
- (a) Pseudo code (b) Flowchart
(c) Algorithm (d) Computer program
- (9) An arrow symbol in flowchart is used to show
- (a) The flow of an action
(b) The sequence of action
(c) The start of actions
(d) The completion of an action
- (10) Which of the following refers to the core part of any solution ?
- (a) Input (b) Output (c) Process (d) Algorithm
- (11) Which of the following symbol represents a Process ?
- (a) Rectangle (b) Square (c) Circle (d) Diamond
- (12) Which of the following is used to distinguish different connector pairs in flowchart ?
- (a) Arrows are used
(b) Alphabets or other characters are used
(c) Circles are used
(d) Diamonds are used

LABORATORY EXERCISE

Draw a flowchart and write an algorithm to perform the following operations :

1. Convert the given meters into centimeters.
2. Convert the given centigrade to Fahrenheit.
3. Find minimum of the given two numbers.
4. Find minimum of the given two numbers and check whether the minimum number is odd or even.
5. Find minimum of the given three numbers.
6. Check whether a given number is divisible by other given number.
7. Find average of three numbers.
8. Given total marks of student find whether the student has passed or failed. (If total marks are less than 35 then the student is declared as fail.)
9. Given total marks of student find what grade the student has obtained. (If total marks are less than 35 then the student gets D grade. If total marks are between 35 and 60 then C grade, if total marks between 60 and 70 then B grade, above 70 A grade)
10. Calculate the compound interest on a loan amount of Rs. X taken at the rate of R% for N years.
11. Find the summation of digits of a given number. For example if the given number is 8327 the output will be 20 ($8 + 3 + 2 + 7$)





Introduction to C Language

In chapter 9 we have learnt what is a flowchart and algorithm. We also learnt how to develop an algorithm. The fact is that flowchart and algorithm form basic solution to any problem. We then convert this flowchart or algorithm into a program. For writing programs many languages are there. C language is one of them. In this chapter we will be introducing you to C language.

Need of Programming Language

The first question we might ask is what is the need of programming language? Why not write program in the languages that we already know how to write and read in like English or Gujarati. Well the answer is, a sentence in such languages may not have just one meaning. This does not work well with computers. For computers to work properly each sentence has to be clear and precise.

Programming language when used allows us to write an instruction that has only one meaning. It consists of set of predefined rules. These rules form syntax of that language. Hence learning a programming language is only learning new syntax to represent the instructions we want to give to the computer. It is like learning new grammar when we learn new language.

Need of Translator

Computers however do not understand the language that we speak neither does it understand the programming language. It only understands 0 and 1. The problem discussed here is similar to the problem that we face at times in our lives. Consider that there are two persons, person X and person Y who don't know each other's language. Person X knows Gujarati while person Y knows Hindi. How would these two persons communicate? The option is third person Z who knows both languages. So now when person X wants to communicate a message to person Y, he communicates the message to person Z in Gujarati first. Person Z then translates it to Hindi and then communicates the message to person Y. Same process is repeated in reverse order when person Y wants to communicate with person X. Here person Z is known as translator and the process of converting one language to another is known as translation.

The problem of computers not understanding our language is solved by using software programs called translators. This translator is known as compiler. The process of translation has been discussed at the end of this chapter.

Program and Characteristics of Program

A program can be defined as finite set of precise and clear instructions given to a computer for performing a predefined task. The process of writing these step by step instructions using a chosen language is known as programming.

A good program should possess following characteristics :

1. A program must end after finite number of steps.
2. The instructions of program must be precisely defined, i.e. it should not have multiple meaning.
3. All the instructions must be effective, i.e. they should be carried out exactly.

4. A program may take zero or more inputs.
5. A program may produce one or more outputs.

As can be seen these characteristics are similar to characteristics of an algorithm. Example 10.1 gives a sample C program. Figure 10.1 shows the code listing and output of example 10.1 as can be seen in SciTE editor. We will learn how to use SciTE editor at the end of this chapter.

The screenshot shows the SciTE editor interface with the title bar "10_1.c - SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, and Help. The buffer tab shows "1 10_1.c". The code listing is as follows:

```

/* Example 1: My first C program */

#include <stdio.h>
int main( )
{
    printf("Welcome to the world of C programming using Scite \n");
    return 0;
}

```

To the right of the code, the output is shown:

```

>gcc -pedantic -Os -std=c99 10_1.c -o 10_1
>Exit code: 0
>/10_1
Welcome to the world of C programming using Scite
>Exit code: 0

```

Figure 10.1 : My first C Program

This program is used for printing a message "Welcome to the world of C programming using SciTE". Here main() is known as user defined function, while printf() is known as inbuilt or library function. User defined functions are the functions that a user creates as per his or her requirement. The functions already available in C that a user can use in his or her program are known as inbuilt or library functions.

Before we start discussing in detail about various C language components let us have a look at another C program shown in figure 10.2 and its output shown in figure 10.3.

The screenshot shows the SciTE editor interface with the title bar "10_2.c - SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, and Help. The buffer tab shows "1 10_2.c". The code listing is as follows:

```

/* Example 2: Program to calculate Circumference of a circle */

#include <stdio.h>
#define PI 3.14
int main( )
{
    float radius, circumference;
    printf("\nEnter the value of radius: ");
    scanf("%f", &radius);
    circumference = 2 * PI * radius;
    printf("\nCircumference of circle with radius %f is %f \n", radius, circumference);
    return 0;
}
/* End of program */

```

Figure 10.2 : Code listing of Example 10.2

The screenshot shows a terminal window with the following content:

```
harshal@harshal-Compaq-435-Notebook-PC: ~/Desktop/Chapter10
File Edit View Search Terminal Help
harshal@harshal-Compaq-435-Notebook-PC:~/Desktop/Chapter10$ ./a.out
Enter the value of radius: 2.5
Circumference of circle with radius 2.500000 is 15.700000
harshal@harshal-Compaq-435-Notebook-PC:~/Desktop/Chapter10$
```

Figure 10.3 : Output of Example 10.2

Explanation

The program given here is used for calculating circumference of a circle for a given radius.

- The first statement is a comment that indicates the purpose of a program.
- The second statement instructs the compiler to include the contents of header file called "stdio.h". This file contains information about the inbuilt functions to be used.
- The third statement defines a symbolic constant PI that has value 3.14. This statement is known as pre processor directive. It instructs the translator to replace all occurrence of PI with 3.14 before executing the program.
- The fourth statement is a user defined function main(), a must for all executable C programs. Here this function returns and integer value hence the statement int main().
- The fifth statement is beginning of function main().
- The sixth statement defines two variables radius and circumference. These variables are capable of storing real values. An entity in C that is capable of storing different values is known as variable.
- The seventh statement uses an inbuilt function printf to print a message.
- The eighth statement uses inbuilt function scanf to read a value of "radius" from user.
- The ninth statement is a C expression; it is a formula for calculating the circumference of circle.
- The tenth statement prints a message along with "radius" and "circumference" on the screen.
- The eleventh statement provides normal exit from the function. If we don't write this statement we will get a warning message "Function should return a value". This may not affect the working of the program, but a normal exit is always a better option.
- The twelfth statement indicates the end of the function main().

As can be seen from example 10.1 and 10.2 main() function forms an integral part of C programs. Rather we should say that all executable C programs will have one user defined function named as main(). The execution of every C program starts from the first open curly bracket when main() is encountered.

Structure of C Program

As seen from the previous examples a C program is a set of blocks called functions. A function is made up of one or more statements used for performing a predefined task. A complete C program structure is given in figure 10.4. Let us discuss these sections in detail.

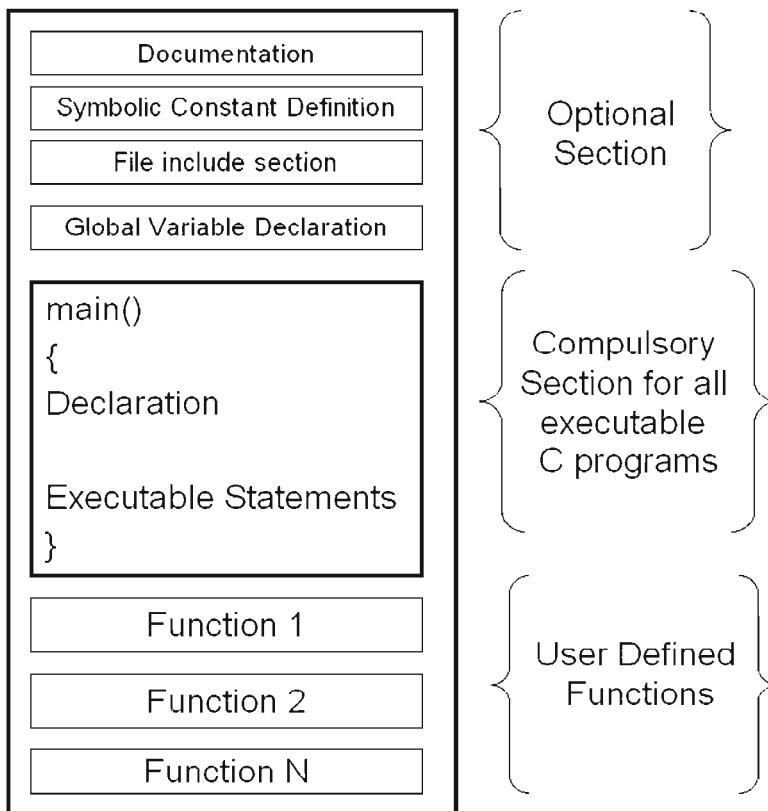


Figure 10.4 : Structure of complete C program

Documentation Section

This section is an optional section. As the name indicates it is used for the purpose of documentation. It is enclosed within /* and */. Whenever the text is enclosed between /* and */, it is considered as a comment in C. Comments are not processed by C compiler and are left as it is.

This section is normally used to tell about the purpose of program, author, date of creation etc. Comments in C program can be added anywhere. It is always a good practice to use comments within the functions as it improves the readability and understanding of the program.

Symbolic Constant Definition

As seen in example 10.2 we have used a symbolic constant PI in our program. To use such symbolic constant in our program we need to use #define as a prefix to it. Here #define is known as pre processor directive. It instructs the compiler to replace all occurrences of symbolic constants with the values specified against it. Normally symbolic constants are defined using capital letters. Doing this differentiates them from normal variables.

File include Section

C provides inbuilt or library functions. Some examples are pow(), sqrt() etc. These functions have a predefined purpose like pow() is used for calculating value of x raised to given power,

`sqrt()` is used to find square root of a given number. We can use these functions in our program if needed. To use them we have to include files that hold information about these functions. These files are known as header files in C. The extension of header file is ".h". We use the syntax `#include <filename.h>` to include header files in our program. Appendix IV gives the list of header files available in C and their uses.

Global Variable Declaration Section

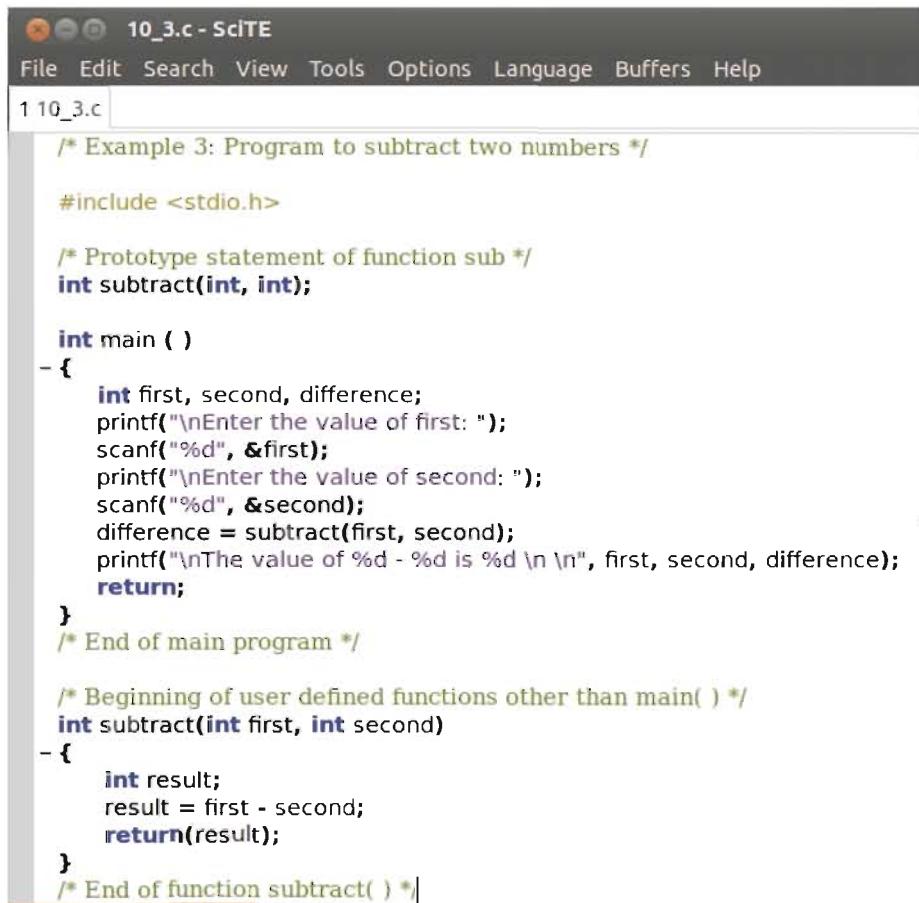
C variables are governed by scope. A scope of C variable is decided by using opening and closing curly braces { }. The variables defined within curly braces are known as local variables. For example the variables "radius" and "circumference" used in example 10.2 are local variables of function `main()`. These variables cannot be used outside the scope. At times we need to use a variable in all the functions, such a variable is known as global variable. This variable is defined before defining all the functions.

Main Function

All C programs contain one function with the name `main()`. This is the function from where the execution of any C program starts. The control is first transferred to this function and from here rest of the operations are carried out.

User Defined Function

C provides us a facility of breaking a single program into set of small pieces. These pieces are known as functions. In this section we define all the additional functions used in our program. Example 10.3 shown in figure 10.5 shows a program that consists of two user defined functions.



The screenshot shows the SciTE code editor interface with the title bar "10_3.c - SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, and Help. The code area contains the following C program:

```
/* Example 3: Program to subtract two numbers */

#include <stdio.h>

/* Prototype statement of function sub */
int subtract(int, int);

int main ( )
{
    int first, second, difference;
    printf("\nEnter the value of first: ");
    scanf("%d", &first);
    printf("\nEnter the value of second: ");
    scanf("%d", &second);
    difference = subtract(first, second);
    printf("\nThe value of %d - %d is %d \n \n", first, second, difference);
    return;
}

/* End of main program */

/* Beginning of user defined functions other than main( ) */
int subtract(int first, int second)
{
    int result;
    result = first - second;
    return(result);
}

/* End of function subtract( ) */
```

Figure 10.5 : Code listing of Example 10.3

Explanation

This program is used to calculate subtraction of two given numbers. Although it is a small program we have divided it into two user defined functions. The first function main() accepts the values to be subtracted and second function subtract() performs the actual subtraction.

Observe that we have written a prototype statement before actually using the function subtract. It is needed in C if we write a user defined function after main(). The main function declares three integer variables, and asks user to input value of two of them. These values are then passed to function subtract() by using statement difference = subtract(first, second). Here first and second are known as parameters of function subtract(). After encountering this statement the control is transferred to function subtract(). This function then subtracts the value and stores it in "result". The value of result is then returned to statement in main() and assigned to "difference". Finally main() function prints the value of subtraction and exits the program. The output of example 10.3 is shown in figure 10.6.

```
harshal@harshal-Compaq-435-Notebook-PC: ~/Desktop/Chapter10
File Edit View Search Terminal Help
harshal@harshal-Compaq-435-Notebook-PC:~/Desktop/Chapter10$ ./a.out
Enter the value of first: 55
Enter the value of second: 24
The value of 55 - 24 is 31
harshal@harshal-Compaq-435-Notebook-PC:~/Desktop/Chapter10$
```

Figure 10.6 : Output of Example 10.3

The structure of C program shown in figure 10.4 although used by almost all programmers is not a standard. It is possible that some sections may not be there at all as can be seen in example 10.1. It is also possible to interchange certain section with others. For example user defined function section can be interchanged with main program section as can be seen in code listing 10.1. Here we have kept the user defined function before the main() function.

```
/* Example 4: Program to subtract two numbers */

#include <stdio.h>

/* Beginning of user defined functions other than main( ) */
int subtract(int first, int second)
{
    int result;
    result = first - second;
    return(result);
}
/* End of function subtract( ) */
```

```

/* Beginning of function main() */
int main ( )
{
    int first, second, difference;
    printf("\nEnter the value of first: ");
    scanf("%d", &first);
    printf("\nEnter the value of second: ");
    scanf("%d", &second);
    difference = subtract(first, second);
    printf("\nThe value of %d - %d is %d \n \n", first, second, difference);
    return;
}
/* End of main program */

```

Code Listing 10.1 : Code of Example 10.4

This program is similar to the one shown in example 10.3 except for two differences. First the function subtract() is defined before main() and second, we have not used the prototype statement. In example 10.3 on encountering the statement difference = subtract(first, second); the control flow goes below main(), in case of example 10.4 the control flow will go above main().

C Character Set

Recall that when we learnt a new language in school, we first had to learn its alphabets. These alphabets form a basic set in constructing words and sentences in that language. C language also has its own character set. The characters here can be divided into four categories.

1. Letter
2. Digits
3. White spaces
4. Special Characters.

Table 10.1 lists the characters that fall under these categories.

Letters	Digits	White Spaces
A to Z a to z	0 to 9	Blank Space
		Form feed
		Horizontal tab
		New line
		Vertical tab

Special Characters			
Operator	Use	Operator	Use
&	Ampersand	>	Greater than
'	Apostrophe	#	Hash sign
*	Asterisk	<	Less than
@	At the rate	-	Minus
\	Backward slash	()	Parenthesis left / right
{ }	Braces left / right	%	Percent
[]	Bracket left / right	.	Period
^	Caret	+	Plus
:	Colon	?	Question mark
,	Comma	"	Quotation mark
\$	Dollar	;	Semi colon
=	Equal to	~	Tilde
!	Exclamation	_	Under score
/	Forward slash		Vertical bar

Table 10.1 : C Character set

As can be seen in table 10.1, we have used characters that belong to English language. Hence now we would learn to use these characters as per C language syntax.

C Words

The character set given in table 10.1 is used to form words in C language. These words are used to construct a C statement. Set of logically sequential C statements thus is identified as a C program. A word in C is known as a token of C. Each character discussed in table 10.1 is itself a token. C basically identifies six types of tokens keyword, identifier, constant, string, operator and special character. Figure 10.7 shows how these tokens are used in C program.

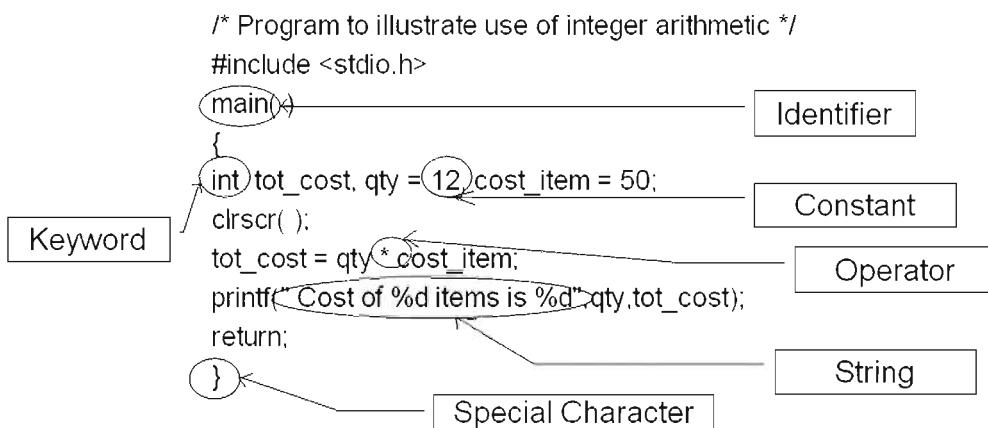


Figure 10.7 : Use of words in C program

Keyword

All of us must have used a dictionary at some point of time. We used it for finding a meaning of a predefined word of that language. C also has such predefined words. To be specific ANSI C standard supports 32 predefined words. These predefined words in C language are known as keyword. Each keyword has a predefined meaning associated to it. Table 10.2 gives the list of the keywords available in ANSI C.

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Table 10.2 : ANSI C keywords

The keywords and their use have been discussed in subsequent chapters throughout the course of study.

Identifier

A word that a user can construct by making use of C character set is known as identifier. It consists of set of letters, digits and some special characters. Some example of identifiers are "difference", "main()", "PI", "float" etc. Here "difference" is name of a variable, "main()" is name of a function, "PI" is name of a symbolic constant and "float" is a predefined word. Before continuing further let us see what is variable.

Variable

A C program generally accepts input, this input comes in a form of data. To store and manipulate a data we use memory space. The data within this memory space can vary based on the operations performed on it. The data within the memory space is referred by a name known as variable name. The data as it is capable of changing itself is called variable. To define a variable name in C we have to adhere to certain rules. These rules have been mentioned below :

1. Variable name cannot be same as keyword.
2. Variable name consists of letter, digit and under score. No other special character is allowed.
3. The first character of variable name must be a letter or under score.
4. The maximum length of variable name as per ANSI standards is 31 characters. However, it is dependent on compiler.
5. The variable names are case sensitive hence num, nuM, nUM, nUm and Num are considered as different variables.

Table 10.3 gives some examples valid and invalid variables of C language.

Valid
Double → Double is not same as keyword double.
INTEREST → Capital letters are part of C character set.
total_quantity → Underscore and characters are allowed.
mark100 → As first letter is character it is allowed.
_file → First letter can be underscore.
Invalid
char → Use of keyword not allowed.
Total Value → Blank space not allowed.
total&value → Special character not allowed.
10mark → First character has to be character or underscore.

Table 10.3 : Example of C variable

Constant

The entities of C that do not change its value throughout the execution of program are known as constant. C constants can be divided into different categories as shown in figure 10.7. Let us discuss these constants in detail.

Numeric Constant

The constant that store numeric value is known as numeric constant. The numeric constants are divided in two categories, integer constants and real constants.

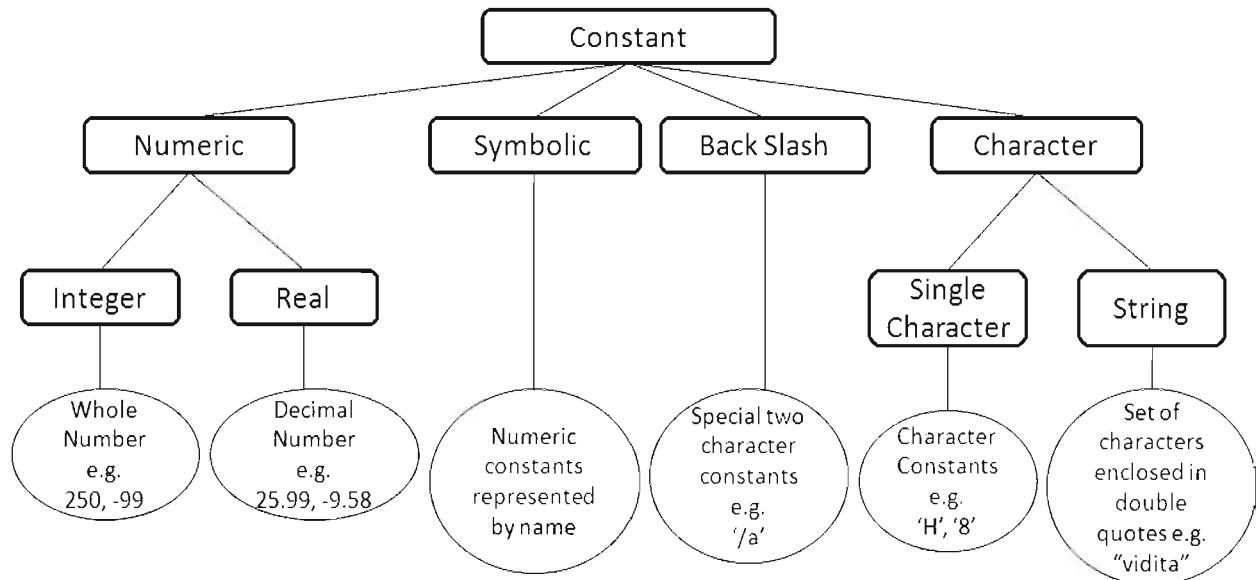


Figure 10.7 : C Constants

Integer Constant

Integer constant refers to whole numbers. We have three types of integer constants decimal, hexadecimal and octal. Example of various number systems is given in Appendix I.

The decimal constants use numeric digits from 0 to 9. We can use as prefix unary plus or minus with such numbers.

The hexadecimal constants use numeric digits from 0 to 9 and letters A to F. Here A to F refers to numeric values 10 to 15 respectively. The hexadecimal numbers have base 16. When using this value in C we differentiate it from decimal numbers by using 0x or 0X as prefix.

The octal constants use numeric digits from 0 to 7. The base of this number system is 8. When using this value in C we differentiate it from decimal numbers by using 0 as prefix. Table 10.4 gives a list of some valid and invalid integer constants

Valid
40000 → Valid decimal positive whole number.
-120 → Valid decimal negative whole number.
79999L → Valid decimal positive long whole number.
0xB5 → Valid Hexadecimal number.
0X79 → Valid Hexadecimal number.
045 → Valid Octal number.
Invalid
25,000 → comma is not allowed.
-100.0 → it is real constant.
Rs79999 → Alphabets are not allowed.
0xG → G is not part of hexadecimal numbers.
0X8,9 → Comma is not allowed.
096 → 9 is not part of octal numbers

Table 10.4 : Integer Constants

Real Constant

Real constant refer to decimal numbers that have fractional part. Example of real constant are 10.50, -65.75 etc. Real constant can also be represented by using scientific form. In this form a number is represented using a mantissa and exponent. For example the value 25.75 can be represented as 0.2575 e 2 or 0.2575 E 2. Here 0.2575 is known as mantissa and 2 is known as exponent, also "e" and "E" is equivalent. Table 10.5 gives a list of some valid and invalid real constants.

Valid
250.00 → Valid decimal fractional number.
295 → Valid decimal number. Automatically converted to 295.00
-15.55 → Valid negative decimal number.
-20.5e5 → Valid scientific value.
15E-2 → Valid scientific value.
Invalid
19,800.00 → comma is not allowed.
\$786 → Special character not allowed.
-9.4 e 5.0 → Exponent should be integer.
51E -2 → Blank spaces not allowed.

Table 10.5 : Real Constants

Character Constant

Character constants as the name suggests contain characters. There are two types of character constants in C. Single character constant and String constant.

Single Character Constant

A single character constant is represented by using single character enclosed within single quotes. Some example of single character constants are 'H', 'v', '7', '\$' etc. Here each character constant is associated with a numeric value known as ASCII (American Standard Code for Information Interchange) value. Appendix II gives detail of ASCII values associated with different characters.

String Constant

String constants are denoted by using sequence of characters enclosed within double quotes. Some examples of string constants are "C Language", "Bye", "V". String constants don't have any ASCII value associated with it. Here string constant "V" is not equal to single character constant 'V'. Here string "V" will be allocated two memory spaces while character 'V' will be allocated only one memory space. This is due to the fact that strings in C are ended by adding '\0', a null character.

Back Slash Characters

The single character constant as the name suggests use single character, while string character uses sequence of characters. C also provides a special character constant that uses two characters. These constants are known as back slash characters or escape sequences. They are known as back slash characters as the first character is always a back slash i.e. "\", while they are known as escape sequence characters as the output of these constants is not a character but white spaces. Similar to single character constants, these constants also have an ASCII value associated with them. Table 10.6 lists the back slash constant available in C along with their use and ASCII values.

Back Slash Character	Use	ASCII Value
\0	Inserting null value	0
\a	Inserting audible alert	7
\b	Inserting backspace	8
\t	Inserting horizontal tab	9
\n	Inserting new line	10
\v	Inserting vertical tab	11
\f	Inserting form feed	12
\r	Inserting carriage return	13
\"	Inserting double quotes	34
\'	Inserting single quote	39
\?	Inserting question mark	63
\\\	Inserting back slash	92

Table 10.6 : Back Slash characters

The escape sequences are mostly used for formatting purpose. For example '\n' is used for inserting new line at the time of input or output. Throughout the book you will be able to see the use of these characters.

Symbolic Constant

Certain numeric and character constants can be defined using symbolic identifier. Such constants are known as symbolic constant. To define a symbolic constant we use the given syntax:

#define identifier value

Here #define is known as a preprocessor directive. Identifier is symbolic name of the constant that we want to define, and value is the constant itself. Some examples of symbolic constants are

#define PI 3.14

define MAXVALUE 100

#define f float

The first statement here defines a symbolic constant called "PI" that refers to a real value "3.14". Second statement defines a symbolic constant called "MAXVALUE" that refers to an integer value "100" and the last statement defines a symbolic constant called "f" that refers to a keyword "float" in C language.

The preprocessor directive statement instructs the compiler that the occurrence of the symbolic constant used in program should be replaced by the constant value specified in the definition. Example 10.5 shows a program to find surface area of a sphere and uses preprocessor directives. Figure 10.8 shows the code listing of example 10.5 while Figure 10.9 shows the output of the program.

```

10_5.c - SciTE
File Edit Search View Tools Options Language Buffers Help
1 10_5.c
/* Example 5: Program to find surface area of a sphere */

#include <stdio.h>

/* Definition of a symbolic constant */

#define F float
#define P printf
#define S scanf
#define PI 3.14

int main( )
{
    F radius, sarea;
    P("\nEnter the value of radius: ");
    S("%f", &radius);
    sarea = 4 * PI * radius * radius;
    P("\nSurface Area of sphere with radius %.2f is %.2f\n", radius, sarea);
    return 0;
}
/* End of program */

```

Figure 10.8 : Code listing of Example 10.5

```

harshal@harshal-Compaq-435-Notebook-PC: ~/Desktop/Chapter10
File Edit View Terminal Help
harshal@harshal-Compaq-435-Notebook-PC:~/Desktop/Chapter10$ ./a.out
Enter the value of radius: 2.5
Surface Area of sphere with radius 2.50 is 78.50
harshal@harshal-Compaq-435-Notebook-PC:~/Desktop/Chapter10$ 

```

Figure 10.9 : Output of Example 10.5

Explanation

In this program we are defining four symbolic constants F that refers to a keyword float, P that refers to a function printf, S that refers to a keyword scanf and PI that refers to a float value 3.14. The first statement within function main defines two variables of type F. As F represents float this statement actually defines two float variables. The third statement uses P to print a message. Then we read the value of radius using S and calculate the surface area of sphere. Finally we again use P to print a message and value of surface area.

Points to Remember in C Program

C programming though easy needs a bit of a practice. The points given in this section needs to be remembered by all C programmers.

- Header file should be included before the main() function.
- An executable C program should always contain a main() function.

- Execution of C program starts from the first opening curly brackets after main().
- Two functions with same name are not allowed in C program.
- Normally C programs consist of lower letter alphabets.
- Identifiers are case sensitive.
- There should be atleast one blank space between two words of a C program.
- Usually C statements end with a semi colon.
- Comment can be inserted wherever a blank space can be inserted.
- Every opening curly bracket should have a corresponding closing curly bracket.

Execution of C Program

Till now we have seen so many C programs and their output. Let us now learn to execute the C program. Before taking hands on experience let us understand the actual steps that we need to perform. Figure 10.9 shows the entire process.

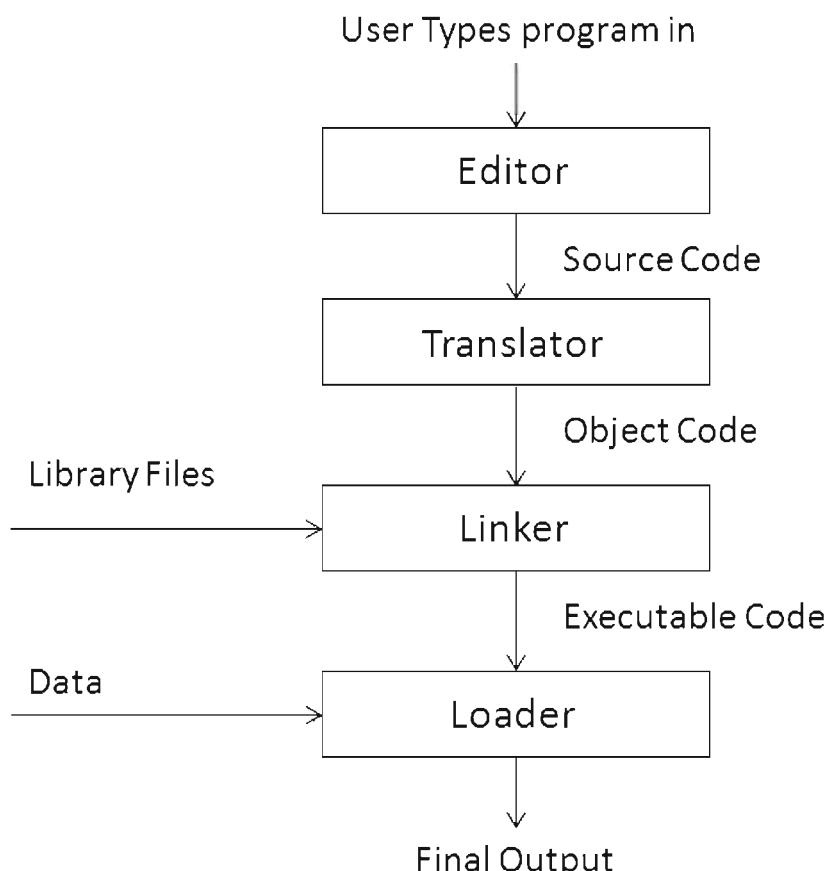


Figure 10.9 : Steps of executing C program

As seen in figure 10.9, the first step is to write a program using any text editor, the program that we write using an editor is known as source code or source program. The extension of the source file written in C language is "c". We then compile the source program using a compiler. A compiler is the translator that converts a source program into machine language program known as object code or object program. Different formats of object code are available. A program called linker is then used to link the object code with the library functions giving executable program or executable code. Finally executable code is loaded on to memory by a program called loader along with the data required to give us the output.

Throughout this book we have discussed a compiler known as gcc compiler. The gcc compiler is provided by Free Software Foundation, it is a Unix/Linux based ANSI C compiler. It is usually operated via the command line but can also be integrated with a text editor like SciTE or an Integrated Development Environment (IDE) like Eclipse. It often comes bundled with almost all Linux installations hence we can simply start using it without any problem.

Let us try to use the compiler, First we need to write a program using a text editor. We can use any good text editor to create a C program. In Linux environment, we have editors such as vi, gedit, emacs and many more. Choose the one you are comfortable with. The only thing that should be kept in mind is the fact that the file created should be saved with an extension c. Make sure that the character c is written in lower case. Let us try to use gedit to create a C program. Open terminal, at the prompt type gedit myprogram.c this will open a blank editor as shown in figure 10.10.

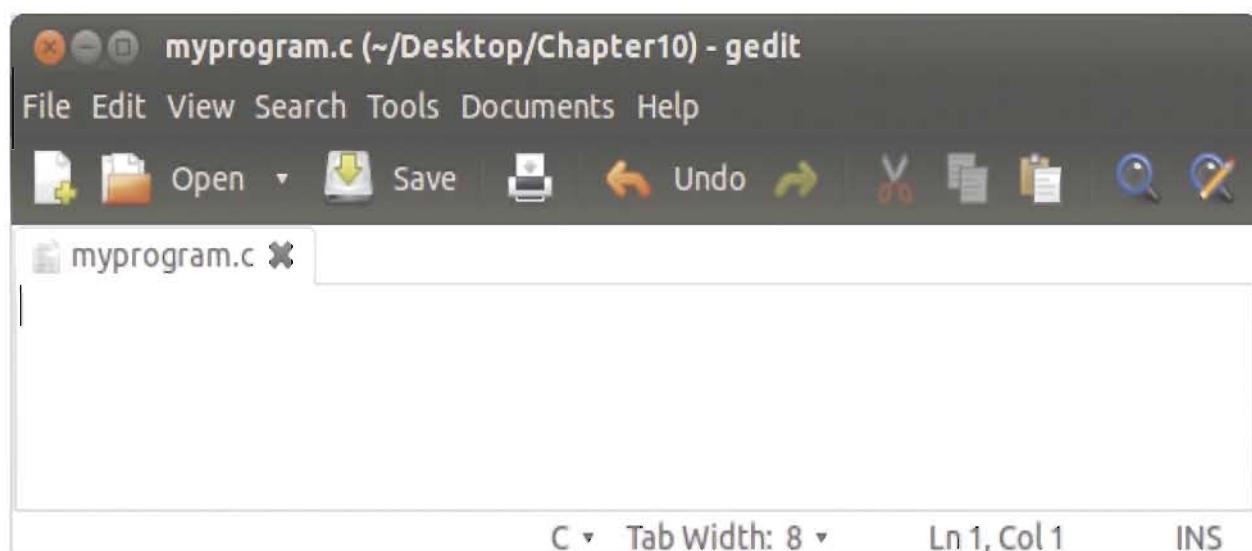


Figure 10.10 : Blank editor screen

Now type the contents given in code listing 10.2 in the blank editor screen.

```
/* Example 6: My C program */

#include <stdio.h>
int main()
{
    printf("\nWelcome to the world of C programming using gcc\n");
    return 0;
}
```

Code Listing 10.2 : Code of Example 10.6

Once the program is ready your editor will look similar to the figure 10.11, save it and close the editor.

The screenshot shows a window titled "myprogram.c (~/Desktop/Chapter10) - gedit". The menu bar includes File, Edit, View, Search, Tools, Documents, and Help. The toolbar contains icons for Open, Save, Undo, Redo, Cut, Copy, Paste, Find, and Replace. A tab labeled "myprogram.c" is open. The code in the editor is:

```
/* Example 6: My C program */

#include <stdio.h>
int main()
{
    printf("\nWelcome to the world of C programming using gcc\n");
    return 0;
}
```

At the bottom, status bars show "C" and "Tab Width: 8", "Ln 8, Col 2", and "INS".

Figure 10.11 : Editor with code listing of Example 10.6

Now we are ready to compile it, go back to the terminal and type the command mentioned:

```
$ gcc myprogram.c
```

If there is no error in the program, a new prompt line will be visible. It means that the compilation is successful and an executable file with the default name a.out is created in the same directory as the source file. In case, we have errors, then appropriate error message would appear on the screen. If error occurs, we need to rectify the program and recompile. To see the output of the program, type the command as mentioned and press Enter key:

```
$ ./a.out
```

This command will display the output of myprogram.c on the screen as shown in figure 10.12.

The terminal window shows the following session:

```
harshal@harshal-Compaq-435-Notebook-PC: ~/Desktop/Chapter10
File Edit View Search Terminal Help
harshal@harshal-Compaq-435-Notebook-PC:~/Desktop/Chapter10$ gedit myprogram.c
harshal@harshal-Compaq-435-Notebook-PC:~/Desktop/Chapter10$ gcc myprogram.c
harshal@harshal-Compaq-435-Notebook-PC:~/Desktop/Chapter10$ ./a.out

Welcome to the world of C programming using gcc
harshal@harshal-Compaq-435-Notebook-PC:~/Desktop/Chapter10$
```

Figure 10.12 : Output of Example 10.6

As compilation of all C programs generates a default output file named a.out, it is a good practice to give a name to this output file. Thus, an output file with the given name once created can be reused many times until there is a change in the actual source code. The following command shows how to give a name to output file using gcc command :

```
$ gcc -o myprogram.o myprogram.c
```

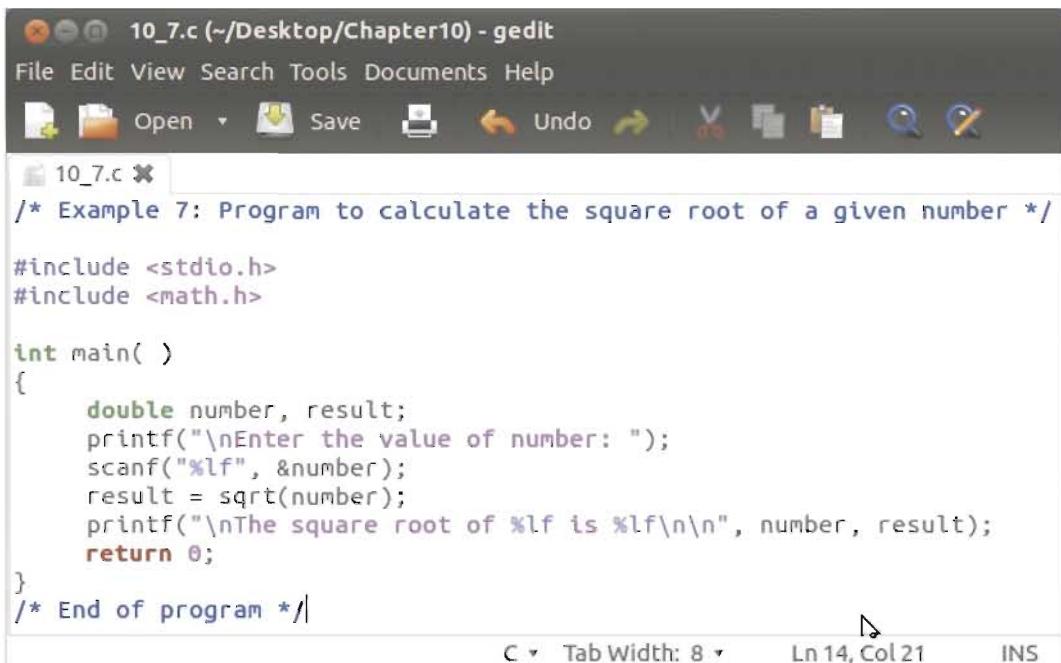
In this command the first parameter myprogram.o represents the output file name, while the second parameter represents the source file name. Observe that we have written .o as an extension for the first parameter, it is just an indication that this file is an output file. The compiler though does not require this extension. We may simple write myprogram instead of myprogram.o. After a successful execution of the preceding command, the compiler will now create a file named myprogram.o instead of a.out. We can now see the output of the program by using the command mentioned herewith :

```
$ ./myprogram.o
```

Many other parameters can also be used along with gcc command. To get the detailed help of gcc command type the help command as mentioned :

```
$ man gcc
```

Let us try to write and compile one more program. Example 10.7 uses an inbuilt function to calculate the square root of a number entered by the user. Figure 10.13 gives the code listing of the example.



The screenshot shows a gedit window titled "10_7.c (~/Desktop/Chapter10) - gedit". The menu bar includes File, Edit, View, Search, Tools, Documents, and Help. The toolbar includes Open, Save, Undo, Redo, and other file operations. The code editor contains the following C program:

```
10_7.c (~/Desktop/Chapter10) - gedit
File Edit View Search Tools Documents Help
Open Save Undo Redo
10_7.c
/* Example 7: Program to calculate the square root of a given number */

#include <stdio.h>
#include <math.h>

int main( )
{
    double number, result;
    printf("\nEnter the value of number: ");
    scanf("%lf", &number);
    result = sqrt(number);
    printf("\nThe square root of %lf is %lf\n\n", number, result);
    return 0;
}
/* End of program */
```

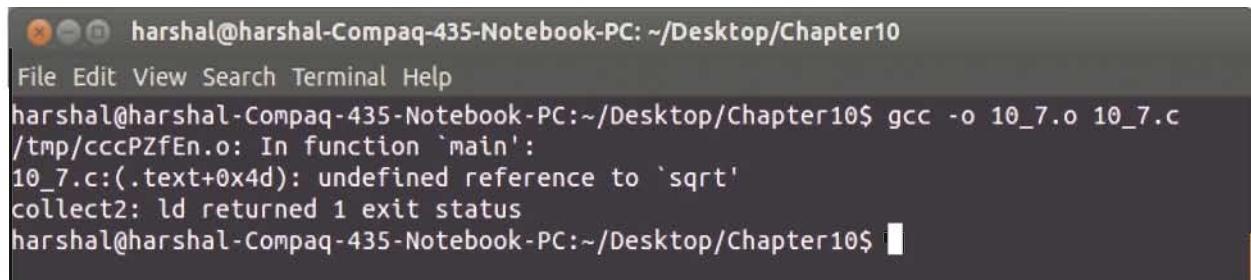
The status bar at the bottom shows "C Tab Width: 8 Ln 14, Col 21 INS".

Figure 10.13 : Code listing of Example 10.7

Let us now try to compile this program using gcc. Open terminal and type the command mentioned below at the prompt.

```
$ gcc -o 10_7.o 10_7.c
```

Figure 10.14 gives the output of this gcc command.



The terminal window shows the following output:

```
harshal@harshal-Compaq-435-Notebook-PC: ~/Desktop/Chapter10$ gcc -o 10_7.o 10_7.c
/tmp/cccPZFEn.o: In function `main':
10_7.c:(.text+0x4d): undefined reference to `sqrt'
collect2: ld returned 1 exit status
harshal@harshal-Compaq-435-Notebook-PC:~/Desktop/Chapter10$
```

Figure 10.14 : Output of Example 10.7

Observe that we are getting a new prompt along with a message "undefined reference to 'sqrt'" on screen. As we have used an inbuilt function sqrt() in our program we need to link the math library at the time of compilation. The following command will let us link the math library.

```
$ gcc -o 10_7.o 10_7.c -lm
```

Once we issue this command we will get a new prompt as our program did not have errors. Now at the prompt type ./10_7.o and observe the output. Figure 10.15 gives us the correct output.

```

harshal@harshal-Compaq-435-Notebook-PC: ~/Desktop/Chapter10
File Edit View Search Terminal Help
harshal@harshal-Compaq-435-Notebook-PC:~/Desktop/Chapter10$ gcc -o 10_7.o 10_7.c -lm
harshal@harshal-Compaq-435-Notebook-PC:~/Desktop/Chapter10$ ./10_7.o

Enter the value of number: 25

The square root of 25.000000 is 5.000000
harshal@harshal-Compaq-435-Notebook-PC:~/Desktop/Chapter10$ █

```

Figure 10.15 : Correct Output of Example 10.7

The method mentioned here is the most basic method for creating and executing a C program on Linux environment. Instead of using any normal text editor we will use an editor named SciTE to create and execute our program in this book. SciTE allows us to compile and run simple programs with one window itself. For programs with user inputs we may compile the program using SciTE, but will use terminal to execute the program. Let us try to see the usage of SciTE editor.

Open the SciTE Text Editor from appropriate location of your machine. We have already seen the look the SciTE editor window in previous examples. Type the contents of example 10.1 shown in figure 10.1 in the blank screen of the SciTE editor. Now save the file with the name 10_1.c. You can use CTRL + S or use **File → Save**. The window will now look similar to the one shown in figure 10.16.

```

10_1.c - SciTE
File Edit Search View Tools Options Language Buffers Help
1 10_1.c
/* Example 1: My first C program */

#include <stdio.h>
int main( )
{
    printf("Welcome to the world of C programming using Scite \n");
    return 0;
}

```

Figure 10.16 : Example 10.1 as written in SciTE window

Once the program has been written and saved we need to find syntax errors if present. To find syntax errors we need to compile the program. Press CTRL key along with F7 key (CTRL + F7) or select Tools → Compile from the menu. If no errors are there in the program then we will see a screen as shown in figure 10.17.

```

10_1.c - SciTE
File Edit Search View Tools Options Language Buffers Help
1 10_1.c
/* Example 1: My first C program */

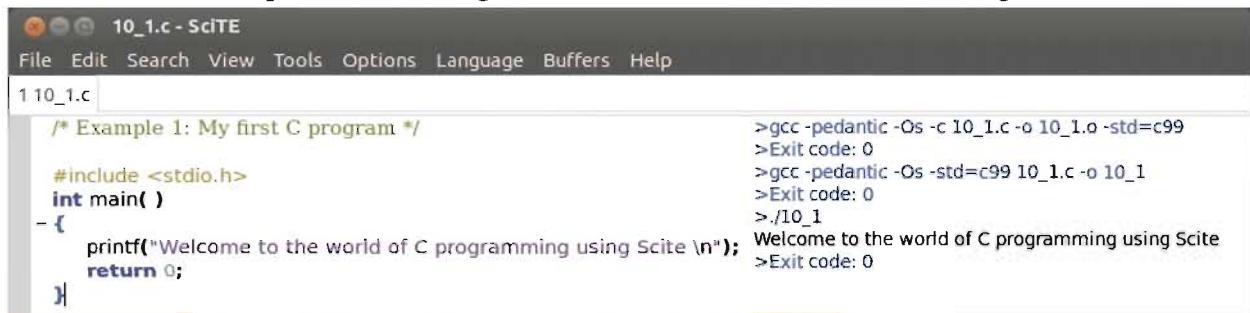
#include <stdio.h>
int main( )
{
    printf("Welcome to the world of C programming using Scite \n");
    return 0;
}

>gcc -pedantic -Os -c 10_1.c -o 10_1.o -std=c99
>Exit code: 0

```

Figure 10.17 : Successful compilation message

Now we need to execute this program. For this press the F5 key or select Tools → Go. We will now be able to see the output window along with source code window as shown in figure 10.18.



The screenshot shows the SciTE editor interface. The title bar says "10_1.c - SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, and Help. A toolbar with icons for file operations is visible. The left pane contains the C source code:

```
/* Example 1: My first C program */

#include <stdio.h>
int main( )
{
    printf("Welcome to the world of C programming using Scite \n");
    return 0;
}
```

The right pane shows the command-line output of the compilation process:

```
>gcc -pedantic -Os -c 10_1.c -o 10_1.o -std=c99
>Exit code: 0
>gcc -pedantic -Os -std=c99 10_1.c -o 10_1
>Exit code: 0
>/10_1
Welcome to the world of C programming using Scite
>Exit code: 0
```

Figure 10.18 : SciTE editor with two windows

Observe that we are able to see output of both the activities that we have performed. Observe the first output shown below that we had got when we just compiled the program.

```
> gcc -pedantic -Os 10_1.c -o 10_1.o -std=c99
> Exit code: 0
```

Here the file 10_1.o although an output file is not executable. Now observe the output shown below that we got when we used Go.

```
>gcc -pedantic -Os -std=c99 10_1.c -o 10_1
```

```
>Exit code: 0
```

```
>/10_1
```

Welcome to the world of C programming using scite

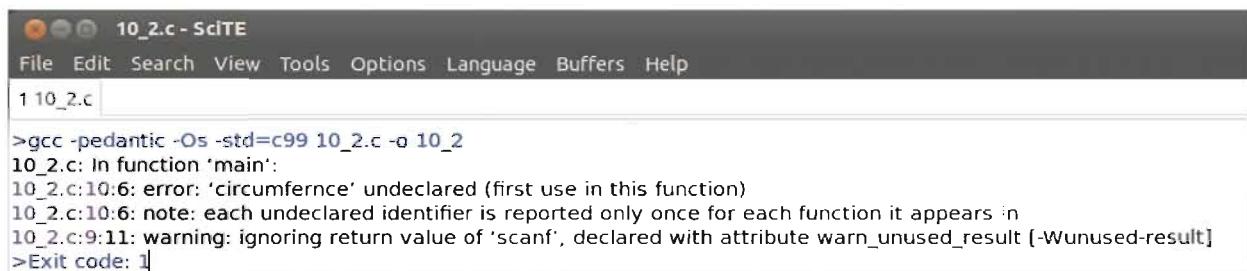
```
>Exit code: 0
```

Here the file was processed in two phases. First the file was compiled and an executable file with the name 10_1 was created. This operation is represented by first two lines. In the second phase the output file was executed using command ./10_1 (as shown in the third line). The last two lines are outcome of this execution.

Note: the text succeeding > symbol refers to actions taken by compiler, while the text shown without a preceding > symbol is output that user wants to display on screen.

It is also possible to enable and disable the output window by pressing F8 key or by selecting **View → Output**. To clear all the previous outputs press Shift + F5 or select **Tools → Clear Output**.

In this example we have taken a program that worked properly. Hence the process of compilation gave us no error. Let us now see what would happen if we have entered a wrong program. Assume that we have entered example 10.2 in the editor with a small change. Instead of writing the statement "circumference = 2 * PI * radius;" we did a typing mistake and wrote "circumfernce = 2 * PI * radius;". Now if we press CTRL + F7 or F5 we will get the screen as shown in figure 10.19.



The screenshot shows the SciTE editor interface. The title bar says "10_2.c - SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, and Help. The left pane contains the C source code:

```
>gcc -pedantic -Os -std=c99 10_2.c -o 10_2
10_2.c: In function 'main':
10_2.c:10:6: error: 'circumfernce' undeclared (first use in this function)
10_2.c:10:6: note: each undeclared identifier is reported only once for each function it appears in
10_2.c:9:11: warning: ignoring return value of 'scanf', declared with attribute warn_unused_result [-Wunused-result]
>Exit code: 1
```

Figure 10.19 : Compilation of program containing error

As complete image could not be shown properly we have shown only the output window. To see only output select Options → Vertical Split. Observe that we are able to see errors in figure 10.19. Observe the line "10_2.c:10:6: error: 'circumfernce' undeclared (first use in this function)" in the figure. It informs us that the variable with a name circumfernce is not declared in the program. Rectify the error and repeat the process of compilation. If no errors are found by the compiler, it will try to execute the program if we have pressed F5. Although we will not get any output in our case as we have used scanf() function. The Ubuntu version of SciTE seems to have a bug as of now as it does not wait for user inputs. We can execute the program using the name of the file without any extension from the terminal.

History of C

Having learnt how to program in C, let us now have some historical overview of C. The origin of C has been dated back to 1972 in Bell laboratories. The man who owns the credit of creating C language is Dennis M. Ritchie. It was derived from *Basic Combined Programming Language* commonly known as BCPL. The aim of developing C was to build robust system software. But it became a pet of programmers in coming years and has been used for developing all kind of software. Hence it has come to be known as general purpose programming language.

Although born in 1972, it was standardized in 1989 by American National Standards Institute (ANSI). From there on it came to be known as ANSI C. Different operating systems and compilers support this standard.

C is a structured language. It allows our program to be broken into small pieces known as functions. These functions once generated are reusable. Set of such functions then becomes a C program. Taking such an approach helps us to solve problem arising in the program easily, as we have to concentrate on only one function rather than an entire program. Usually programs written in C can be ported to different machines having different operating systems and compilers with almost negligible modification. Thus it is considered to be portable language. C is also considered to be a middle level language by some and higher level language by others. In any case it has all the features that a programmer would like to have.

Summary

In this chapter we learnt about the structure of C program. We looked at different components that can be part of C program. We then learnt about C character as mentioned can be divided into four categories namely letter, digits, white spaces and special characters. We saw how to use these characters and form valid words in C language. Then we looked at the steps of creating and executing a C program. We then learnt how to use gcc compiler to compile and execute our programs. Finally we learnt how to use the SciTE Text Editor to create and execute the C program.

Instruction for Teachers

The discussion of SciTE Text Editor has been made based on the assumptions that the editor is installed and its short cut would be available on the computer. Before starting to write C programs in your SciTE editor make sure you select Language -> C/C++ at least once. This will ensure the highlighting of C keywords when programming.

Also you may need to add the line mentioned below in properties file of SciTE to allow direct execution of C programs by using F5 or Tools → Go.

```
command.go.needs.*.c=gcc $(ccopts) -std=c99 $(FileNameExt) -o $(FileName)
```

The steps mentioned here will allow you to change the properties file.

1. Find location of cpp.properties file in your machine. (In our computer it is /usr/share/scite/cpp.properties)
2. Open terminal and type sudo gedit your_file_path/cpp.properties and press Enter key.
3. You will be prompted for administrator's password. Key in the correct password and press Enter key.
4. Now cpp.properties file will be opened in gedit window. Locate the code given in table 10.7 in the file.

```

ccopts=-pedantic -Os
cc=g++ $(ccopts) -c $(FileNameExt) -o $(FileName).o
ccc=gcc $(ccopts) -c $(FileNameExt) -o $(FileName).o

make.command=make
command.compile.*.c=$(ccc) -std=c99
command.build.*.c=$(make.command)
command.build.*.h=$(make.command)
#command.go.*.c=./a.out
command.go.*.c=./$(FileName)

```

Table 10.7 : Code to be searched in cpp.properties

The make sure that the last two lines look similar on your computers. If they are different, then change it as can be seen in table 10.9. This change makes sure that the executable (output) file will always be saved as the filename that you specify for your source code when using SciTE.

5. Add the following line to this file and save it.

```
# To make the Go command both compile (if needed) and execute, use this setting:
command.go.needs.*.c=gcc $(ccopts) -std=c99 $(FileNameExt) -o $(FileName)
```

6. Close the gedit and terminal windows. You are now ready to use the SciTE editor.

The screens given in this chapter are sample screens. These may differ based on the version of Ubuntu available in the school. The functionality of the screens though would remain same.

EXERCISE

1. List and explain the characteristics of a program.
2. Explain the significance of main() function.
3. What is the purpose of file include section in a C program ?
4. What is an identifier ? How is it useful in C program ?
5. What is variable ? State the rules for defining a variable.
6. Differentiate between single character constant and string constant.

7. State whether true or false :

- (a) The extension of C program is ".h".
- (b) Usually C statements end with a comma.
- (c) Amount is a valid variable name.
- (d) #define PI 3.24 includes a file named PI in C program.
- (e) "X" is a valid single character constant.

8. Choose the correct option from the following :

(1) Which of the following is an extension of C program file ?

- (a) c
- (b) h
- (c) s
- (d) t

(2) Which of the following number refers to number of C character categories?

- (a) 0
- (b) 2
- (c) 4
- (d) 8

(3) Which of the following C character categories does the symbol = belong ?

- (a) Letter
- (b) Blank Space
- (c) Special Character
- (d) Digit

(4) Which of the following is a valid keyword of C ?

- (a) ofsize
- (b) sizeof
- (c) forsize
- (d) sizefor

(5) Which of the following is an invalid variable name in C ?

- (a) Register
- (b) RegIster
- (c) registre
- (d) register

(6) Which of the following is an invalid integer constant in C ?

- (a) 0xG
- (b) 0xA
- (c) 0xB
- (d) 0xD

(7) Which of the following is valid real constant in C ?

- (a) -2.0.5e5
- (b) -20.5e5.5
- (c) -20.5e5
- (d) -20.5e.5

(8) Which of the following is valid single character constant in C ?

- (a) 'a'
- (b) '\a'
- (c) "a"
- (d) Both a and b

(9) The preprocessor directive `#define` is used to define which of the following in C ?

- (a) String constant
- (b) Symbolic constants
- (c) Integer constant
- (d) Single character constant

(10) Which of the following function key is used to directly execute a program ?

- (a) F7
- (b) F9
- (c) F5
- (d) F8

LABORATORY EXERCISE

1. Write a C program to display your name, school name, the standard you study in and school address in the center of the screen.

* Name : *

* School Name : *

* Standard : *

* Address : *

* *

2. Write a C program to print first letter of your surname on the screen. For example if the first letter of your surname is P then the output should be.

* *

*

*

3. Write a C program to print a greeting of your choice on the screen. For example if you want to wish someone on Diwali, print "Wishing you a happy and prosperous Diwali".





Data Types, Operators and Expression in C Language

In the previous chapter we saw some simple C programs. We also learnt about the character set and tokens of C language. One of the tokens discussed was identifier. We make use of identifiers in all C programs. Suppose in our program we have defined an identifier named "date", that is capable of storing value of date. What do you think is the value that can be stored in it? Can we assign it a value 12.50? The answer would be no. The correct value could be 12 or 13 or any valid positive two digit whole number between 1 and 31. This example suggests that, not only the identifier but the values stored in it are also important. C Language uses certain keywords to identify the type of value that can be stored in an identifier. These keywords associate a data with its types and hence are known as data types. In this chapter we will discuss in detail different data types available in C.

What is Data Type ?

The type of value that can be assigned to an identifier is known as its data type. If an identifier "date" has been assigned a value 12, then we would say that data type of "date" is integer. Similarly if an identifier "amount" has value 99.50, then we would say that data type of "amount" is real.

C language uses set of keywords to relate a data with its value. These keywords identify two things, the type of value that can be stored in an identifier and the memory space required by an identifier. Each data type is allocated a fixed memory space in C. It is denoted by bytes. 1 byte is combination of 8 bits.

Basic Data Types of C

C language supports basic data types called integer, decimal and character. These data types are represented using keywords int, float and char respectively. We also have one more primitive data type known as void. An identifier is associated with its data type by using the syntax:

Data type identifier 1 [, identifier 2, identifier 3, , identifier n];

The contents written in square brackets for the given syntax are optional. Let us now have a look at each of these basic data types one by one.

Integer

In the previous topic we mentioned that an identifier "date" can be assigned a value 12. Here 12 is an integer value. An integer is a positive or negative whole number with no fractional part in it. Some example of integer numbers are -99, 12, -10, 900, 30000 etc. To declare a variable of integer type we use the syntax:

int identifier 1 [, identifier 2, identifier 3, , identifier n];

Such a statement in C is called a declaration statement. Some examples of declaration statement are :

int roll_number;

int date, month, year;

The first statement declares an identifier with a name "roll_number" capable of storing an integer value. The second statement declares three identifiers "date", "month" and "year" each capable of storing integer values.

In ANSI C the data type int uses 4 bytes of memory space. It has a range of -2147483648 to +21474836411. All the variables defined till now are signed, i.e. they are capable of storing both positive as well as negative values. In many situations we may not require a negative value at all. None of the variables declared by us can have negative value. It is possible to allocate only positive values in C language. For this purpose we use a keyword unsigned. The examples now can be modified to

```
unsigned int roll_number;
```

```
unsigned int date, month, year;
```

These statements indicate that roll_number, date, month and year are identifiers of type integer and are capable of storing only positive value. Table 11.1 gives a brief description of integer data type.

Data Type	Range	Bytes required	Example
int	-2147483648 to +2147483647	4	int balance_amount;
unsigned int	0 to 4294967295	4	unsigned int counter;

Table 11.1 : Integer data type

In table 11.1 the value of range depends on the number of bytes used. For signed integers the value of range is calculated as $-2^{(n-1)}$ to $+2^{(n-1)} - 1$. Here "n" refers to number of bits required. Similarly for unsigned integers the range is calculated as 0 to $2^n - 1$. It is possible to increase the range of integer numbers by using a prefix long with the integer data type. Adding long as a prefix to int allocates 8 bytes of memory to the variable. We can define such a variable using the syntax shown below :

```
long int population;
```

Having seen different forms of integer data type available in C let us now see a sample program. Figure 11.1 shows the code listing of a program that illustrates the use of integer data type.

The screenshot shows the SciTE IDE interface with the title bar "11_1.c - SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, and Help. The current file tab is "11_1.c". The code area contains the following C program:

```

/* Example 1: Program to illustrate use of integer data type */

#include <stdio.h>
int main( )
{
    int mark = -10;
    unsigned int date = 30;
    long int population = 42949672950;
    printf(" Mark = %d", mark);
    printf("\n Date = %u", date);
    printf("\n Population = %ld\n\n", population);
    return 0;
}
/* End of Program */

```

Figure 11.1 : Program illustrating use of integer data type

Explanation

Let us now see what each of the line in figure 11.1 mean. The first statement after the opening curly brackets declares an integer variable called "mark". In the same statement we assign it value -10. The second statement declares positive integer variable called "date" and assigns it a value 30. The third statement declares an integer variable with higher range called "population" and assigns it value 42949672950. The next three statements prints the values assigned to these variables on the screen. Figure 11.2 shows the output of example 11.1.

The screenshot shows the SciTE IDE interface with the title bar "11_1.c - SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, and Help. The current file tab is "11_1.c". The code area contains the command to compile the program and the resulting output:

```

>gcc -pedantic -Os -std=c99 11_1.c -o 11_1
>Exit code: 0
>./11_1
Mark = -10
Date = 30
Population = 42949672950

>Exit code: 0

```

Figure 11.2 : Output of Example 11.1

Real

Many a times we need real numbers instead of integer numbers. For example amount to be paid by a person can have value 95 or 95.50. In such situations the data type *int* does not work. To use real values C language supports a data type identified by keyword *float*. It uses 4 bytes of storage space. Float values are precise to 6 and at times 7 decimal digits. If we need higher precision then we have to use keyword *double* rather than *float*. It is an extension of *float* data type. Double values use 8 bytes and are precise to 16 and at times 17 decimal digits. Examples of real variables are :

```
float amount_to_pay;
```

```
double balance_amount;
```

The first statement declares an identifier with a name "amount_to_pay" capable of storing float value. The second statement declares an identifier "balance_amount" capable of storing float values but having higher range and precision.

We can also use long as a prefix to double. This increases the range of double. Table 11.2 gives a brief description of float data type.

Data Type	Range	Significant Digits	Bytes required	Example
float	+/-3.4 e -38 to +/-3.4 e +38	6	4	float price;
double	+/-1.7 e -308 to +/-1.7 e +308	16	8	double tot_price;
long double	+/-3.4 e - 4932 to +/-1.1 e + 4932	16	16	long double credit;

Table 11.2 : Float data type

As can be seen in table 11.2 we have specified range in scientific form. This is so because C also supports scientific format. For example 95.50 could be represented as 0.9550 e 2. Here 0.9550 is known as mantissa while 2 is known as exponent. The float values can be negative or positive. The memory representation of float is given in figure 11.3.

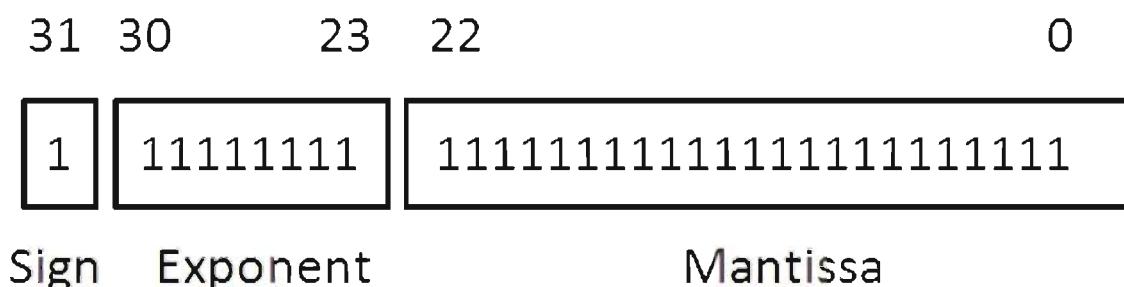


Figure 11.3 : Memory representation of float

Let us now see an example that shows the use of float, double and long double data type. Figure 11.4 shows the code listing of a program that illustrates the use of float data type.

The screenshot shows the SciTE IDE interface with the title bar "11_2.c - SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, and Help. The code editor window contains the following C program:

```
/* Example 2: Program to illustrate use of real data type */

#include <stdio.h>
int main( )
{
    /* First three statements declares and initializes of variables first, second and third */

    float first = 9876543210987654321.987654321;
    double second = 9876543210987654321.987654321;
    long double third = 9876543210987654321.987654321;

    printf(" First = %f", first);
    printf("\n Second = %lf", second);
    printf("\n Third = %Lf\n", third);
    return 0;
}
/* End of Program */
```

Figure 11.4 : Program illustrating use of float data type

Explanation

The first three statements after the opening curly brackets declare three variables "first", "second" and "third". Each of these variables is assigned a float value of 9876543210987654321.987654321. The next three statements prints the values assigned to these variables on the screen. Figure 11.5 shows the output of example 11.2.

The screenshot shows a terminal window with the title bar "11_2.c - SciTE". The command line shows the compilation and execution of the program:

```
>gcc -pedantic -Os -std=c99 11_2.c -o 11_2
>Exit code: 0
>/11_2
First = 9876543516404875264.000000
Second = 9876543210987655168.000000
Third = 9876543210987655168.000000

>Exit code: 0
```

Figure 11.5 : Output of Example 11.2

Observe that the values of variable first, second and third are 9876543516404875264.000000, 9876543210987655168.000000 and 9876543210987655168.000000 respectively. Looking at the output properly we can see only first 7 digits of variable "first" are matching the actual entry. Similarly in variables "second" and "third" only first 15 digits are matching. This is the reason we have mentioned that the float is precise to 6 and at times 7 decimal digits. Similarly double and long float are precise to 15 and at times 16 decimal digits.

Floating point numbers when used at times lead to loss of precision. We may not get an exact result as expected. Hence to get better accuracy of results it is advisable to use higher precision. The benefit of using float values is that it can represent much broader range on values compared to integer.

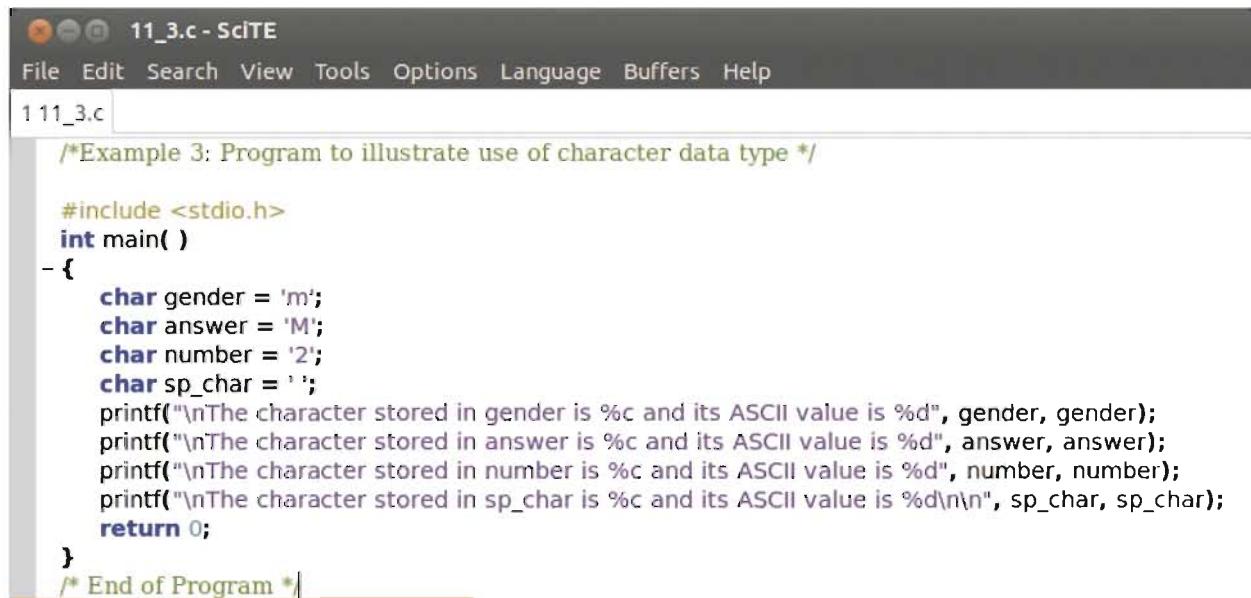
Character

In both the above cases we have stored numbers. What happens if we need to store a name of city or gender of a person say 'M' for male and 'F' for female. Here we only have alphabets. Such values cannot be stored by using int or float. To store such values we have to use a keyword char. It requires 1 byte of memory space. Each character is associated with an integer value called ASCII (American Standard Code for Information Interchange) value. Appendix II gives details of all ASCII values. By default char is unsigned. It is also possible to have signed char. Table 11.3 gives a brief description of character data type.

Data Type	Range	Bytes required	Example
char	- 128 to + 127	1	char gender;
unsigned char	0 to 255	1	unsigned char choice;

Table 11.3 : Character Data type

Figure 11.6 shows the code listing of a program that illustrates the use of character data type.



```

11_3.c - SciTE
File Edit Search View Tools Options Language Buffers Help
11_3.c
/*Example 3: Program to illustrate use of character data type */

#include <stdio.h>
int main()
{
    char gender = 'm';
    char answer = 'M';
    char number = '2';
    char sp_char = ' ';
    printf("\nThe character stored in gender is %c and its ASCII value is %d", gender, gender);
    printf("\nThe character stored in answer is %c and its ASCII value is %d", answer, answer);
    printf("\nThe character stored in number is %c and its ASCII value is %d", number, number);
    printf("\nThe character stored in sp_char is %c and its ASCII value is %d\n\n", sp_char, sp_char);
    return 0;
}
/* End of Program */

```

Figure 11.6 : Program illustrating use of char data type

Explanation

The first four statements after the opening curly brackets declare four variables "gender", "answer", "number" and "sp_char" and are assigned values 'm', 'M', '2' and ' ' (blank space) respectively. The next four statements prints the values assigned to these variables along with their ASCII values on the screen. Figure 11.7 shows the output of example 11.3.

The screenshot shows the SciTE IDE interface with the title bar "11_3.c - SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, and Help. A tab labeled "11_3.c" is selected. The code area contains the following command-line output:
>gcc -pedantic -Os -std=c99 11_3.c -o 11_3
>Exit code: 0
>./11_3

The program's output is displayed below the command line:
The character stored in gender is m and its ASCII value is 109
The character stored in answer is M and its ASCII value is 77
The character stored in number is 2 and its ASCII value is 50
The character stored in sp_char is and its ASCII value is 32

>Exit code: 0

Figure 11.7 : Output of Example 11.3

Note the difference in the ASCII values of small m and capital M. We can see that it is also possible to use a number in character form. To use numeric value 2 as character we have to write it as '2'. Also observe that though the character assigned to sp_char is not visible on the screen we still do see its ASCII value.

Empty Data Set

C provides a special data type identified by keyword void. This data type has no value; hence we denote it to be empty. It is used mainly as a return type of functions. As mentioned in previous chapter C program consist of function. Functions in C may return a value. If we don't want the function to return a value then we can prefix it with void.

As seen in previous examples the last statement of all programs is return 0;. This statement tells the compiler to exit from the function body. If we don't write this statement then we will get a warning message "Function should return a value". To avoid getting such a message we can prefix main() with void.

Assigning Values to Variables

Variables once declared can be assigned values during the execution of the program. The value assigned has to be in lines of the specification of data type. To assign a value we use the given syntax:

Variable = value;

It is also possible to initialize and declare a variable in same statement. To do so we use syntax:

Data type Variable = value;

In all the examples discussed so far in this chapter we have used this technique of initialization.

User Defined Data Type

C is a flexible language. It allows user to create new entities from existing one. We can create a new data type by making use of existing data type. Two keywords `typedef` and `enum` are used in C for generating user defined data type. Both these keywords define a variable that can be used as data type in our program.

Type Definition

The syntax for using type definition is:

```
typedef datatype variable;
```

Here data type refers to an existing data type for example int or float. Variable refers to new name that we want to assign to the available data type. In real sense C does not allow us to create a new data type. Instead it allows us to give an alias to an existing data type. In normal life also we use aliases. One example would be pet name of a person. Some examples of typedef are:

```
typedef char alpha;
```

```
typedef double twice;
```

The first statement gives an alias "alpha" to character data type, while second statement gives an alias "twice" to double data type. After providing an alias, to declare a character variable called "choice" and double variable called "amount" we can use the given statements:

```
alpha choice;
```

```
twice amount;
```

Enumerated data type

Enumerated data types can be created in C by using keyword enum. The syntax for using enum is :

```
enum identifier {value 1, value 2, value 3, ..... , value n};
```

Here enum is the keyword. Identifier is the name that can be used to define a new variable. Value 1 to Value n refers to names of numeric constants 0, 1, 2,etc. The variable defined using enum can be assigned any numeric value or values given in curly brackets. An example of enum would be

```
enum money {rupee, dollar, pound, yen};
```

```
enum money currency;
```

```
currency = dollar;
```

The first statement defines enumerated data type "money". The next statement declares a variable "currency" of type money. The last statement assigns value dollar to currency and is equivalent to currency = 1. Here rupee, dollar, pound and yen refer to numeric constants 0, 1, 2 and 3 respectively. These numeric constants are assigned by the compiler automatically. It is possible to modify these values by assigning new values. For example if we want rupee to be equivalent to 10, dollar to be equivalent to 50, pound to be equivalent to 75 and yen to be equivalent to 100 then we may rewrite the first statement as: enum money {rupee = 10, dollar = 50, pound = 75, yen = 100}; Now the statement currency = dollar would be equivalent to currency = 50.

User defined data types improve the readability of C programs to some extent. It also allows user to define data types with meaningful names.

Derived Data Type

As seen in the previous topics, C is an extensible language. The basic data types may suffice our needs in many cases. But it may not be appropriate to use them due to complexity involved. Say for example we need to store grades of 15 students. In such cases we need to define 15 variables.

Handling 15 variables may increase the complexity of the program. Similarly if we need to store complete information of 15 students like their roll number, gender, name and address then we may need additional variables. Grouping of the variables may become a problem here. To overcome these problems C provides us derived data types. Array, structure, union and pointers are examples of derived data types.

Array

It is possible to create a data structure called "array" in C. This data structure consists of group of variables having same property. The syntax for defining an array is:

```
data type variable[size];
```

Here data type can be any basic data type, user defined data type or derived data type. Variable is name of the array and size refers to number of elements in the array. For example C statement **char name_of_subject[10];** would define an array called name_of_subject having size 10. This means that it is possible to store group of 10 characters and identify it by using variable "name_of_subject". The data type of each element here would be character.

It is also possible to have a statement like **char name_of_subject[] = "C Language";** in C. The compiler will automatically decide length of the array. In the given case the length would be 11. Arrays are discussed in detail in chapter 15.

We also have some more derived data types like structure, union and pointers. The discussion of all these data types is at present out of scope of this book.

Operators and Expression

Till now we learnt about the data types available in C. Let us now learn about different operators available in C. We will also learn how to make use of the operators and operands to construct an expression. Finally we will see how expressions are evaluated.

Operators

When learning basic mathematics in school we have performed operations like addition of two numbers, subtraction of two numbers. For example $5 + 3$ and $9 - 7$, here 5, 3, 9 and 7 are constants while '+' and '-' are symbols that identify the operations to be performed on the constants.

An operator is a symbol that identifies the operation that can be performed on operands. The operators in C can be categorized into eight types as mentioned :

1. Arithmetic Operators.
2. Assignment Operators.
3. Relational Operators.
4. Increment and Decrement Operators.
5. Conditional Operators.
6. Logical Operators.
7. Bitwise Operators.
8. Special Operators.

Let us now discuss these operators in detail.

Arithmetic Operators

To perform an arithmetic operation C provides various basic operators like '+', '-', '*', '/', '%'. An operator for exponent is not available in C. Table 11.4 lists the operators and its use.

Operator	Use
+	Addition of two operands or Unary Plus.
-	Subtraction of two operands or Unary Minus.
*	Multiplication of two operands.
/	Division of two operands resulting in Quotient.
%	Division of two operands resulting in Remainder.

Table 11.4 : Arithmetic Operators

We are familiar with the first four operators given in table 11.4. C provides us an additional operator '%' known as modulo operator. It is used to get a remainder when performing division of two integer operands. Note that it cannot be used with float operands.

Let us now see how we can use the operators. Consider an example of C statement

```
total_cost = quantity * cost_item;
```

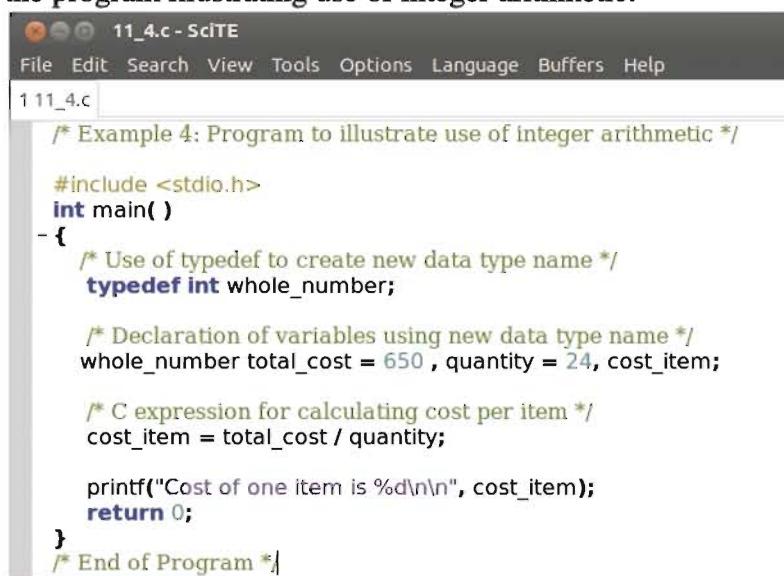
This C statement forms a C expression, here total_cost, quantity and cost_item are known as C variables. As can be seen the expression consists of two operators and three operands. This statement when executed performs two operations. First, it multiplies quantity and cost_item and second, it stores the result of multiplication in total_cost.

Based on the operand used in the expression, the arithmetic can be categorized as integer arithmetic, real arithmetic or mixed mode arithmetic.

Integer Arithmetic

Arithmetic is considered to be integer arithmetic when the operands used in the expression are positive or negative whole numbers. The expression here is called integer expression. The result of integer arithmetic is always integer.

Let us write a program to find cost per item if we are given the total cost and number of items. Figure 11.8 shows the program illustrating use of integer arithmetic.



The screenshot shows a SciTE window titled "11_4.c - SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, and Help. The code area contains the following C program:

```
/* Example 4: Program to illustrate use of integer arithmetic */

#include <stdio.h>
int main()
{
    /* Use of typedef to create new data type name */
    typedef int whole_number;

    /* Declaration of variables using new data type name */
    whole_number total_cost = 650, quantity = 24, cost_item;

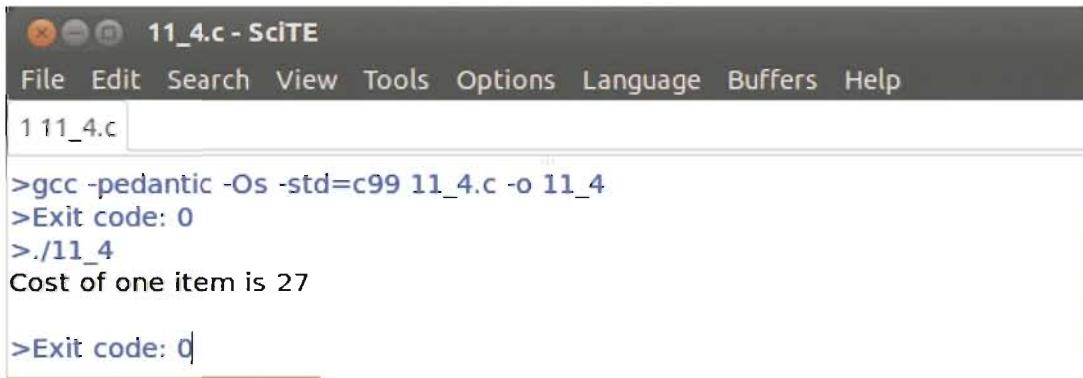
    /* C expression for calculating cost per item */
    cost_item = total_cost / quantity;

    printf("Cost of one item is %d\n\n", cost_item);
    return 0;
}
/* End of Program */
```

Figure 11.8 : Program illustrating use of integer arithmetic

Explanation

Here the first statement uses `typedef` keyword to create an alias for `int`. Then three integer variables "`total_cost`", "`quantity`" and "`cost_item`" are defined using the new alias. We also assign value 650 to "`total_cost`" and 24 to "`quantity`". Next statement divides "`total_cost`" by "`quantity`" and assigns the result to "`cost_item`". Then we print a message and value of "`cost_item`". Figure 11.9 shows the output of example 11.4.



The screenshot shows the SciTE IDE interface with the title bar "11_4.c - SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, and Help. The code editor window contains the following text:

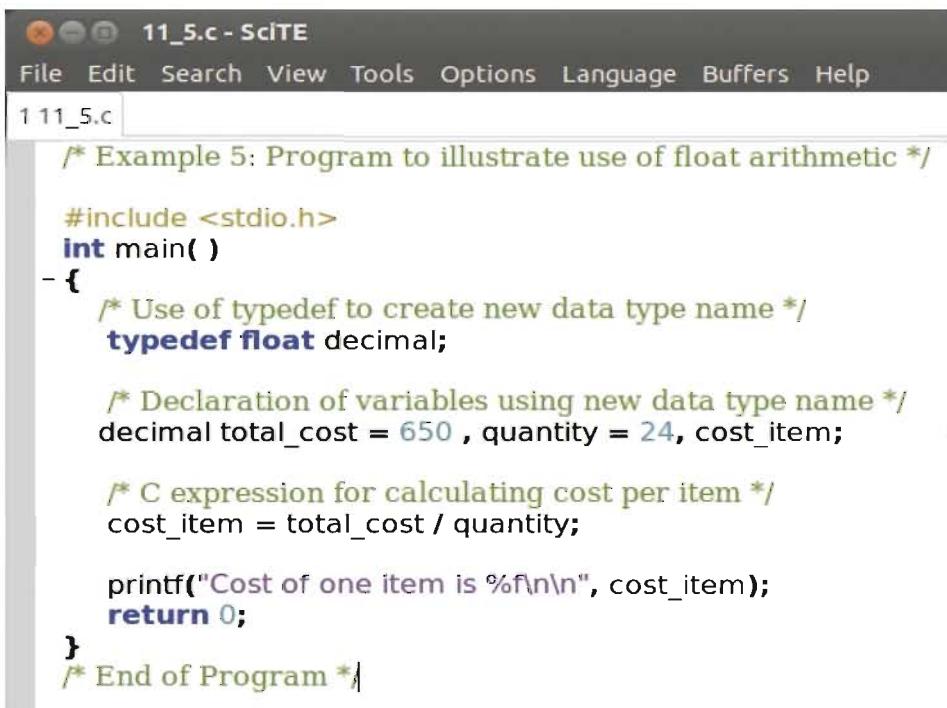
```
>gcc -pedantic -Os -std=c99 11_4.c -o 11_4
>Exit code: 0
>./11_4
Cost of one item is 27
>Exit code: 0
```

Figure 11.9 : Output of Example 11.4

As per the conventional mathematics $650/24$ should give us 27.083333333. But we can see in the output that the value of "`cost_item`" is 27. From this we can say that the output of integer arithmetic will always be integer. In C the division operator works differently from conventional mathematics. In conventional mathematics $5/10$ would give us 0.5, here it would give 0.

Real Arithmetic

If all the operands used in an expression are float the expression is called real expression. The result of real arithmetic is always represented in decimal values. Let us modify the program given in figure 11.8 such that we have real arithmetic instead of integer arithmetic. Figure 11.10 shows the modified program while its output is shown in figure 11.11.



The screenshot shows the SciTE IDE interface with the title bar "11_5.c - SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, and Help. The code editor window contains the following C code:

```
/* Example 5: Program to illustrate use of float arithmetic */

#include <stdio.h>
int main( )
{
    /* Use of typedef to create new data type name */
    typedef float decimal;

    /* Declaration of variables using new data type name */
    decimal total_cost = 650, quantity = 24, cost_item;

    /* C expression for calculating cost per item */
    cost_item = total_cost / quantity;

    printf("Cost of one item is %f\n", cost_item);
    return 0;
}
/* End of Program */
```

Figure 11.10 : Program illustrating use of real arithmetic

The screenshot shows the SciTE IDE interface. The title bar says "11_5.c - SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, and Help. A tab labeled "11_5.c" is selected. The code area contains the following text:

```
>gcc -pedantic -Os -std=c99 11_5.c -o 11_5
>Exit code: 0
>./11_5
Cost of one item is 27.083334

>Exit code: 0
```

Figure 11.11 : Output of Example 11.5

Explanation

This example is similar to example 11.4 except for one thing. Here the variables are declared as float. Although the values given to "total_cost" and "quantity" are integers but internally they are stored as float. The output is also a float value.

Mixed Mode Arithmetic

In this mode of arithmetic we are able to use integer as well as float operands within an expression. The output will depend upon the data type of the variable to which value is assigned. Although an expression like `result = 25.75 * 5;` is allowed in C. It is not directly possible to evaluate it. An expression can only be evaluated if all the operands used in it are of same data type. C as it allows mixed mode arithmetic takes care of this problem. C allows automatic conversion of lower range data type to higher range data type.

Assignment Operators

In examples discussed till now we have used statements like `total_cost = quantity * cost_item;` or `date = 30;` here the symbol "=" is known as assignment operator. As can be seen it is used to assign a constant value or result of an expression to a variable.

C also provides an operator known as short hand operator. It is obtained by prefixing arithmetic operators to "=" . The syntax for its use is:

variable op= constant value; or variable op= expression;

Here "op" refers to an arithmetic operator. While "op=" is known as short hand operator. Some examples of use of short hand operators are as given:

`first -= 1;` This statement is equivalent to `first = first - 1;`

`first += 3;` This statement is equivalent to `first = first + 3;`

`first *= (second + third);` This statement is equivalent to `first = first * (second + third);`

As can be seen the short hand operator provides us with ease of use. Though easy it requires certain amount of practice to use it efficiently.

Relational Operators

The relational operators allow us to compare two similar types of operands and generally are used to change the flow of execution of program. Say for example to give some discount to a customer we may check for the amount of purchase he/she has made. If it exceeds certain predefined amount then we may give discount to the customer. This example may be represented in C as if
(*total_purchase* > 10000) *discount* = 500;

C provides us with six relational operators for the purpose of comparison. Table 11.5 lists relational operators and their meanings.

Operator	Use
<code>= =</code>	Check equality between two operands.
<code>!=</code>	Check non equality between two operands.
<code>></code>	Check for greater value between two operands.
<code><</code>	Check for smaller value between two operands.
<code>>=</code>	Check for greater value or equality between two operands.
<code><=</code>	Check for smaller value or equality between two operands.

Table 11.5 : Relational Operators

Observe that to check equality of two operands we have use "`==`" instead of "`=`", as "`=`" is used as an assignment operator in C. The syntax for using relational operators is:

expression-1 Rop expression-2

Here "Rop" refers to a relational operator; expression-1 and expression-2 can be an arithmetic expression, variable or constant. Relational operators are used along with decision structures like "if" and control structures like "for", "while" and "do..while". These structures have been discussed in detail in chapter 13 and 14. you will find many programs in this book that make use of relational operators.

Increment and Decrement Operators

At times we need to increase or decrease the value of a variable by 1. Such situations arise when set of statements are to be repeated. Here we need to use a variable that keeps track of the number of times a loop has been executed. We can achieve this objective by using short hand operator. For example we can use the syntax variable `+ = 1` or variable `- = 1` for incrementing or decrementing.

C provides two special unary operators "`++`" and "`--`" for this purpose. These operators require only one operand. "`++`" is known as increment operator while "`--`" is known as decrement operator. The syntax for using these operators is:

`++variable` or `variable++`

`--variable` or `variable--`

Here "`++`" and "`--`" are used as either prefix or suffix to a variable. When it is prefixed we call it pre increment or pre decrement operator. On the other hand when it is suffixed we call it post increment or post decrement operator.

The result of this operation though is increment or decrement in variable value by 1. The effect of this increment and decrement depends upon the way it is used in program.

Suppose we have statements like

```
int first = 15, second = 20, result;  
result = first + second++;
```

Then here the value of "result" would be 35 and not 36. While the value of variable "second" at the end of statement two, would be 21. A value 1 will be added to its old value 20.

Note that post increment operator, when used in an expression uses the old value of a variable to evaluate the expression, then it increments the value of variable by 1.

Similarly if we have statements

```
int first = 15, second = 20, result;  
result = first + ++second;
```

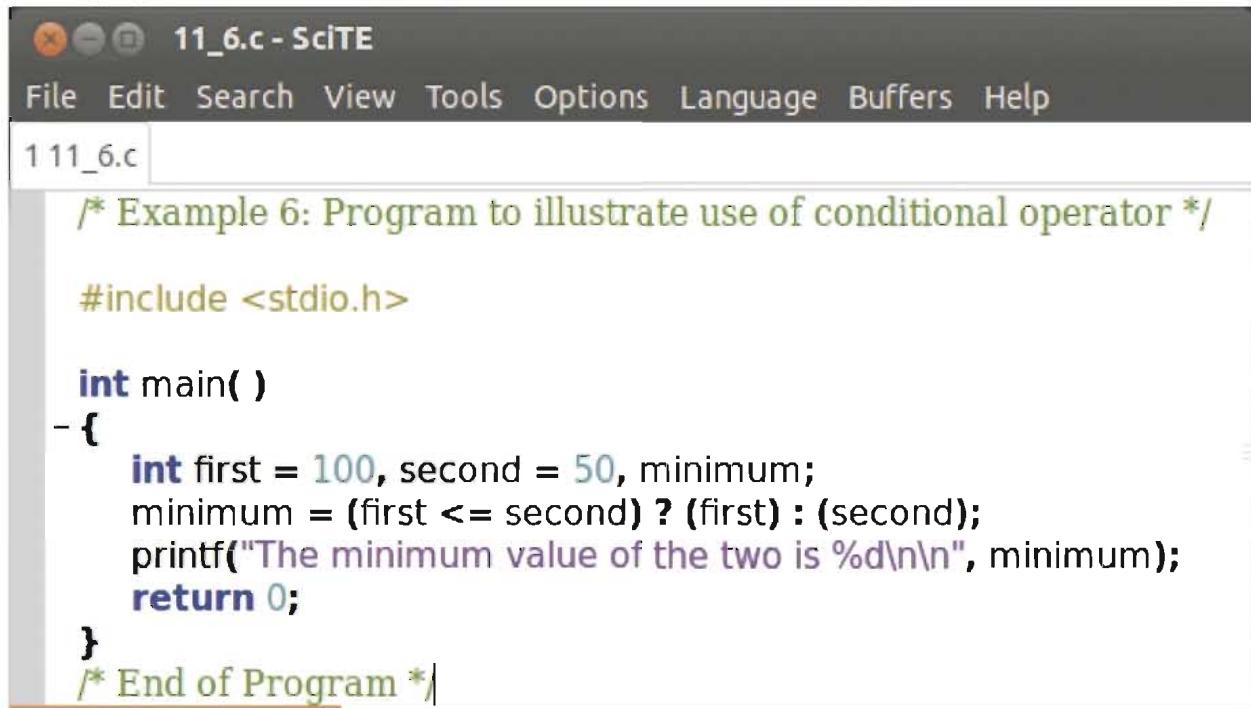
Then the result value of "result" would be 36. While the value of variable "second" at the end of statement two, would be 21. *Note that pre increment operator, when used in an expression, first increments the value of variable by 1 then uses the new value of a variable to evaluate the expression.*

Conditional Operators

C provides us an operator known as ternary operator or conditional operator for condition checking. This operator is identified by combination of two symbols "? :" The syntax for using conditional operator is:

(condition) ? (True statement) : (False statement);

The condition here refers to a relational expression like `first < second`. Let us see how to use conditional operator by making a program to find minimum of two numbers. Figure 11.12 gives the code listing of the program.



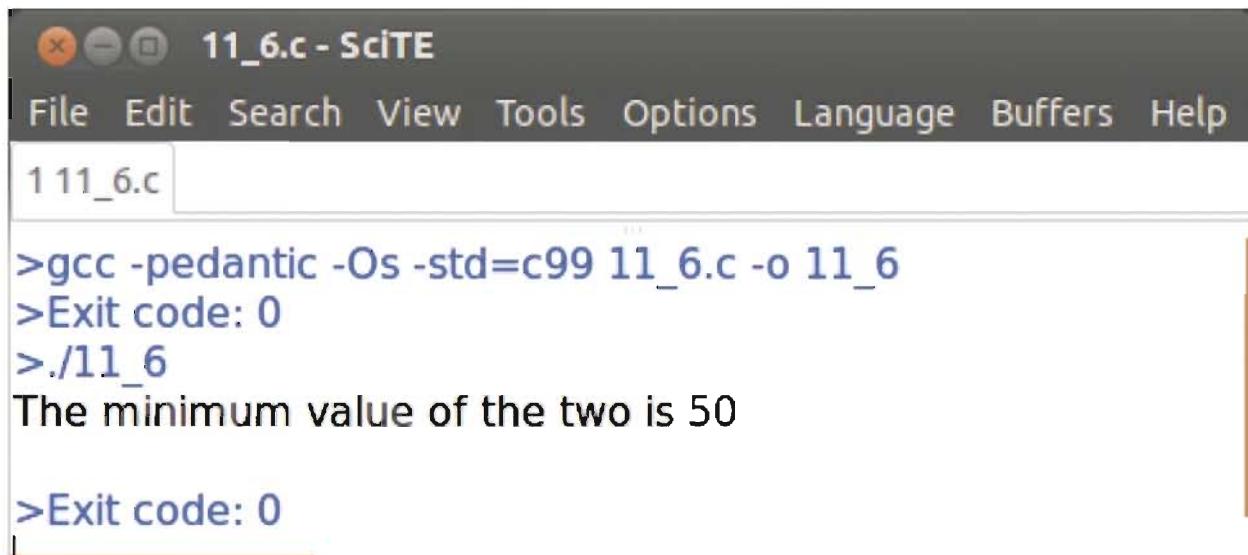
The screenshot shows a window titled "11_6.c - SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, and Help. The title bar shows the file name "11_6.c". The code area contains the following C program:

```
/* Example 6: Program to illustrate use of conditional operator */  
  
#include <stdio.h>  
  
int main()  
{  
    int first = 100, second = 50, minimum;  
    minimum = (first <= second) ? (first) : (second);  
    printf("The minimum value of the two is %d\n\n", minimum);  
    return 0;  
}  
/* End of Program */
```

Figure 11.12 : Program illustrating use of conditional operator

Explanation

Here the first statement of the program defines three integer variables and initializes two of them. It also assigns values 100 and 50 to first and second respectively. Then we make use of conditional operator to check whether first is less than or equal to second. If the condition is true then minimum is assigned value of first, otherwise it is assigned value of second. We then print the message and value of minimum on screen. Finally we exit the program. The output of example 11.6 is shown in figure 11.13.



The screenshot shows the SciTE IDE interface. The title bar says "11_6.c - SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, and Help. A tab labeled "11_6.c" is selected. The code area contains the following text:

```
>gcc -pedantic -Os -std=c99 11_6.c -o 11_6
>Exit code: 0
>./11_6
The minimum value of the two is 50
>Exit code: 0
```

Figure 11.13 : Output of Example 11.6

Logical Operators

At times we may need to evaluate multiple conditions together to give some output. While at times we may get an output even if one out of the given set of conditions is satisfied. Such a relation is generally known as logical relation. For example consider a case where a student gives exam of two subjects. The student now will be declared as passed, if and only if he or she scores 35 or more marks in both the subjects. Here the relationship can be expressed as If (Marks in subject 1 \geq 35) and (Marks in subject 2 \geq 35) then pass else fail.

To represent such a relation, C provides logical operators. Table 11.6 gives the list and use of these logical operators.

Operator	Use
<code>&&</code>	Logical AND
<code> </code>	Logical OR
<code>!</code>	Logical NOT

Table 11.6 : Logical Operators

Logical AND is used when all the given conditions must be satisfied. Logical OR is used when any one of the given conditions must be satisfied. Logical expression gives as an output values either 0 or 1. Here 0 refers to false and 1 refers to true. Table 11.7 gives the output of different logical operations.

Expression 1	Expression 2	! (Expression 2)	Expression 1 && Expression 2	Expression 1 Expression 2
0	0	1	0	0
0	1	1	0	1
1	0	0	0	1
1	1	0	1	1

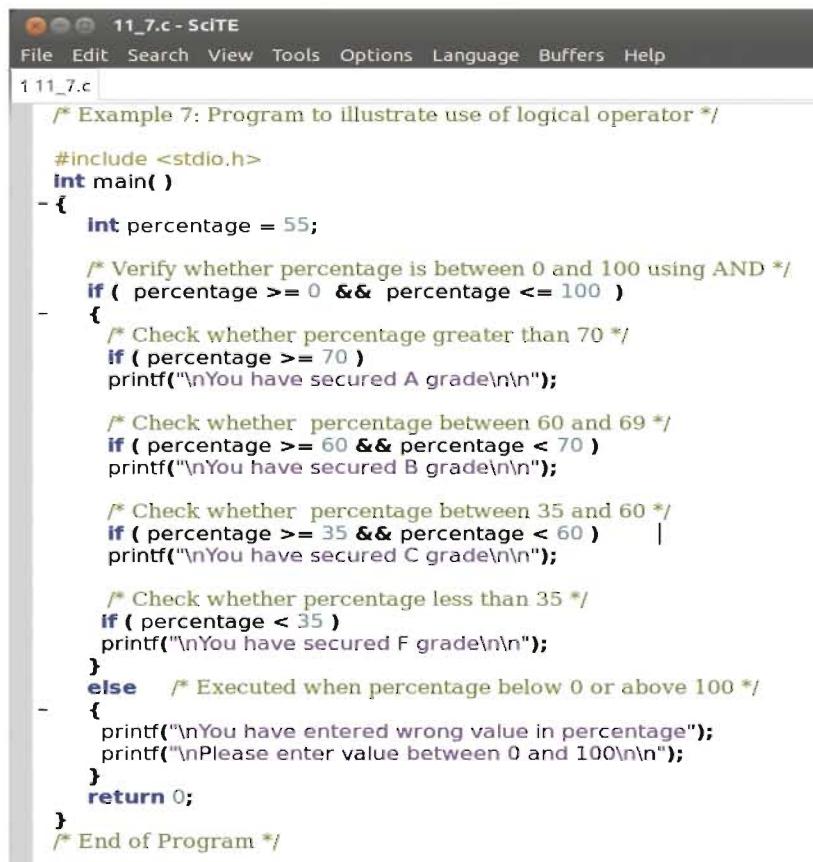
Table 11.7 : Truth table (Results) for logical operations

As can be seen AND operator returns 1 only if both expressions return 1. While OR operator returns 1 when any of the expressions return 1.

Let us write a program that tells us the grade of a student based on the criteria as mentioned:

Grade	Marks
A	If percentage greater than or equal to 70
B	If percentage between 60 and 69
C	If percentage between 35 and 59
F	If percentage less than 35

Figure 11.14 gives the code listing of the program while figure 11.15 shows its output.



```

11.7.c - SciTE
File Edit Search View Tools Options Language Buffers Help
11.7.c
/* Example 7: Program to illustrate use of logical operator */

#include <stdio.h>
int main()
{
    int percentage = 55;

    /* Verify whether percentage is between 0 and 100 using AND */
    if ( percentage >= 0 && percentage <= 100 )
    {
        /* Check whether percentage greater than 70 */
        if ( percentage >= 70 )
            printf("\nYou have secured A grade\n\n");

        /* Check whether percentage between 60 and 69 */
        if ( percentage >= 60 && percentage < 70 )
            printf("\nYou have secured B grade\n\n");

        /* Check whether percentage between 35 and 60 */
        if ( percentage >= 35 && percentage < 60 )
            printf("\nYou have secured C grade\n\n");

        /* Check whether percentage less than 35 */
        if ( percentage < 35 )
            printf("\nYou have secured F grade\n\n");
    }
    else /* Executed when percentage below 0 or above 100 */
    {
        printf("\nYou have entered wrong value in percentage");
        printf("\nPlease enter value between 0 and 100\n\n");
    }
    return 0;
}
/* End of Program */

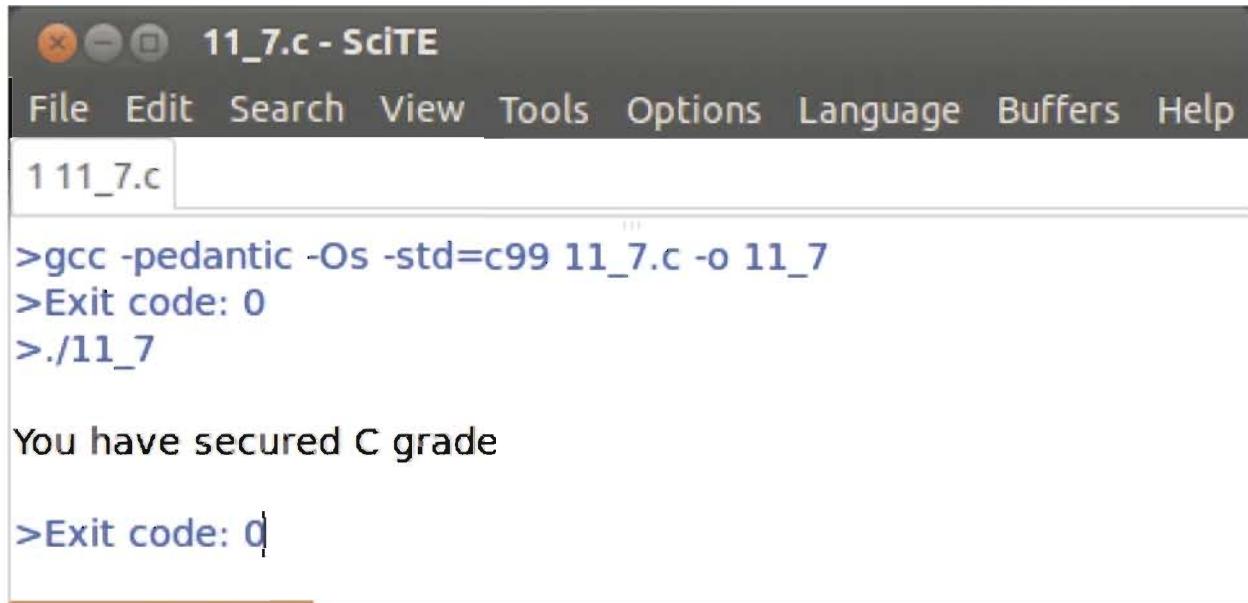
```

Figure 11.14 : Program illustrating use of logical operator

Explanation

Here the first statement declares and initializes an int variable called "percentage". Initially we check whether the value of "percentage" is between 0 and 100. If the answer is no then we move on to else part and print messages "You have entered wrong value in percentage", "Please enter value between 0 and 100" and then exit the program.

If the answer to the comparison is true, we then compare the value of percentage using if statement and logical conditions. The message specified in the if statement that returns true for the checked criteria is printed on screen see figure 11.15.



The screenshot shows the SciTE IDE interface. The title bar says "11_7.c - SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, and Help. A file named "11_7.c" is open in the editor. The terminal window below shows the command-line output:

```
>gcc -pedantic -Os -std=c99 11_7.c -o 11_7
>Exit code: 0
>/11_7

You have secured C grade

>Exit code: 0
```

Figure 11.15 : Output of Example 11.7

Bitwise Operators

We have mentioned in this chapter that data is stored in the form of bits within the memory location. C allows us to operate directly at bit level using bitwise operators. Table 11.8 lists the bitwise operators available in C and their use.

Operator	Use
&	Bitwise AND
	Bitwise OR
~	Bitwise NOT
^	Bitwise Exclusive OR
<<	Left Shift by number of bits specified.
>>	Right Shift by number of bits specified.

Table 11.8 : Bitwise Operators

To perform bitwise AND, OR, Exclusive OR (XOR), Left shift and Right shift we require two operands. NOT on the other hand only requires one operand. The value that the operands can hold is 0 and 1. Detailed discussion of these operators is out of scope of this book.

Special Operators

C provides special operators like sizeof(), ",", ".", "→", "&" and "*". In this section we will discuss only "," and sizeof() operators. The "," operator is used on many occasions. The comma operator is also used in decision structures and control structures discussed later chapters.

The sizeof() operator is a special operator used to return size of bytes required to store an entity. For example size = sizeof(int) will assign value "4" to "size". This is so because data type int uses 4 bytes of memory space.

Figure 11.16 gives the code listing, while figure 11.17 shows the output of the program that makes use of comma and sizeof operator.



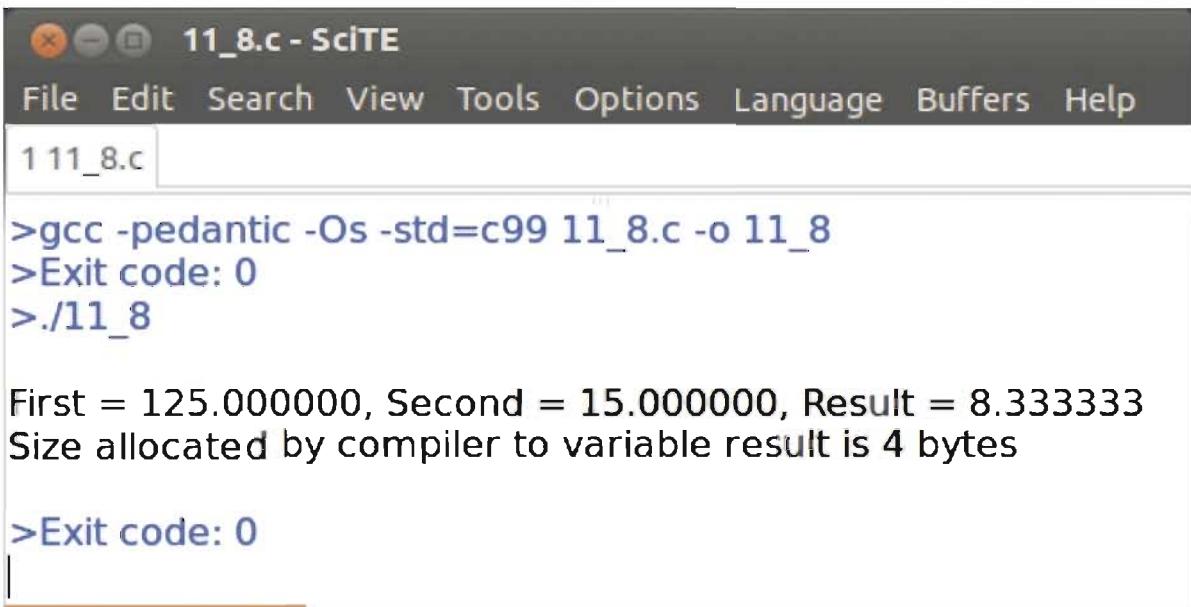
```
11_8.c - SciTE
File Edit Search View Tools Options Language Buffers Help
11_8.c
/* Example 8: Program to illustrate use of comma and size of operator */

#include <stdio.h>
int main( )
{
    float first, second, result;
    int size;
    /* Use of comma operator */
    result = (first = 125, second = 15, first / second);
    printf("\nFirst = %f, Second = %f, Result = %f", first, second, result);

    /* Use of sizeof operator */
    size = sizeof(result);
    printf("\nSize allocated by compiler to variable result is %d bytes\n\n", size);

}
/* End of Program */
```

Figure 11.16 : Program illustrating use of special operator



```
11_8.c - SciTE
File Edit Search View Tools Options Language Buffers Help
11_8.c
>gcc -pedantic -Os -std=c99 11_8.c -o 11_8
>Exit code: 0
>./11_8

First = 125.000000, Second = 15.000000, Result = 8.333333
Size allocated by compiler to variable result is 4 bytes

>Exit code: 0
```

Figure 11.17 : Output of Example 11.8

Explanation

Here the first statement declares three float variables. Second statement declares an integer variable. The third executable statement is an expression that uses comma operator. The statement works as mentioned. The RHS is evaluated first. The first operation is assignment of 125 to "first", second operation is assignment of 15 to "second", the third operation is division of "first" by "second". Finally the value of division is assigned to "result". Next statement prints the value of "first", "second" and "result". Then we have assigned size with the value of number of bytes allocated to the variable result.

Evaluation of Expression

Until now we learnt the operators available in C and how to use them in expression. Let us now see how the expression would be evaluated. Consider the expression in statement

*result = first + second * third - fourth;*

Here first multiplication of second and third would be done. Its value of multiplication would then be added to first. And from this intermediate value fourth would be subtracted. If two operators with same priority appear in an expression, then they will be evaluated from left to right. To modify this sequence we can make use of brackets.

*result = first + second * (third - fourth) / fifth ;*

Here third - fourth will be evaluated first, the intermediate result will be multiplied with second then division with fifth will be done and finally addition with first will be done. See figure 11.18.

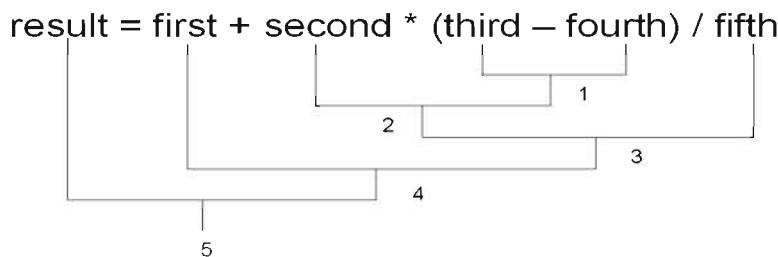


Figure 11.18 : Evaluation order of expression

Priority of Operators

The operators mentioned so far have their own priorities. As seen in the example *result = first + second * (third - fourth) / fifth*; * was at higher priority than +, - or /. C language designers have pre decided these priority. The listing of some of which is given in table 11.9. The detail is available in appendix III.

Operator	Operation Used for	Associativity	Precedence
()	Function call	Left to Right	First
[]	Array expression		
++	Increment	Right to Left	Second
--	Decrement		
sizeof()	Size of operand		
*	Multiplication	Left to Right	Third
/	Division		
%	Modulo division		
+	Binary addition	Left to Right	Fourth
-	Binary subtraction		

Table 11.9 : Summary of C operators

Type Conversion

An expression can be evaluated only if all the operands involved are of same data type. This property of an expression requires internal conversion of data types to be performed. For example if we have following code snippet

```
int year=2;  
float principal, rate, interest;  
interest = (principal * rate * year) / 100;
```

Then the evaluation of the expression shown above would be done as shown in figure 11.19.

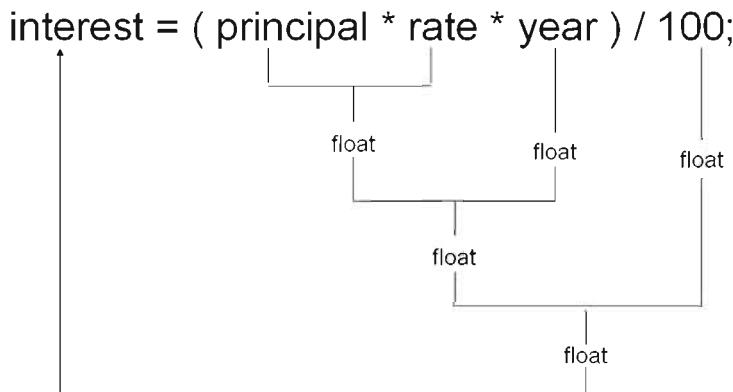


Figure 11.19 : Type Conversion

It is possible to override the internal conversion by performing a process called type casting. To type cast any value we use the syntax

(data type) variable or (data type) constant

Figure 11.20 gives the code listing of the program that makes use of type casting. The output of the program is shown in figure 11.21.

```
11_9.c - SciTE  
File Edit Search View Tools Options Language Buffers Help  
11_9.c  
/* Example 9: Program to illustrate use of type cast */  
  
#include <stdio.h>  
int main( )  
{  
    int total_cost = 575 , quantity = 24;  
    float cost_item;  
  
    /* C expression for calculating cost per item without type casting */  
    cost_item = total_cost / quantity;  
  
    printf("\nCost of one item without type casting is %f", cost_item);  
  
    /* C expression for calculating cost per item without type casting */  
    cost_item = total_cost / (float) quantity;  
  
    printf("\nCost of one item after type casting is %f\n", cost_item);  
    return 0;  
}  
/* End of Program */
```

Figure 11.20 : Program illustrating use of type casting

```

11_9.c - SciTE
File Edit Search View Tools Options Language Buffers Help
11_9.c
>gcc -pedantic -Os -std=c99 11_9.c -o 11_9
>Exit code: 0
>./11_9

Cost of one item without type casting is 23.000000
Cost of one item after type casting is 23.958334

>Exit code: 0

```

Figure 11.21 : Output of Example 11.9

Explanation

The program here is to find out cost per item, given the cost of 24 items. We have defined "total_cost" and "quantity" as integers while "cost_item" is float. As can be seen the first C expression for calculating cost per item gives value 23.000000 as both the operands used are integer. While the same expression when used along with type cast gives value 23.958334 which is desired result.

Storage Classes

The variables used in C program are generally stored in primary memory of the computer. It is also possible to store some variables in registers. C language provides us four storage classes namely, automatic, external, register, and static. These storage classes allows user to specify the location where he/she wants to store the variable. Let us now discuss them one by one.

Automatic Variables

All the variables by default have storage class as automatic. To explicitly define the variable as automatic, we use the keyword auto. The following syntax allows us to define an automatic storage class for a variable:

auto data type identifier;

For example, the definition `auto int number;` indicates that the variable `number` is of type integer and its storage class is automatic. Such variables are created automatically when the function is called and destroyed when the function returns the control back to the calling function. These variables are by default, are assigned a garbage value (value that does not have any significance to the user) and are stored in the primary memory.

External Variables

At times the programmer needs to share a variable between two functions or programs. We can use storage class external to share variables between two different functions or programs. It is necessary though that the variable must be defined as global variable in at least one program. The following syntax allows us to define an external storage class for a variable:

extern data type identifier;

For example, the definition `extern char choice;` indicates that the variable `choice` is of type character and its storage class is external. The default value of the external variable is zero, and they are stored in the primary memory.

Register Variables

To have fast access to variables, we can store the variables in the CPU registers. C language provides us a storage class called register to store the variables in the CPU registers. The following syntax allows us to define a register storage class for a variable:

register data type identifier;

For example, the definition `register int counter;` indicates that the variable `counter` is of type integer and its storage class is register. The register class variables are assigned garbage value by default.

Static Variables

The static storage class when used along with variable makes the variable value permanent within a specified region. A static storage class variable is stored permanently in the primary memory and, by default, is assigned value zero. The following syntax allows us to define a static storage class for a variable:

static data type identifier;

For example, the definition `static int max;` indicates that the variable `max` is of type integer and its storage class is static.

Summary

In this chapter we learnt about the basic data types like `int`, `float`, `char` and `void`. We also learnt about some user defined and derived data types like `typedef`, `enum` and arrays. We also saw how to assign values to the variables using assignment operators. Later we looked at various operators like arithmetic, relational, increment - decrement, conditional, logical, bitwise and some special operators. We saw how to create expression using different operators and how to evaluate them. Finally we saw four storage classes that C allows us to use.

Instruction for Teachers

All the data types discussed here are based on ANSI C99 standards. The examples discussed here show how to use the operators. Teacher can design more exercises and then give it to students.

EXERCISE

1. What is data type ? Explain its significance.
2. Explain the significance of `void` data type.
3. Explain the significance of user defined data type.
4. Explain the significance of derived data type.
5. State whether true or false :
 - (a) `char` data type requires two bytes of memory space.
 - (b) An `unsigned` integer variable can store value less than zero.
 - (c) The data type `double` is used to store integer value.
 - (d) To increase the range of `int` data type it is prefixed by keyword `long`.
 - (e) Scientific form of real value is expressed as `exponent * 10 mantissa`

- 6.** Choose the correct option from the following:
- (1) Which of the following keywords refer to integer data type ?
(a) integer (b) Integer (c) INTEGER (d) int
- (2) Which of the following refers to an empty value ?
(a) void (b) Void (c) char (d) float
- (3) Which of the following refers to size of memory allocated to float as per gcc compiler ?
(a) 1 (b) 2 (c) 4 (d) 6
- (4) Which of the following is a valid keyword in C that identifies valid character value ?
(a) char (b) character (c) CHAR (d) CHARACTER
- (5) Which of the following is a user defined data type in C ?
(a) int (b) enum (c) char (d) float
- (6) Which of the following is used to provide an alias to existing data types in C ?
(a) enum (b) def (c) pointer (d) typedef
- (7) The symbol & belong to which of the following operator types in C ?
(a) Relational (b) Arithmetic (c) Logical (d) Bitwise
- (8) The symbol ! belong to which of the following operator types in C ?
(a) Relational (b) Arithmetic (c) Logical (d) Bitwise
- (9) The = symbol is used for which of the following operations in C ?
(a) Equality Check (b) Assignment
(c) Comment (d) Formula
- (10) Which of the following refer to the meaning of value++ in C ?
(a) Post Increment (b) Post Decrement
(c) Pre Increment (d) Pre Decrement

LABORATORY EXERCISE

- 1.** Write a C program to convert given meters into millimeters.
- 2.** Write a C program to convert given Fahrenheit into Celsius using formula $C = (F - 32) * 5/9$.
- 3.** Write a C program to find average of three numbers.
- 4.** Write a program to find out how many tiles of 50 square cms can be fitted in the plot of 2000 sq meters.
- 5.** Write a C program to find area of rectangle.
- 6.** Write a C program to check whether a given number is divisible by other given number.
- 7.** Write a C program to calculate the compound interest on a loan amount of Rs. P taken at the rate of R% for N years. The values of P, R and N are to be specified by student.
- 8.** Write a C program to find volume of a cube.
- 9.** Write a C program to calculate area of circle.
- 10.** Write a C program to calculate the simple interest on a loan amount of Rs. P taken at the rate of R% for N years. The values of P, R and N are to be specified by student.



Using I/O Operations

All programming languages allow to read input (known as input operations) and to write output (known as output operations). Here, input means to read data from any input device like keyboard and output means to write data to output devices like monitor, printers, etc. Input/Output operations are more popularly known as I/O operations. The C language does not have any built-in statement to perform input and output operation. These data input and output operations are carried out by the standard input/output built-in library functions. So far we have used `scanf()` as a standard input and `printf()` as a standard output function in our programs. In this chapter we will discuss how to perform I/O operations using functions like `getchar()`, `getch()`, `gets()`, `scanf()`, `printf()`, `putchar()`, `putc()`, `puts()`, etc. We will also discuss how to perform formatted input and output operations.

We use variables to process data in our program. There are two basic ways to give input to the variables. One way is to use assignment operator. For example, the statement `number=5;` will give input 5 to the number variable. Second way is to read data from user during execution of a program. For this you may use any inbuilt functions related to input operations. Let us discuss commonly used inbuilt input functions.

Inbuilt Input Functions

Input in a program can be possible through any of the input devices like keyboard, mouse etc. C provides several inbuilt input related functions which are stored in a C library. To use any inbuilt function from C library, we have to include respective library of that function in the beginning of program using `#include` statement. You may have noticed that in many programs we have used following statement in beginning of our program:

```
#include<stdio.h>
```

The name `stdio.h` stands for standard input-output header file. This header file contains various input and output operations related functions. The statement `#include` informs the compiler to find `stdio.h` file and place its contents in the beginning of our program. The content of header file is then become a part of our program. More discussion related to using C library function is given in chapter of C Function. Note that on some of the computer system it is not necessary to include this header file when we use `scanf()` function. In following section now we discuss input functions `getchar()`, `getch()`, `getc()` and `gets()`.

`getchar()`

One of the simplest way to read a character in a C program at runtime is to use the `getchar()` function. The standard form of `getchar()` is as under:

```
variable_name = getchar();
```

`variable_name` is any valid C variable name with character as a data type. The `getchar()` function has no parameters. Every time when we call it, it reads single character input. The function returns an int, the ASCII code of the relevant character, but we can assign the input given by user to a char variable.

Let us try to understand the use of getchar() function using example 12.1. Here we want to use the answer typed in by the user to print a message in our program. Figure 12.1 gives the code listing of the example 12.1 while figure 12.2 shows its output.

The screenshot shows the SciTE editor window with the file '9-1.c' open. The code is as follows:

```
/* Example 1: Program to demonstrate use of getchar() function. */

#include<stdio.h>
int main()
{
    char answer;

    printf("Would you like to study? \n");
    printf("Enter 'Y' for yes and 'N' for no: ");

    answer = getchar();

    if(answer =='Y' || answer =='y')
        printf("Go to study room...\n");
    else
        printf("Go to play ground...\n");

    return 0;
}
// End of Program
```

Figure 12.1 : Code listing of Example 12.1

The screenshot shows a terminal window on an Ubuntu system. The command 'gcc 9-1.c' is run, followed by './a.out'. The program asks 'Would you like to study?' and waits for input. The user types 'Y' and the program outputs 'Go to study room...'. The terminal prompt 'kpp@ubuntu:~\$' is visible at the bottom.

Figure 12.2 : Output of Example 12.1

Explanation

The first statement of example 12.1 declares a variable `answer` of character data type to store a single character. When `getchar()` function is encountered, the program will wait for the user to press a key. Any character pressed by the user from keyboard will be stored in the `answer` variable. The same will also be echoed on the screen. Say for example if the user enters character 'Y' or 'y' then a message "Go to study room..." will be displayed. If user enters any other character then message "Go to play ground..." will be displayed.

getch()

The function getch() is also used to read a character from the user. The difference between getchar() and getch() is that the character keyed in will not be echoed on the screen, i.e. the character entered by the user will not be visible on screen. This function is useful when we don't want to show a character typed by the user on screen. Replace the function getchar() with getch() in example 12.1 and observe the difference between these two functions.

getc()

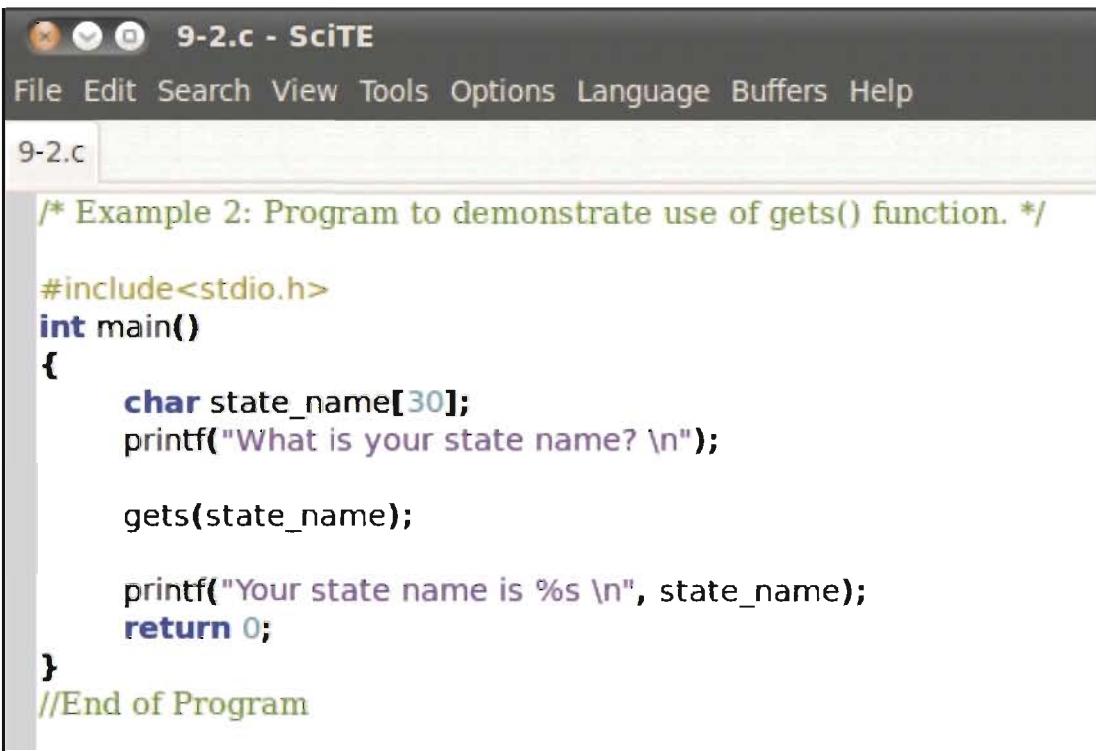
Like getchar() and getch(), getc() is also used to read a single character. But the difference here is that getc() reads a character from a file instead of the standard input device. More discussion related to getc() is out of scope of your syllabus.

gets()

So far we have seen library function getchar(), getch() and getc() which can read only one character at a time. To read two characters, we can call getchar() function two times. Calling getchar() function many times is not good logic when we want to read a group of characters. The better alternative is to use gets() function to read a string. Note that we call a group of characters as a string. The general syntax of gets() function is as under:

gets(variable_name);

The gets() function takes one string variable as an argument. Here variable_name is the character array. More about character array is discussed in Array chapter. During execution of program when gets() is encountered, it will wait for user to enter characters from the input device. The function will read characters until a new line character is entered by the user and then appends a null character ('\\0') to the end of string. After execution of this function, all the characters and null value at the end of all the characters are stored into the declared variable_name. Let us try to understand the use of gets() function using example 12.2. Figure 12.3 gives the code listing of the example 12.2 while figure 12.4 shows its output.



The screenshot shows the SciTE IDE interface with the title bar "9-2.c - SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, and Help. The current buffer is "9-2.c". The code listed is:

```
* Example 2: Program to demonstrate use of gets() function.

#include<stdio.h>
int main()
{
    char state_name[30];
    printf("What is your state name? \n");

    gets(state_name);

    printf("Your state name is %s \n", state_name);
    return 0;
}
//End of Program
```

Figure 12.3 : Code listing of Example 12.2

```

kpp@ubuntu: ~
File Edit View Terminal Help
kpp@ubuntu:~$ ./a.out
What is your state name?
Gujarat
Your state name is Gujarat
kpp@ubuntu:~$ 

```

Figure 12.4 : Output of Example 12.2

In the example 12.2, the user input will be stored in state_name variable. The printf() function displays the same string on the screen. Details about printf() statement is discussed later in this chapter.

Note that during compilation of any program containing gets() function, you may get warning message like "the gets function is dangerous and should not be used". But as it is only warning message, you may continue to execute your program.

Formatted Input

There are some situations in our program where use of input functions discussed so far are not much useful. The getchar(), getch() and getc() can be used to read only single character at a time whereas gets() can be used to read multiple characters until user press enter key. What happens if user wants to input data like

33 Mudra Ahmedabad

In the said example first part of data is the roll number of student; second and third parts of data are name and city of the student respectively. As can be observed roll number is of integer data type and remaining two are of character data type.

C provides the facility called formatted input functions using which we can read such type of data. We have already used one of the formatted input function scanf() in many examples. Let us explore various options available for formatted data input using scanf() function.

scanf()

The **scanf()** stands for **scan** formatted. This function allows us to input data into particular data format like int, char, float, etc. The common syntax of scanf() function is as under:

```
scanf("control string", &variable1, &variable2, ....&variableN);
```

The *control string* specifies the data format in which values of variables are to be stored. The variable1, variable2, ...variableN are names of variables. These variable names are preceded by &(ampersand) sign and are separated by comma ','. The & sign is known as address of operator in C language and it specifies the memory location where the input given by user is stored. This may be clear from the figure 12.5.

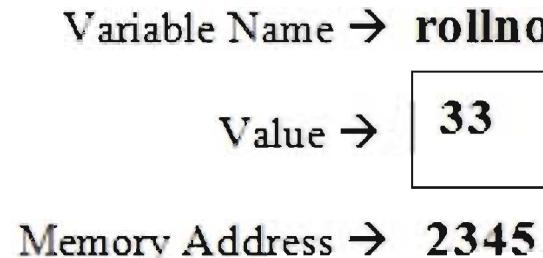


Figure 12.5 : Memory Layout

As shown in figure 12.5 variable name is rollno and its value is 33. Actual value of variable is stored at memory address 2345(this number is just an example; it may be any valid memory address of computer). In C each variable is identified by its name as well as its memory address. The C compiler uses memory address for processing whereas we refer memory location by its name in our program. More discussion related to the address of operator is out of scope of this book. You will learn this concept in detail while studying C pointers.

The *control string* is also known as **format string**. This will help the compiler to interpret data input given by the user. Control string also specifies the order in which variable values appear in the input. Each format must be preceded by '%' sign and a character specifying data type. Let us see how to use control string in our program with example.

Reading Integers

Following program segment can be used to read three integer values from the users which are stored into marks1, marks2 and marks3 variables.

```
int marks1, marks2, marks3;  
scanf("%d %d %d", &marks1, &marks2, &marks3);
```

During execution of above statement, if user enters: 70 80 90 then 70 will be stored in marks1, 80 in marks2 and 90 in marks3. We can also specify the field width of a number to be read along with %d as under:

```
scanf("%2d %4d", &marks1, &marks2);
```

During execution of previous scanf(), if user enters: 70 1234 then 70 will be stored in marks1 and 1234 in marks2 variable.

But for previous scanf() statement, if user enters: 1234 70 then variable marks1 will be assigned value 12 (because of field width %2d) and mark2 will be assigned 34 (which is unread part from 1234). The value 70 will remain unused for the said scanf() statement.

Reading Real Numbers

To read real numbers (floating point numbers) like percentages of a student in a program, following program segment can be used.

```
float per1, per2;  
scanf(" %f %f ", &per1, &per2);
```

Here we have defined two float variables per1 and per2. The scanf statement stores float values in these variables. While executing above code, if user enters: 85.25 90.65 then 80.25 will be stored in variable per1 and 90.65 in variable per2. The scanf() statements reads real numbers using specification "%f". If user wants to read a number with **double** data type then specification "%lf" is to be used in place of "%f".

Reading Character and Word

We have seen how to read a character and characters (string) using in-built function like getchar() and gets(). Same task of reading a characters and a words can be achieved using scanf() statement with %c and %s specification respectively. Observe the given program segment:

```
char c1, c2;  
char city[20];  
scanf(" %c %c %s", &c1, &c2, city);
```

During execution of the given scenario if user enters data like: **A B Gandhinagar** then 'A' will be stored in variable c1, 'B' in c2 and 'Gandhinagar' in character array named city. More discussion related to character array is given in Chapter 15 of this book.

Note that while reading string using scanf() statement, & (ampersand) is not required with the variable name. Also remember that %s specification terminates its reading when blank space encounters in input. See the following example :

```
char city[20];
scanf("%s ", city);
```

During execution, if user enters: **New Delhi** then only "New" will be stored into variable city. The word "Delhi" is ignored by the scanf() statement as %s specification terminates its reading at space after "New".

Restricted user input with %[characters] and %[^ characters]

C language scanf() function provides wonderful facility of %[characters] using which we can specify the permissible characters in the input string. If user input string contains any character which is not listed in the %[characters] specification then the string will be terminated at the first occurrence of such character. Let us try to understand this concept using example 12.3. Figure 12.6 gives the code listing of the example 12.3 while figure 12.7 shows its output.

The screenshot shows the SciTE code editor interface. The title bar says "9-3.c - SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, Help. The code area contains:

```
/* Example 3 Program to demonstrate use of %[characters] specification. */

#include<stdio.h>
int main()
{
    char student_name[30];

    printf("Enter your name containing only alphabets and space: \n");
    scanf("%[a-zA-Z]", student_name);
    printf("Your name stored in variable is %s \n", student_name);
    return 0;
}
//End of Program
```

Figure 12.6: Code listing of Example 12.3

The screenshot shows a terminal window on Ubuntu. The prompt is "kpp@ubuntu: ~". The session starts with "File Edit View Terminal Help", followed by:

```
kpp@ubuntu:~$ gcc 9-3.c
kpp@ubuntu:~$ ./a.out
Enter your name containing only alphabets and space:
Ragi Bharat Patel
Your name stored in variable is Ragi Bharat Patel
kpp@ubuntu:~$ ./a.out
Enter your name containing only alphabets and space:
Ragi B. Patel
Your name stored in variable is Ragi B
kpp@ubuntu:~$
```

Figure 12.7 : Output of Example 12.3

In example 12.3, specification `%[a-z A-Z]` in `scanf()` specifies that only a-z, space and A-Z are accepted and stored into variable `student_name`. Hence in first execution, full name of student is stored into variable. But during second execution, user input contains dot(.) which is not listed into specification hence only "Ragi B" is stored into variable.

By using `%[^characters]` specification in the `scanf()`, the process of inputting the value is immediately terminated when any of the character from the list is encountered. Let us try to understand this concept using example 12.4. Figure 12.8 gives the code listing of the example 12.4 while figure 12.8 shows its output.

The screenshot shows the SciTE IDE interface with the title bar '9-4.c - SciTE'. The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, and Help. The code editor window contains the following C program:

```
/* Example 4: Program to demonstrate use of %[^characters] specification. */

#include<stdio.h>
int main()
{
    char user_data[100];

    printf("Enter your details:\n");
    printf("Enter * to exit:\n");
    scanf(" %[^\n]", user_data);
    printf("Your details stored in variable is \n%s\n", user_data);
    return 0;
}
//End of Program
```

Figure 12.8 : Code listing of Example 12.4

The screenshot shows a terminal window titled 'kpp@ubuntu: ~'. The terminal prompt is 'kpp@ubuntu:~\$'. The user runs the command 'gcc 9-4.c' followed by './a.out'. The program prompts for input, and the user types 'My name is Purva.' followed by a new line. The program then prints the stored input back to the user.

```
kpp@ubuntu:~$ gcc 9-4.c
kpp@ubuntu:~$ ./a.out
Enter your details:
Enter * to exit:
My name is Purva.
Your details stored in variable is
My name is Purva.
My mother's name is Krina*
kpp@ubuntu:~$
```

Figure 12.9 : Output of Example 12.4

During execution of example 12.4 we can input any character (including new line character). When * is detected in the input string, the process of accepting input string is stopped. Note that in our example, we will be able to store only 100 characters as `user_data` variable length is declared as 100.

Reading Mixed Data

It is possible to read different types of data (like integer, float, char, string) using a single scanf() statement. In such cases we have to make sure that input data items must match with the control specification of scanf() in order and data types. Observe the following program segment to understand this concept.

```
int roll_no;  
char grade, name[30];  
float percen;  
scanf("%d %f %s %c", &roll_no, &percen, name, &grade);
```

If during execution the input specified is: **11 90.55 Vani A** then 11 is stored in roll_no, 90.55 in percen, 'Vani' in name and 'A' in grade variable. But for the same program code, if user inputs data as : **11 Vani A 90.55** then the system will give error because user had entered a characters when compiler was looking for floating point number. In this case scanf() statement will terminate its reading process after reading the first value.

The list of characters which are used in scanf() control string to read different types of data is given in table 12.1

Data Type	Corresponding Character
For reading a decimal integer	%d
For reading a character	%c
For reading a floating point value	%f or %e or %g
For reading a string	%s
For reading a unsigned integer	%u
For reading a short integer	%h
For reading a long integer	%ld
For reading a double	%lf
For reading a long double	%L

Table 12.1 : Characters used in scanf() control string

Inbuilt Output Functions

The computer system performs three basic tasks: input, process and output. We had seen that how inbuilt input functions can be used to read data from user. Now let us study how inbuilt output functions can be used for displaying processed data. The output can be obtained on output devices like monitor, printer and file. We will discuss how output can be displayed on the monitor, which is considered as a standard output device in C. We will not discuss file and printer management as it is out of scope of your syllabus.

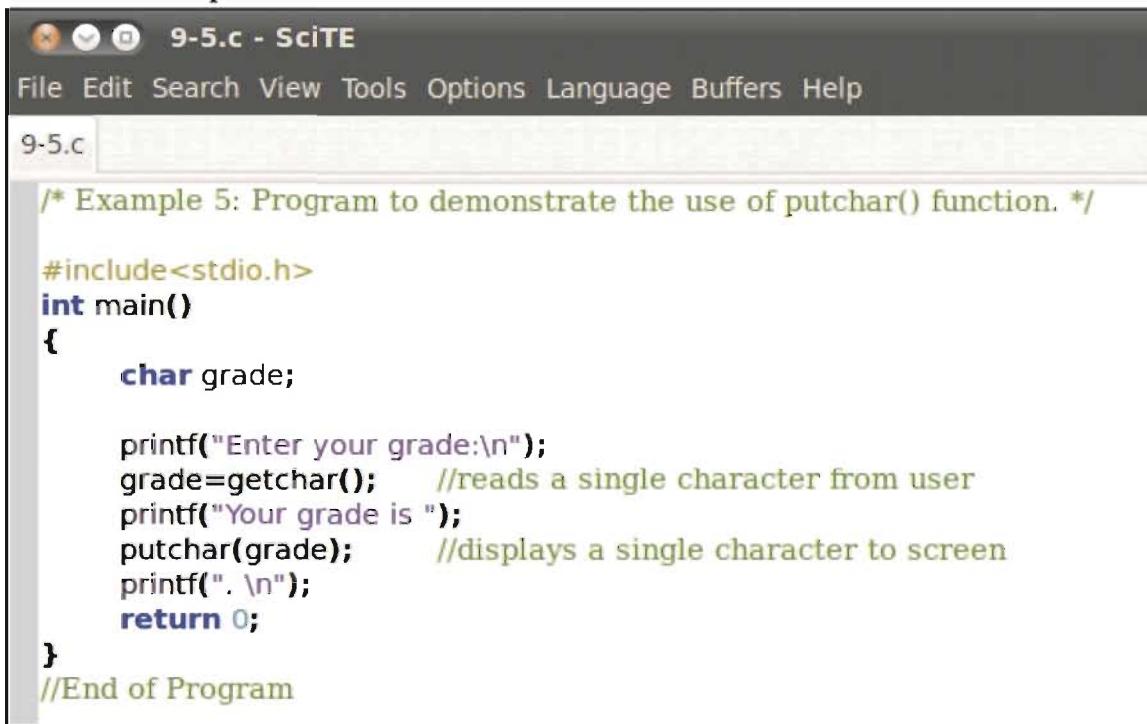
Like input, sending output to output devices is possible through C library functions. The inbuilt output functions are available in the <stdio.h> header file. Let us discuss the output related inbuilt functions: putchar(), puts() and printf().

putchar()

The function putchar() can be used to write a single character to the standard output device. The general syntax of putchar() is

```
putchar(character);
```

Here character may be the char type variable or it may be any valid C language character. When this function is executed, character will be displayed on the monitor. Let us try to understand this concept using example 12.5. Figure 12.10 gives the code listing of the example 12.5 while figure 12.11 shows its output.



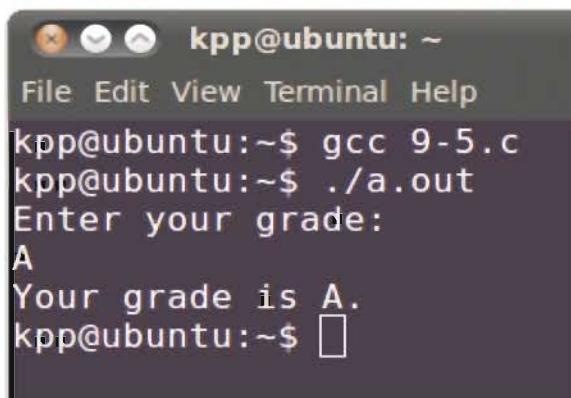
The screenshot shows the SciTE code editor window titled "9-5.c - SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, and Help. The code area contains the following C program:

```
/* Example 5: Program to demonstrate the use of putchar() function. */

#include<stdio.h>
int main()
{
    char grade;

    printf("Enter your grade:\n");
    grade=getchar();      //reads a single character from user
    printf("Your grade is ");
    putchar(grade);        //displays a single character to screen
    printf(".\n");
    return 0;
}
//End of Program
```

Figure 12.10 : Code listing of Example 12.5



The screenshot shows a terminal window on an Ubuntu system. The title bar says "kpp@ubuntu: ~". The menu bar includes File, Edit, View, Terminal, and Help. The terminal prompt is "kpp@ubuntu:~\$". The user runs "gcc 9-5.c" and then "./a.out". The program asks for input ("Enter your grade: ") and displays the character "A" back to the user ("Your grade is A."). The terminal prompt is "kpp@ubuntu:~\$".

Figure 12.11 : Output of Example 12.5

The program will read a single character using getchar() function and displays it using putchar() function on the screen.

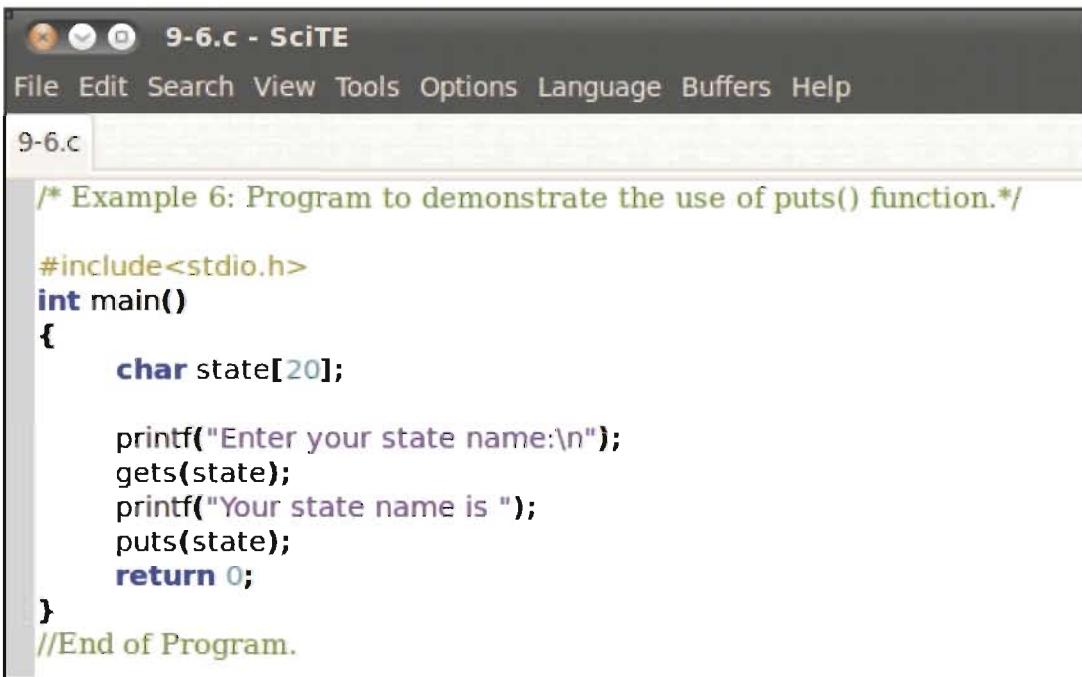
The putchar() function has one limitation that it can display only one character at a time. To display multiple character at a time you may use loop concept of C. The simpler solution to display multiple characters is puts() function.

puts()

We may use puts() function to display multiple characters (string or character array) on an output device. The general syntax of puts() is as under:

puts(variable_name);

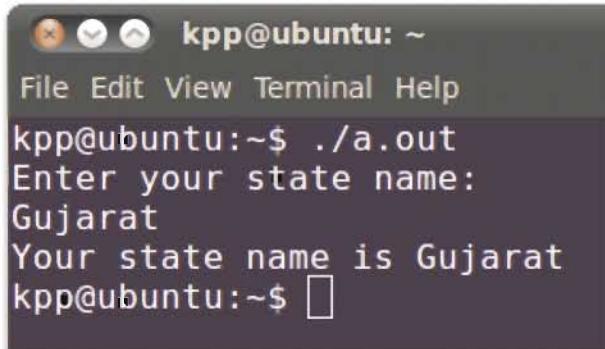
where *variable_name* is the character array or string. The puts() functions writes the content stored into *variable_name* to the monitor till the null character is encountered. Note that every string contains a null character at the end but puts() function will not display this character on screen. Observe the example 12.6 to make this concept clear. Figure 12.12 gives the code listing of the example 12.6 while figure 12.13 shows its output.



```
9-6.c - SciTE
File Edit Search View Tools Options Language Buffers Help
9-6.c
/*
 * Example 6: Program to demonstrate the use of puts() function.*/
#include<stdio.h>
int main()
{
    char state[20];

    printf("Enter your state name:\n");
    gets(state);
    printf("Your state name is ");
    puts(state);
    return 0;
}
//End of Program.
```

Figure 12.12 : Code listing of Example 12.6



```
kpp@ubuntu: ~
File Edit View Terminal Help
kpp@ubuntu:~$ ./a.out
Enter your state name:
Gujarat
Your state name is Gujarat
kpp@ubuntu:~$
```

Figure 12.13 : Output of Example 12.6

The program will reads multiple characters at a time using gets() function and stores it to the state variable. Using puts() function value stored into state variable is displayed on the screen.

Formatted Output

The output functions discussed so far give output without any formatting. The value stored in the variable is simply displayed on screen. But sometimes to prepare reports from our program, we may want to

display output with formatting which makes it good in look and easy to understand. The printf() function provides features which can be used to display output with different types of formatting.

printf()

The printf() function is used to display formatted output on screen. The general syntax of the printf() is as under:

printf("control string", var1, var2, ...varN);

where var1, var2 ... varN may be variables or constants or expressions. The output format specification of variables is given in control string. The control string may contain all or some of the following items :

- Set of characters (string) which will be printed on monitor as they appear in control string
- Format specifier for each variables
- Escape sequence characters like \n (new line), \t(tab), \b(back space)

Like scanf(), the control string entries are separated by space and preceded by '%'. The later part of printf() contains list of variables whose values are to be printed. These variables must match in number, order and its types with format specification given in control string.

Followings are some of the simple examples of printf() statements which we had used in our programs so far to display output on the screen. These statements will print output without any formatting.

- printf("Hello World");
- printf("Your age is %d", age);
- printf("Your name is %s", student_name);

To display formatted output with different types of alignment, we may use format specification in the control string of printf() statement. The control string has following general format:

%l.m p

Here, '**T**' is an integer number specifying total character positions in which the variable value is to be printed. '**m**' is used for specifying the number of digits after decimal point(of real number) or number of characters to be printed from a string variable. Specifying value of l and m are optional. The '**p**' shows the type of variable. The list of different data types used with printf() is given in table 12.2

Data Type	Corresponding Character
For printing a decimal integer	%d
For printing a character	%c
For printing a floating point value without exponent	%f
For printing a floating point value in exponent form	%e
For printing a string	%s
For printing a unsigned integer	%u
For printing a long decimal integer	%ld
For printing a double	%lf
For printing a long double	%Lf
For printing integer in octal form	%o
For printing integer in hexadecimal form	%x

Table 12.2 : Data type used in printf() control string

Let us try to understand the code listing of example 12.7 shown in figure 12.14. It shows various formats of printf() statement.

The screenshot shows a SciTE window titled "9-7.c - SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, and Help. The file "9-7.c" contains the following C code:

```
/* Example 7: Program to demonstrate the various format  
of printf( ) statement.*/  
  
#include<stdio.h>  
int main()  
{  
    int number = 1234;  
  
    printf(" %d \n", number);  
    printf(" %8d \n", number);  
    printf(" %-8d \n", number);  
    printf(" %2d \n", number);  
    printf(" %08d \n", number);  
  
    return 0;  
}  
//End of Program
```

To the right of the code, the output of the program is displayed:

```
gcc 9-7.c  
>gcc 9-7.c  
>Exit code: 0  
.a.out  
>.a.out  
1234  
1234  
1234  
1234  
00001234  
>Exit code: 0
```

Figure 12.14 : Code listing and output of Example 12.7

Explanation

As can be observed in the first output by default the printf() statement uses left justification to display the output. The second printf() statement uses total 8 characters width to display output with right side justification. Third printf() statement shows that it is possible to print output with left side justification using minus sign(-) after % character. In fourth printf() statement there is no effect of specifying %2d as the specified width is less than width of actual variable. Last printf() statement shows that it is possible to print additional padding characters (zero) before the actual value of number variable.

In previous example we had seen that how integer variable can be displayed with different types of formatting on screen. Let us check that how to specify various formatting options with floating point/real numbers. Recall the general specification %l .m p where m indicates number of digits to be displayed after decimal point. When float variable is displayed, it is rounded to m decimal places and printed right justified in the width of l columns. By default 6 digits are displayed after decimal point when m is not specified. Figure 12.15 shows how to use printf() statements to display float variable f1 with 123.456 value using different formatting.

Statement	Output
<code>printf(" %f ", f1);</code>	1 2 3 . 4 5 6 0 0 0
<code>printf(" %8.3f ", f1);</code>	1 2 3 . 4 5 6
<code>printf(" %8.1f ", f1);</code>	1 2 3 . 5
<code>printf(" %08.1f ", f1);</code>	0 0 0 1 2 3 . 5
<code>printf(" %-8.1f ", f1);</code>	1 2 3 . 5
<code>printf(" %10.3e ", f1);</code>	1 . 2 3 5 e + 0 2
<code>printf(" %10.4e ", f1);</code>	1 . 2 3 4 6 e + 0 2

Figure 12.15 : Float variable displayed using different formats

Using printf() statement formatting can also be applied to character and string. The format specification is similar to that of real number. We use value of l to specify the field width for display and the value of m for number of characters to be printed. The string in output will be printed left justified when l is not specified. When l is specified the right justified string will be displayed on screen. Like integer format specification, by writing minus sign, we can print the string left justified.

Let us make string output format specification concept clear using following printf() statements. We use string (character array) variable str with value "GUJARAT INDIA" as value stored in it. Figure 12.16 shows various formatted output.

Statement	Output
<code>printf(" %s ", str);</code>	G U J A R A T I N D I A
<code>printf(" %8s ", str);</code>	G U J A R A T I N D I A
<code>printf(" %16s ", str);</code>	G U J A R A T I N D I A
<code>printf(" %16.7s ", str);</code>	G U J A R A T
<code>printf(" %.7s ", str);</code>	G U J A R A T
<code>printf(" %16.9s ", str);</code>	G U J A R A T I
<code>printf(" %16.15s ", str);</code>	G U J A R A T I N D I A

Figure 12.16 : Character variable displayed using different formats

In last printf() statement the value of m is 15 which is greater than length of string stored in str, hence all characters from str variable will be displayed with right justification.

Summary

In this chapter we had learnt that how formatted input and output related inbuilt function can be used in our program. We can display output without formatting also, but the output displayed with the help of format specification makes it more readable and attractive to the user. Now it is upto the programmers how they apply these functions to make their programs more attractive.

EXERCISE

1. How is the control string of scanf() different than the control string of printf() ?
2. What is the purpose of formatted output functions?
3. Discuss the use of formatted input functions in a program. Giving suitable example, explain how scanf() function can be used to read the integer and real numbers.
4. Explain the concept of restricted user input with %[characters] and %[^ characters] with suitable examples.
5. State whether true or false :
 - (a) The function getchar() can be used to read multiple character at a time.
 - (b) The function gets() can be used to read a single character.
 - (c) The single printf() statement may generate multiple lines of output.
 - (d) The %.10s format specification will print first 10 characters of a given string.
 - (e) A single scanf() statement can be used to read multiple values.
 - (f) By default real values are printed with a precision of six decimal points.
6. Do as directed :
 - (1) State the errors if any in the following program statements.
 - (a) scanf("%d %c", &number, city_code);
 - (b) scanf("%d %d %s" \n, &marks1, &marks2, city_name);
 - (c) scanf("%d %f %c %s", &num1, &price, item_code);
 - (2) State the output produced by the following statements.
 - (a) printf("%d %c %f", 101,'X', 20.20);
 - (b) printf(" %3d %8.2f ", 12345, 222.123);
 - (c) printf(" %.5s", "Hello World");
 - (d) printf(" %15.8s", "Hello Student");
 - (e) printf("%0.5s", "Hello World");

- (3) Write output of following program.

```
#include<stdio.h>
int main( )
{
    float result = 98.12345;

    printf("Your result is %f \n", result);
    printf("Your result is %.2f \n", result);
    printf("Your result is %5.3f \n", result);
    return 0;
}
```

- (4) Find errors if any, correct it and then write output of the following program.

```
#include<std.h>
int main( )
{
    int sum = 1234;
    float price = = 550.50;
    char City[20] = "Ahmedabad";

    printf("The sum is %d....", price);
    printf("The price is %s....", price);
    printf("The City name is %.3s....", city);
    return 0;
}
```

7. Choose the correct option from the following :

- (1) Which of the followings are useful in C language I/O operations ?
(a) Monitor (b) Keyboard (c) Mouse (d) All of these
- (2) Input in a program can be possible using which of the following device ?
(a) Keyboard (b) Speaker (c) Printer (d) Monitor
- (3) Which of the following is not an input functions ?
(a) getchar() (b) getch() (c) puts() (d) gets()

- (4) What is the output of statement: printf("%4s", "Krusha"); ?
(a) Krus (b) usha (c) Krusha (d) Krush

(5) Which of the following function is more appropriate for accepting a string ?
(a) getch() (b) gets() (c) getchar() (d) getc()

LABORATORY EXERCISE

123.45 34.56 -7878.123 -0.965

Store these numbers in to appropriate data type variables, print the rounded numbers to the nearest integers.

6. Write a C program to read value of A and B, then print the result in one line for following expressions:

(i) $(A + B) / (A - B)$ (ii) $(A+B)(A-B)$ (iii) $(A * A)(B + B)$



Decision Structures

Most of the C programs presented so far follow the sequential structure of program execution. The instructions are executed one by one sequentially in the order in which they appear in a program.

In real life practice during execution of C program, we may want to execute only some of the statements based on some conditions. For this purpose it is required to change the flow of the sequence of instructions. C language provides facility through special kind of statements that can change the flow of a sequence of instructions in the program. Such statements are called decision structure statements.

Need for Decision Structures in C

Decision structure statements help us to jump from one part of the program to another part of program based on result of some conditions. Sometimes decision structure statements are also known as selective structures statements, branching statements or decision making statements. As these statements are controlling the flow of execution, they are also known as control statements.

C language provides two basic types of decision structure statements: ***if*** and ***switch***. In this chapter, we will discuss how ***if*** and ***switch*** statements can be used to change the flow of instructions in the program.

The ***if*** Statement

The ***if*** statement is one of the powerful decision making statement which can be used to transfer control of instruction execution. The ***if*** statement can be used in following different ways:

- Simple if statement
- if-else statement
- nested if statement and
- else-if ladder statement

Simple ***if*** statement

The simplest form of the decision structure statement is the if statement. It is frequently used in the program for decision making and allowing us to change the flow of program execution.

Example 13.1 : Let us try to understand a C program that asks to input marks of one subject and display the congratulation message if marks entered by user are greater than 80. Figure 13.1 gives the code listing of the example 13.1 while figure 13.2 shows its output.

The screenshot shows the SciTE IDE interface with a window titled "10-1.c - SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, and Help. The code editor contains the following C program:

```

/*
 * Example 1: Program to illustrate use of simple if statement
 */
#include<stdio.h>
#include<stdlib.h>

int main()
{
    int marks; // declare a variable
    system("clear"); // clears the screen
    printf("Enter your marks:"); // message to the user
    scanf("%d", &marks); // read a number from user

    if ( marks > 80 ) // check whether marks greater than 80 or not
        printf("Congratulations...\n"); // display message only when marks > 80

    printf("Have a Nice Time...\n");
    return 0;
}

/* End of Program */

```

Figure 13.1 : Code listing of Example 13.1

The screenshot shows a terminal window titled "kpp@ubuntu: ~". The terminal prompt is "kpp@ubuntu: ~\$". The user enters "Enter your marks:85" and the terminal displays "Congratulations..." followed by "Have a Nice Time...".

```

File Edit View Terminal Help
Enter your marks:85
Congratulations...
Have a Nice Time...
kpp@ubuntu:~$ 

```

Figure 13.2 : Output of Example 13.1

Explanation

The first statement in the program after main() function declares the variable named marks. Second statement clears the screen. The printf statement is used to display the message to user. The scanf statement is used to read a value from user and store the entered value in the marks variable. The if statement checks the condition: Is entered marks are greater than 80? If the test expression (marks > 80) is evaluated to true then it executes the printf statement displaying the "Congratulations..." message. If the test expression (marks > 80) is evaluated to false then the printf statement displaying the "Congratulations..." message is skipped. The next printf statement always displays the "Have a Nice Time..." message.

Syntax of simple if statement

The if statement has following syntax:

```

if (test expression)
{
    Statement-block;
}

```

Here the test expression is any valid C language expression. The logical operators are used in test expression. The statement-block may contain any valid C language statement or statements. Note that test expression part should not end with a semicolon.

The syntax says if the test expression is true then executes the statement-block and if the test expression is false then computer skips the statement-block and executes the next instruction of the program.

C programming language assumes any **non-zero** and **non-null** values as true and if it is either **zero** or **null** then it is assumed as false value. Figure 13.3 shows the flow chart of simple if statement.

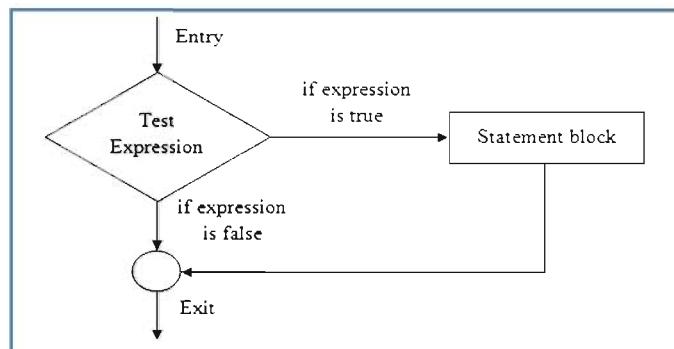


Figure 13.3 : Flowchart of Simple if Statement

The if ... else statement

In simple if statement, there is only one statement block, which gets executed only when test expression is true. Simple if statement takes no action in case when the test expression is false. What happens if we want to take some action in case of false answer from test expression? For example if marks of student are greater than 40 then we want to display message "Student Passed" otherwise we want to display message "Student Failed". In such cases, the if...else structure is useful.

Example 13.2 : Let us try to understand a C program that asks to input marks of one subject and display the message "Congratulation...you are passed" if entered marks are above 40 otherwise display message "Better Luck Next Time...you are failed". Figure 13.4 gives the code listing of the example 13.2 while figure 13.5 shows two different runs of output.

The screenshot shows the SciTE IDE interface with the file '10-2.c' open. The code is as follows:

```

10-2.c - SciTE
File Edit Search View Tools Options Language Buffers Help
10-2.c

/*
 * Example 2: Program to illustrate use of if..else statement.
 */
#include<stdio.h>
#include<stdlib.h>

int main()
{
    int marks; // declare a variable
    system("clear"); // clears the screen
    printf("Enter your marks:"); // message to the user
    scanf("%d", &marks); // read a number from user

    if ( marks > 40 ) // check whether marks greater than 40
    {
        printf("Congratulations...you are passed.\n"); // Condition true
    }
    else
    {
        printf("Better Luck Next Time...you are failed.\n"); // Condition false
    }

    printf("Have a Nice Time...\n");
    return 0;
}
/* End of Program */
  
```

Figure 13.4 : Code listing of Example 13.2

```

kpp@ubuntu: ~
File Edit View Terminal Help
Enter your marks:80
Congratulations...you are passed.
Have a Nice Time...
kpp@ubuntu:~$ 

kpp@ubuntu: ~
File Edit View Terminal Help
Enter your marks:35
Better Luck Next Time...you are failed.
Have a Nice Time...
kpp@ubuntu:~$ 

```

Figure 13.5 : Output of Example 13.2

Explanation

The first statement of program declares marks variable. Second statement clears the screen. Third and fourth statement displays the message and read marks from user respectively. Fourth statement checks whether entered marks are greater than 40 or not. If user enters marks greater than 40 then message "Congratulations...you are passed." will be displayed otherwise the message "Better Luck Next Time...you are failed." will be displayed. The next printf statement always displays the message "Have a Nice Time..." message.

Syntax of if...else statement

The *if...else* statement has following syntax:

```

if (test expression)
{
    True statement-block;
}
else
{
    False statement-block;
}

```

Here the test expression is any valid C language expression. When test expression is evaluated to true, the flow of control is transferred to True statement-block. In case when test expression is evaluated to false, the flow of control is transferred to False statement-block. In both the cases, after execution of if statement the flow control is transferred to the next statement in a sequence of the program. The said concept is explained in the Figure 13.6 using Flowchart of if...else statement.

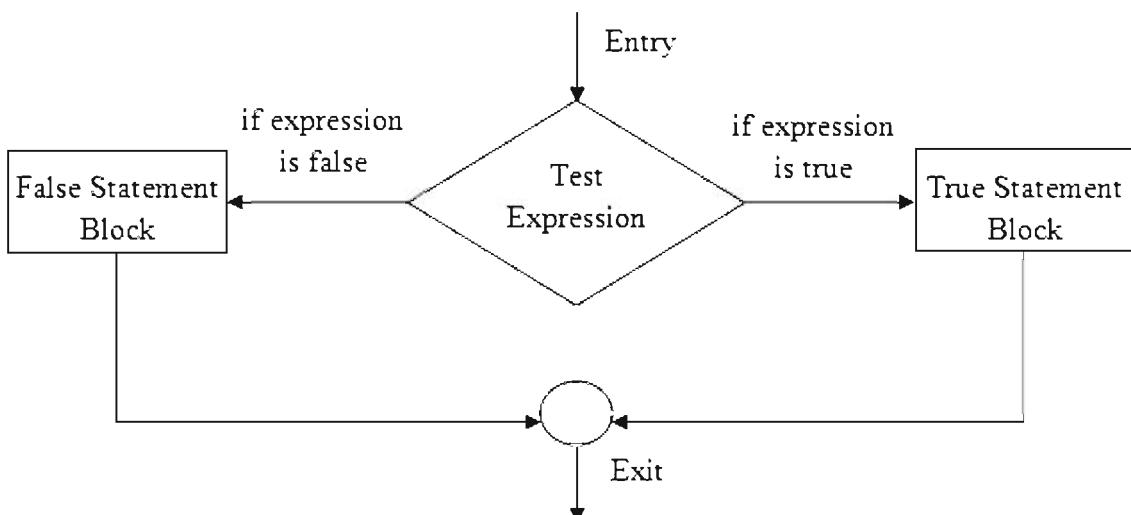


Figure 13.6 : Flowchart of if...else statement

Nested if...else statement

There are situations where we need to execute series of decisions in our program. We may use series of if..else statement in nested form to achieve our task. Consider the following case where one if..else statement block is used inside another if..else block.

```
if (test expression-1)
{
    if(test expression-2)
    {
        Statement block-1;
    }
    else
    {
        Statement block-2;
    }
}
else
{
    Statement block-3;
}
```

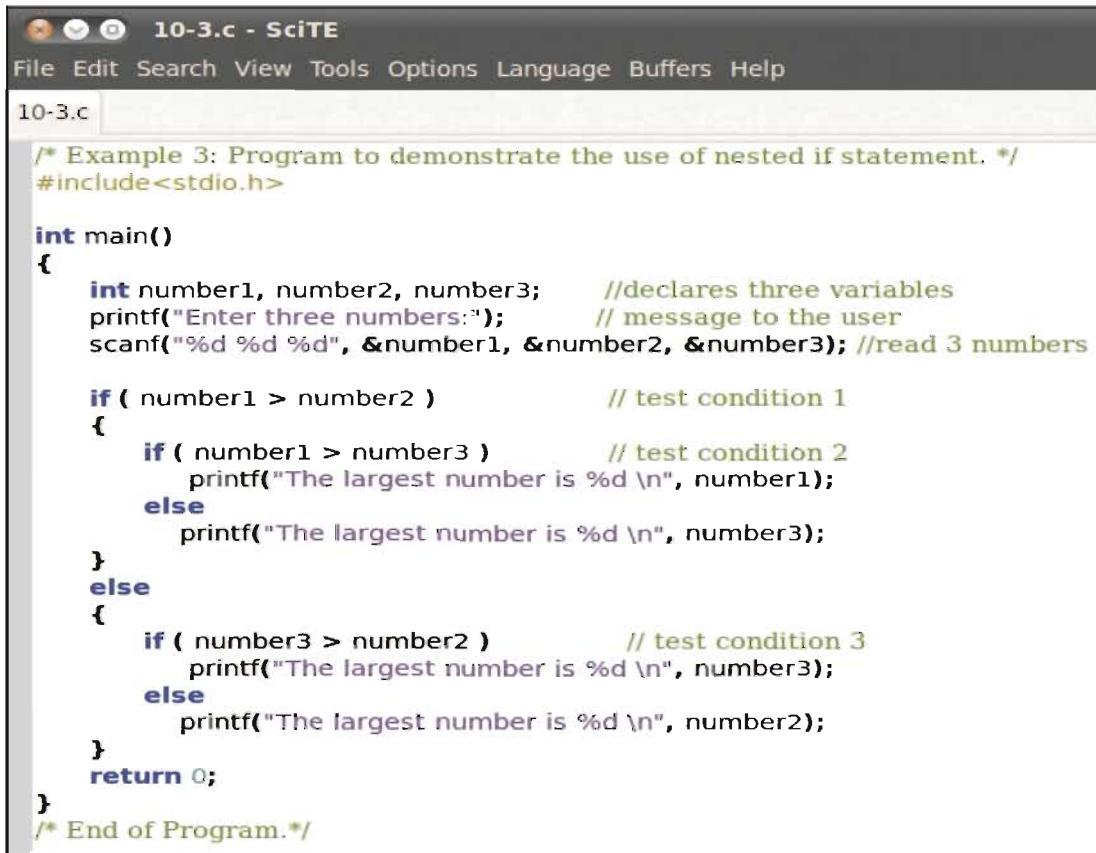
The logic of execution of nested if statement is as under :

- If test expression-1 is true, then test expression-2 is checked.
- If test expression-2 is true then Statement block-1 will be executed otherwise Statement block-2 will be executed.
- If test expression-1 is false, then Statement block-3 will be executed. Note that we may also use more if...else statement as a part of Statement block-3.

Note: The structure shown above is just for example. A user may change this structure as per his/her needs. For example the below structure is also valid.

```
if (test expression-1)
{
    Statement block-1;
}
else
{
    if(test expression-2)
    {
        Statement block-2;
    }
    else
    {
        Statement block-3;
    }
}
```

Example 13.3 : Let us try to understand a C program to read three numbers from users and display the largest of three numbers using nested if statement. Figure 13.7 gives the code listing of the example 13.3 while figure 13.8 shows its output.



```

10-3.c - SciTE
File Edit Search View Tools Options Language Buffers Help
10-3.c

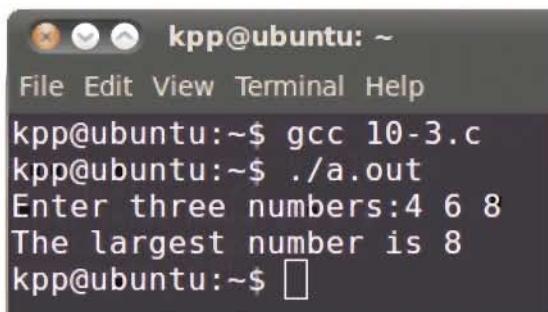
/* Example 3: Program to demonstrate the use of nested if statement. */
#include<stdio.h>

int main()
{
    int number1, number2, number3;      //declares three variables
    printf("Enter three numbers:");     // message to the user
    scanf("%d %d %d", &number1, &number2, &number3); //read 3 numbers

    if ( number1 > number2 )           // test condition 1
    {
        if ( number1 > number3 )       // test condition 2
            printf("The largest number is %d \n", number1);
        else
            printf("The largest number is %d \n", number3);
    }
    else
    {
        if ( number3 > number2 )       // test condition 3
            printf("The largest number is %d \n", number3);
        else
            printf("The largest number is %d \n", number2);
    }
    return 0;
}
/* End of Program.*/

```

Figure 13.7 : Code listing of Example 13.3



```

kpp@ubuntu: ~
File Edit View Terminal Help
kpp@ubuntu:~$ gcc 10-3.c
kpp@ubuntu:~$ ./a.out
Enter three numbers:4 6 8
The largest number is 8
kpp@ubuntu:~$ 

```

Figure 13.8 : Output of Example 13.3

Explanation

The first statement inside `main()` will declare three variables. Second statement will display message to user. The third statement will allow users to enter three numbers which are stored in `number1`, `number2` and `number3` variables. Fourth statement (test condition 1) will check whether value stored in `number1` is greater than value stored in `number2` or not. If test condition1 evaluates to true the inner if-statement (test condition 2) will be executed to check whether `number1 > number3` or not. Depending on true/false evaluation of test condition2, the inner if-statement message will be displayed to user. But when fourth statement (test condition 1) evaluates to false in that case the else part of outer if statement will be executed. Here the test condition 3 checks whether `number3 > number2` or not, and depending on true/false evaluation the message will be displayed to the user.

The else-if ladder statement

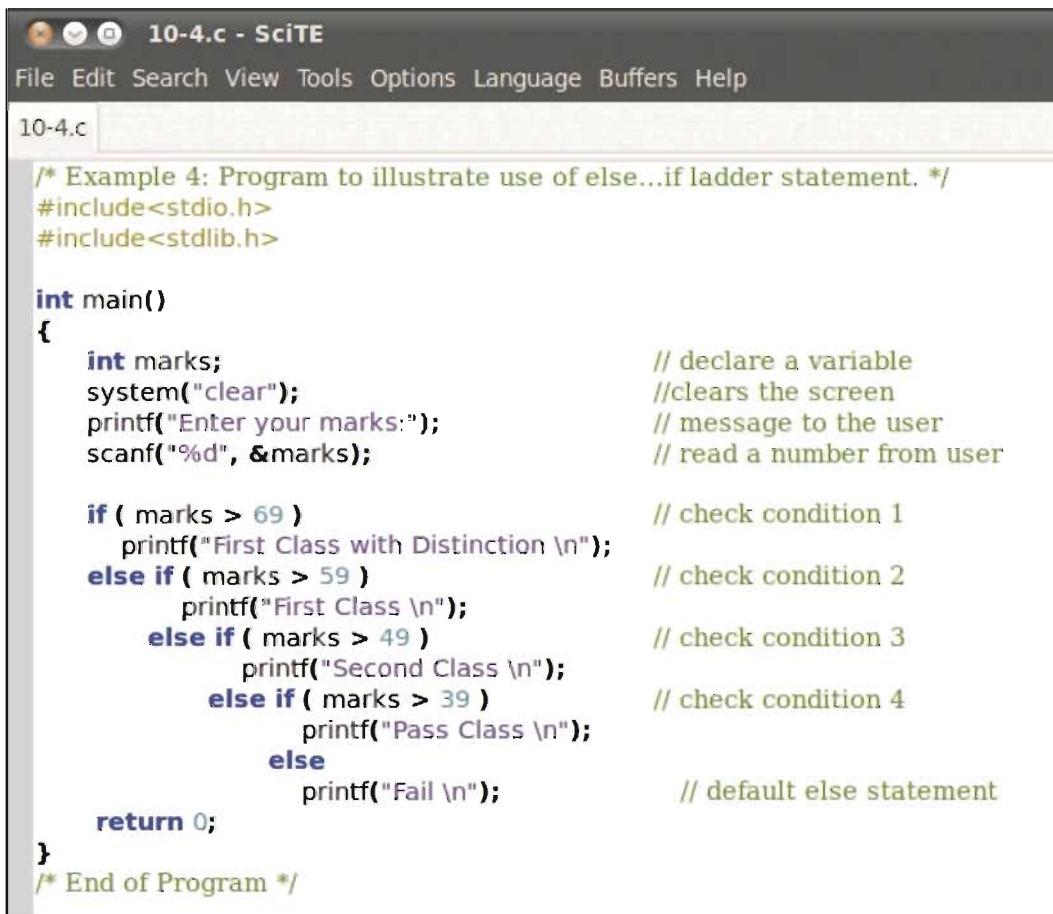
There is another way of using many if statements together when we want to evaluate multiple decisions. When chain of if statements are used in else (false) block of nested if statements, it becomes else-if ladder statement.

Let us use the else-if ladder statement to give class/grade of student based on given subject marks. We will follow the grading rules given in table 13.1 during our program:

Range of Marks	Class/Grade
70-100	First Class with Distinction
60-69	First Class
50-59	Second Class
40-49	Pass Class
0-39	Fail

Table 13.1 : Sample Grading Rules

Example 13.4: Let us try to understand a C program to read marks from the user and assign class/grade to the student based on entered marks. Figure 13.9 gives the code listing of the example 13.4 while figure 13.10 shows two runs of its output.



The screenshot shows the SciTE IDE interface with the file '10-4.c' open. The code implements an else-if ladder to determine a student's class/grade based on their marks. It includes comments explaining each step of the ladder.

```
/* Example 4: Program to illustrate use of else...if ladder statement. */
#include<stdio.h>
#include<stdlib.h>

int main()
{
    int marks; // declare a variable
    system("clear"); //clears the screen
    printf("Enter your marks:"); // message to the user
    scanf("%d", &marks); // read a number from user

    if ( marks > 69 ) // check condition 1
        printf("First Class with Distinction \n");
    else if ( marks > 59 ) // check condition 2
        printf("First Class \n");
    else if ( marks > 49 ) // check condition 3
        printf("Second Class \n");
    else if ( marks > 39 ) // check condition 4
        printf("Pass Class \n");
    else
        printf("Fail \n"); // default else statement

    return 0;
}
/* End of Program */
```

Figure 13.9 : Code listing of Example 13.4

```

kpp@ubuntu: ~
File Edit View Terminal Help
Enter your marks:80
First Class with Distinction
kpp@ubuntu:~$ 

kpp@ubuntu: ~
File Edit View Terminal Help
Enter your marks:55
Second Class
kpp@ubuntu:~$ 

```

Figure 13.10 : Output of Example 13.4

Explanation

First, second, third and fourth statement after main() declares a variable, clears the screen, display message and read marks from the user respectively. Fifth statement checks the first condition of the program that is marks > 69. If condition 1 evaluates to true then message "First Class with Distinction" will be displayed and then program ends. If condition 1 evaluates to false then condition 2 will be evaluated. If condition 2 is false then condition 3 is checked and so on. When all four conditions are evaluated to false in that case last default else statement block will be executed displaying message "Fail".

Syntax of else-if ladder statement

```

if (test expression-1)
    Statement block-1;

else if (test expression-2)
    Statement block-2;

else if (test expression-3)
    Statement block-3;
    ....
    ....
else if (test expression-n)
    Statement block-n;
else
    Default-statement-block;
program-statement-x;

```

This construct is known as else-if ladder. The test expressions are evaluated from the top to bottom in the ladder. When any test expression in a ladder evaluated to true, the associated statement block is executed. After which control is transferred to the program-statement-x skipping rest of the ladder. When all the test expressions are evaluated to false then control is transferred to the Default-statement-block of final else. Figure 13.11 shows flowchart of else-if ladder statement.

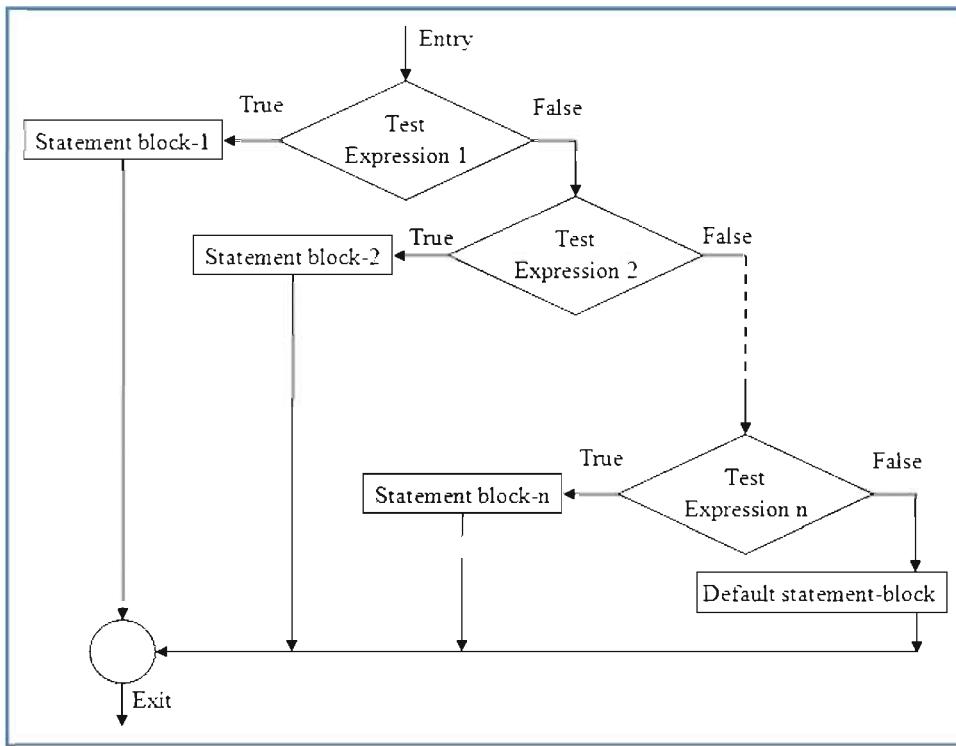


Figure 13.11 : Flowchart of else-if ladder statement

The Switch Statement

We have used the if...else statement for selection of various statement blocks during program execution. However when we use number of if...else statement then our program becomes more complex. The program becomes difficult to read and interpret the logic. To simplify our program, the C language has offered the built-in facility of multiway decision statement called switch. When an action is to be taken based on given multiple choices, switch statement is very useful.

Syntax of the Switch Statement

```

switch (expression)
{
    case value-1:
        statement block-1
        break;
    case value-2:
        statement block-2
        break;
    ....
    ....
    case value-n:
        statement block-n
        break;
    default:
        default statement block
        break;
}
statement-x;

```

Note the following important points related to switch statement :

- The switch statement uses one argument (expression or variable value) which is checked for equality with number of case options listed inside the switch statement block. The argument should be either a variable or expression of type integer or character.
- Each case option contains constant or constant expressions. This constant is known as case label and it ends with colon (:). Each case label must be unique. Any two case labels cannot have the same value.
- Statement block-1, statement block-2 and so on are statement lists which may contain zero or more statements. There is no need to use braces around this statement blocks.
- When switch statement is executed, it first evaluates the value of expression and then compares it with case constants from top to bottom. When a case found whose value is matching with the value of expression, then the corresponding statement block of that case is executed.
- After each statement block in switch, break statement signals the end of case causing the exit from switch statement. Then after flow of control is transferred to the next statement (statement-x) in a program.
- The default is an optional case. When *default* is written and no case found during execution at that time the default statement block is executed. The *default* can be used only once and may be placed anywhere inside switch but usually we place it at last inside the switch.

The execution of switch statement is illustrated in the flowchart shown in Figure 13.12. In the figure it is assumed that each case has break statement as the last statement inside block.

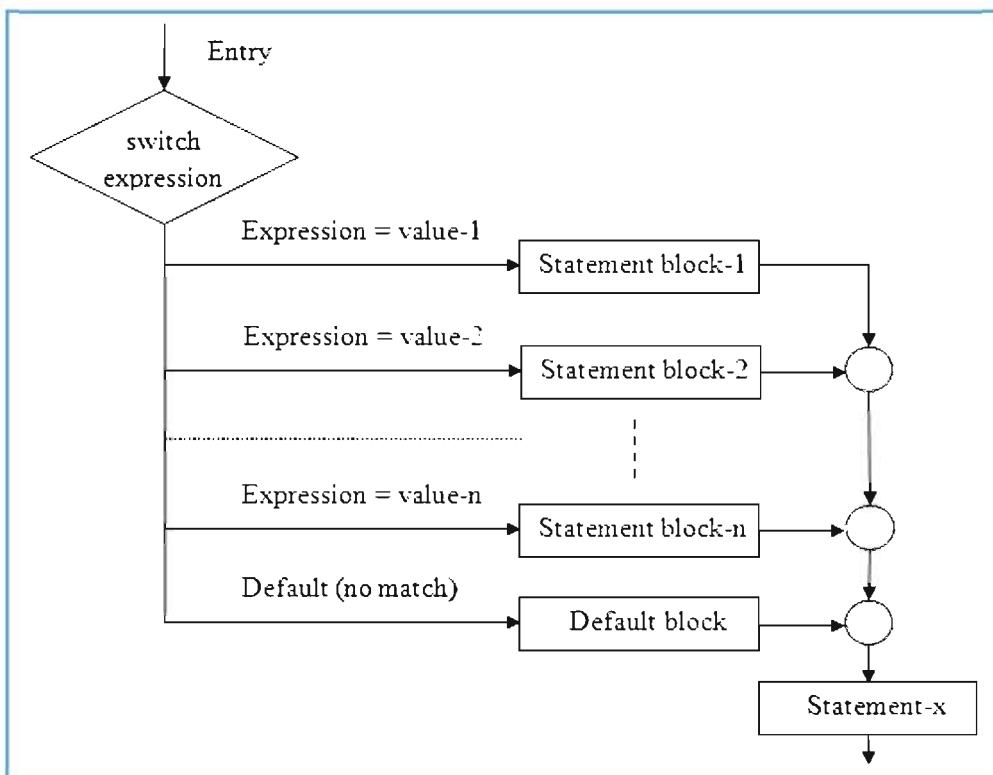
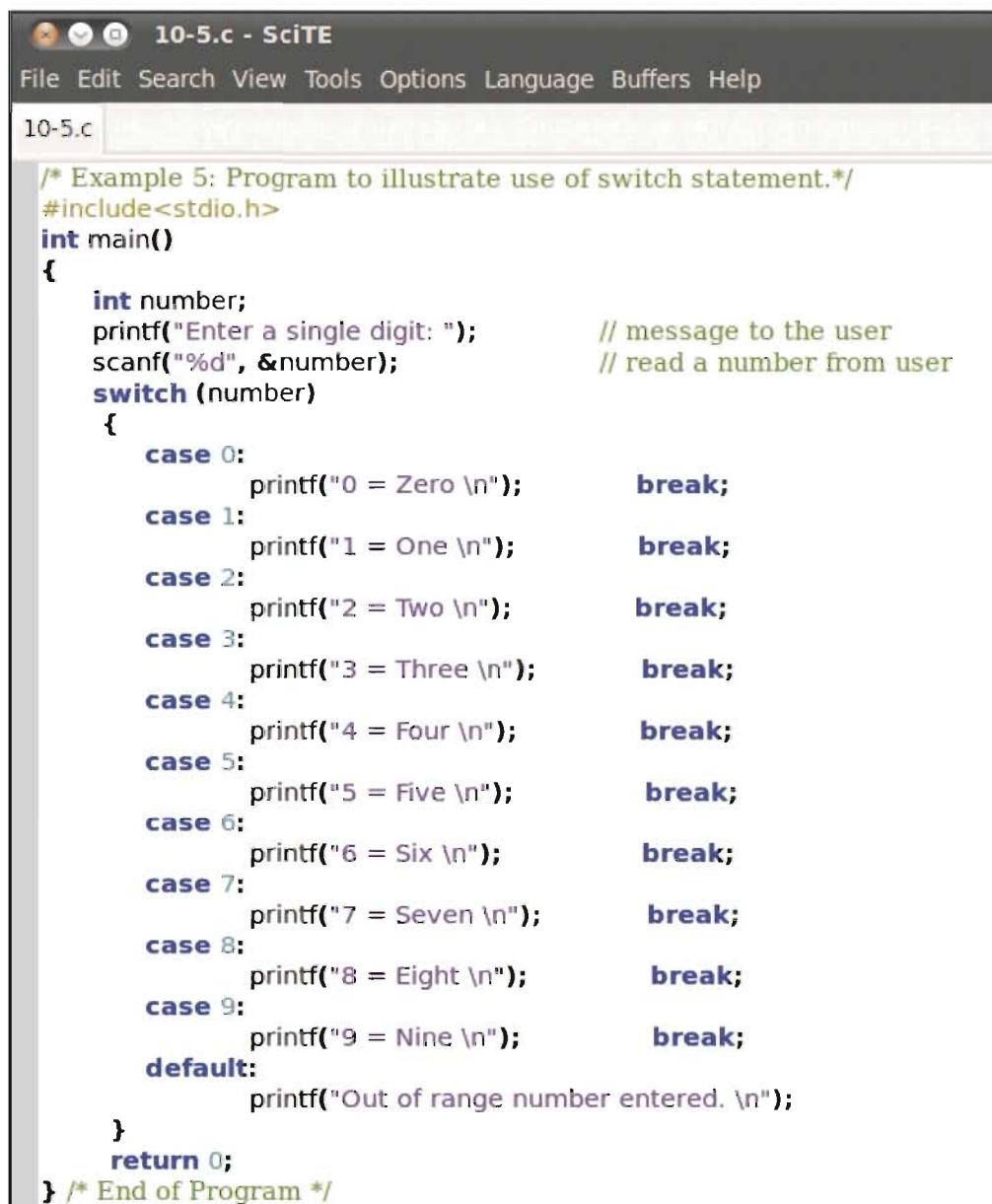


Figure 13.12 : Flowchart of switch statement

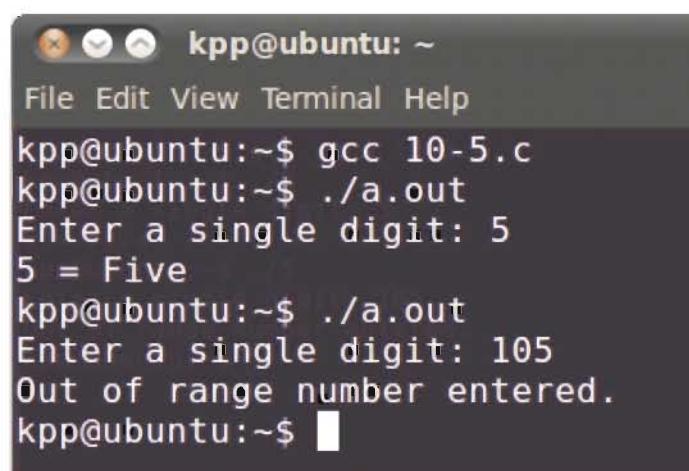
Example 13.5: Let us try to understand a C program to convert a given single digit (0 to 9) into its corresponding word representation. Figure 13.13 gives the code listing of the example 13.5 while figure 13.14 shows its output.



The screenshot shows the SciTE code editor with the title bar "10-5.c - SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, and Help. The code window displays a C program named 10-5.c. The code uses a switch statement to map digits 0-9 to their names. It includes comments explaining the code's purpose and structure.

```
/* Example 5: Program to illustrate use of switch statement.*/
#include<stdio.h>
int main()
{
    int number;
    printf("Enter a single digit: ");           // message to the user
    scanf("%d", &number);                      // read a number from user
    switch (number)
    {
        case 0:
            printf("0 = Zero \n");             break;
        case 1:
            printf("1 = One \n");              break;
        case 2:
            printf("2 = Two \n");              break;
        case 3:
            printf("3 = Three \n");             break;
        case 4:
            printf("4 = Four \n");              break;
        case 5:
            printf("5 = Five \n");              break;
        case 6:
            printf("6 = Six \n");               break;
        case 7:
            printf("7 = Seven \n");             break;
        case 8:
            printf("8 = Eight \n");              break;
        case 9:
            printf("9 = Nine \n");               break;
        default:
            printf("Out of range number entered. \n");
    }
    return 0;
} /* End of Program */
```

Figure 13.13 : Code listing of Example 13.5



The screenshot shows a terminal window on an Ubuntu system. The prompt is "kpp@ubuntu: ~". The user runs "gcc 10-5.c" and then "./a.out". They enter the digit "5", which is mapped to "5 = Five". When they enter "105", it is correctly identified as "Out of range number entered.".

```
kpp@ubuntu:~$ gcc 10-5.c
kpp@ubuntu:~$ ./a.out
Enter a single digit: 5
5 = Five
kpp@ubuntu:~$ ./a.out
Enter a single digit: 105
Out of range number entered.
kpp@ubuntu:~$
```

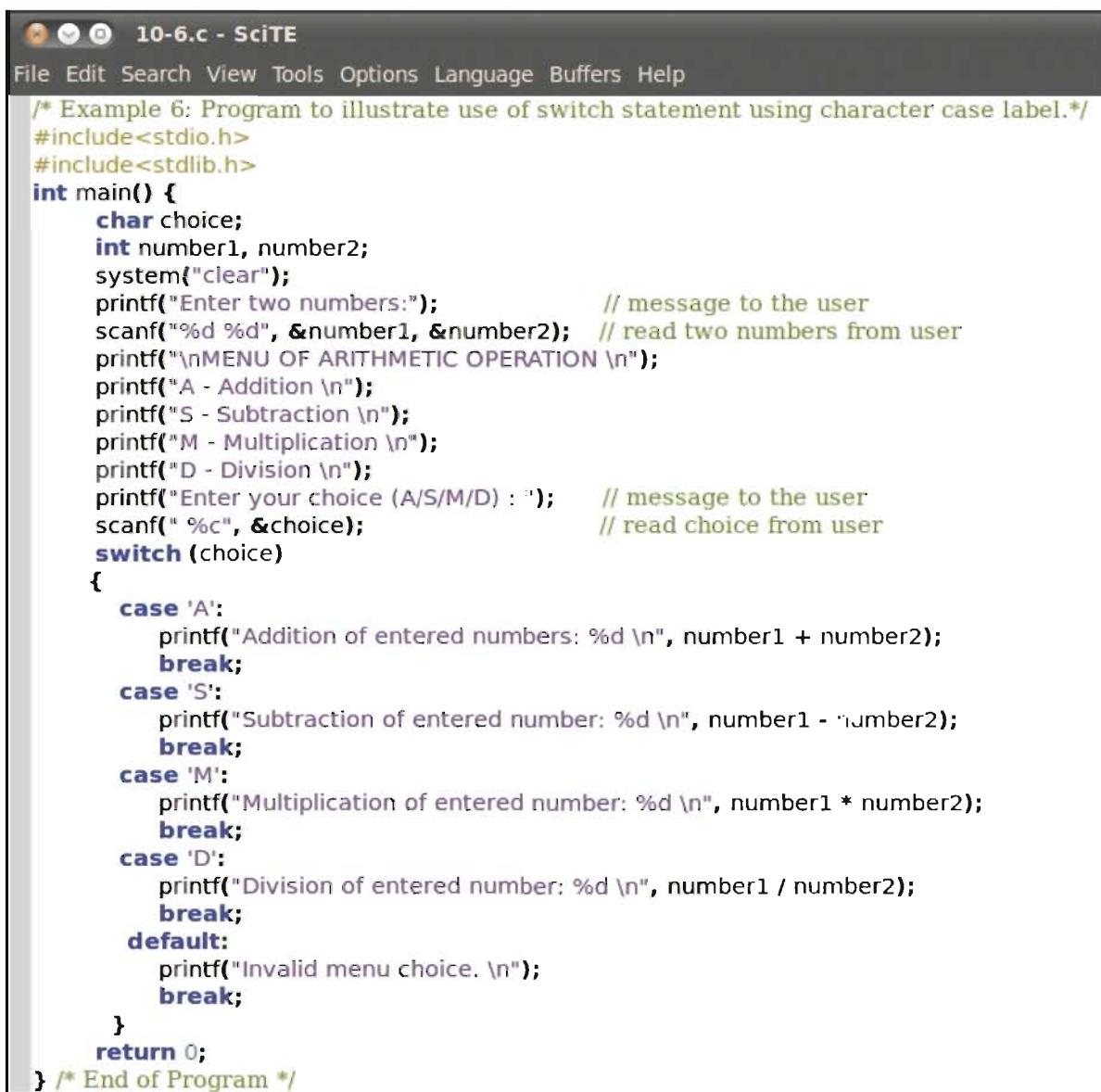
Figure 13.14 : Output of Example 13.5

Explanation

First, second and third statement after main() declares a variable, display message and read a digit from user respectively. Fourth statement of switch uses argument as a value stored in variable named number. Based on value stored in number, the corresponding case block will be executed. If no case found its matching value then default block will display the message "Out of range number entered." and program ends.

Now Let us try to understand how to use character constant as a part of case label in the switch statement using Example 13.6.

Example 13.6 : Write a C program to read two numbers from user and ask the choice of arithmetic operator using menu. Depending on the choice given performs arithmetic operations on given numbers and display the result. Figure 13.15 gives the code listing of the example 13.6 while figure 13.16 shows its output.



The screenshot shows the SciTE IDE interface with the title bar '10-6.c - SciTE'. The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, and Help. The code editor displays the following C program:

```
/* Example 6: Program to illustrate use of switch statement using character case label.*/
#include<stdio.h>
#include<stdlib.h>
int main() {
    char choice;
    int number1, number2;
    system("clear");
    printf("Enter two numbers:");           // message to the user
    scanf("%d %d", &number1, &number2);    // read two numbers from user
    printf("\nMENU OF ARITHMETIC OPERATION \n");
    printf("A - Addition \n");
    printf("S - Subtraction \n");
    printf("M - Multiplication \n");
    printf("D - Division \n");
    printf("Enter your choice (A/S/M/D) : "); // message to the user
    scanf(" %c", &choice);                // read choice from user
    switch (choice) {
        case 'A':
            printf("Addition of entered numbers: %d \n", number1 + number2);
            break;
        case 'S':
            printf("Subtraction of entered number: %d \n", number1 - number2);
            break;
        case 'M':
            printf("Multiplication of entered number: %d \n", number1 * number2);
            break;
        case 'D':
            printf("Division of entered number: %d \n", number1 / number2);
            break;
        default:
            printf("Invalid menu choice. \n");
            break;
    }
    return 0;
} /* End of Program */
```

Figure 13.15 : Code listing of Example 13.6

```

kpp@ubuntu: ~
File Edit View Terminal Help
Enter two numbers:20 30

MENU OF ARITHMETIC OPERATION
A - Addition
S - Subtraction
M - Multiplication
D - Division
Enter your choice (A/S/M/D) : A
Addition of entered numbers: 50
kpp@ubuntu:~$ 

```

Figure 13.16 : Output of Example 13.6

Explanation

The fourth and fifth statement after main() will display message and reads two numbers from user respectively. Statement number six to ten will display menu options on screen. In eleventh and twelfth statement we display message and read menu choice from the user respectively. Based on the choice given by user, the corresponding case block will be executed. If no case found its matching value then default block will display the message "Invalid menu choice." and program ends. Note that in this program you may add additional statement at appropriate location to verify special arithmetic conditions like division by zero error.

Compound Relational Test

The C language provides the necessary mechanisms to perform compound relational tests (multiple test expression). The compound relational test is nothing but simple one or more relational tests joined together by either the logical AND operators or logical OR operators. These logical operators are represented by the character pairs `&&` and `//` respectively. Use of compound relational test will help us to reduce number of if...else statements in our program.

Summary

In this chapter, we have learnt the special kind of C language facility that can change the flow of a sequence of instructions in the program. We had learnt that how two basic types of decision structure statements: if and switch can be used to change the flow of instructions in the program.

EXERCISE

1. What is the disadvantage of if statement as compared to if...else statement ?
2. What do you mean by nested if statement ?
3. Give suitable example in which the use of switch statement is advisable.
4. State the importance of break statement in the switch construct.
5. Explain the use of else-if ladder statement with suitable example.

6. State True or False :

- (a) We can write if statement within switch statement.
 - (b) When test expression of if statement is evaluated to false, the statements inside body of if are skipped.
 - (c) The default case statement is mandatory in a switch statement.
 - (d) The default case must be the last case in the switch statement.
 - (e) The break statement stops execution of program.
 - (f) We can write switch statement within if statement.

7. Choose the correct option from the following :

(1) What is the value of variable flag after the execution of following if statement ?

```
int flag=0;
```

```
if(5 < 8) { flag=1; }
```


(2) What is the value of variable 's' after the execution of following switch statement ?

$$x = 3;$$

```
switch ( x ) {
```

case 1: s = 'A'; break;

case 2: s = 'B'; break;

case 3: s = 'C'; break;

default: s = 'D'; break;

}

(3) What is the value of variable sum after the execution of following statements ?

```
int number1=5, number2=10, sum=0;
```

```
if (number1 > number2)
```

```
sum = sum + number1;
```

else

```
sum = sum + number2;
```

(4) What will be the output when following program segment is executed ?

```
int number1=10, number2=20;  
  
if ( ( number1+number2 ) > 35 || (number1 > number2) )  
    printf("%d", number1);  
  
else  
    printf("%d", number2);
```


(5) What will be the output when following program segment is executed ?

```
char chr='A';  
switch (chr)  
{  
    case 'A': printf("A"); break;  
    case 'B': printf("B"); break;  
    case 'C': printf("C"); break;  
}
```


LABORATORY EXERCISE

Write a C program to perform following tasks :

1. Check whether given number is positive or negative.
 2. Read age of person and display message whether person is eligible to vote or not.
 3. Read three numbers from users and display the smallest of three numbers using nested if statement.
 4. Check whether entered number is odd or even.
 5. Check whether entered character is Vowel or not by using switch statement.



Loop Control Structures

In a C program there may be situations where we may want to execute same block of statements many times. All programming languages offers loop control structure (also known as **looping**) allowing programmers to execute a statement or group of statements multiple times. In this chapter we will discuss the loop (repetitive) control structure provided by the C programming language. Here 'control structure' mean that the flow of execution may not be sequential and control may be transferred to any statement based on given conditions in our program.

In looping, the sequences of statements are executed until some exit condition is satisfied. The looping construct is composed of two parts : *body of loop and control statement*. Depending on the place of control statement in loop, it can be classified as **entry controlled** and **exit controlled** loop. In entry controlled loop the exit/control condition is checked before executions of statements inside loop body. In exit controlled loop the exit/control condition is checked after executions of loop body. This means that in case of exit controlled loop, body will be executed at least once before exiting from the loop. Further discussion related to the same is given during explanation of different types of loop.

Now let us discuss syntax and general rules related to following three basic loop control structure provided by C language :

- for
- while
- do...while

The For Loop

The *for* loop is normally used when we want to execute block of statements fixed number of times. To make *for* loop more dynamic, we can use the exit condition inside for loop. Let us try to understand the use of simple *for* loop statement by example14. 1. This example displays five " * " using simple *for* loop statement. Figure 14.1 gives the code listing and output of example 14.1.

The screenshot shows the SciTE IDE interface. The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, Help. The title bar says "11-1.c - SciTE". The code editor window contains the following C code:

```
/* Example 1: Program to illustrate simple for loop statement.*/
#include<stdio.h>

int main()
{
    int count; // declaration of variables
    for ( count = 0; count < 5 ; count++ ) // for loop body repeats 5 times
        printf(" * ");
    return 0;
}
/*End of Program */
```

To the right of the code editor is a terminal window showing the command-line session and its output:

```
gcc 11-1.c
>gcc 11-1.c
>Exit code: 0
./a.out
>/a.out
* * * * * >Exit code: 0
```

Figure 14.1 : Code listing and output of Example 14.1

Explanation

In the beginning of program one integer variable count is declared. Variable count is used as a counter variable in a for loop. Firstly variable count is initialized to zero. Then second expression (count < 5) is checked. When it is evaluated to true, the body of for loop is executed. In our case one " * " will be displayed on screen using printf statement. Then third for loop expression (count++) will be executed, incrementing value of count by 1. Again second expression (count<5) is evaluated and depending on its result, body of for loop will be executed. In our case five times body part will be executed displaying five stars " * * * * " on the screen.

Syntax of For Loop

```
for ( expression1; expression2; expression3 )
{
    Statement-block;
}
```

The header of *for* loop contains three expressions separated by semicolons in parenthesis. All these expressions are optional. Statement-block which is also known as body of *for* loop may contain simple statement or compound statement. Following figure 14.2 shows the flowchart of *for* loop.

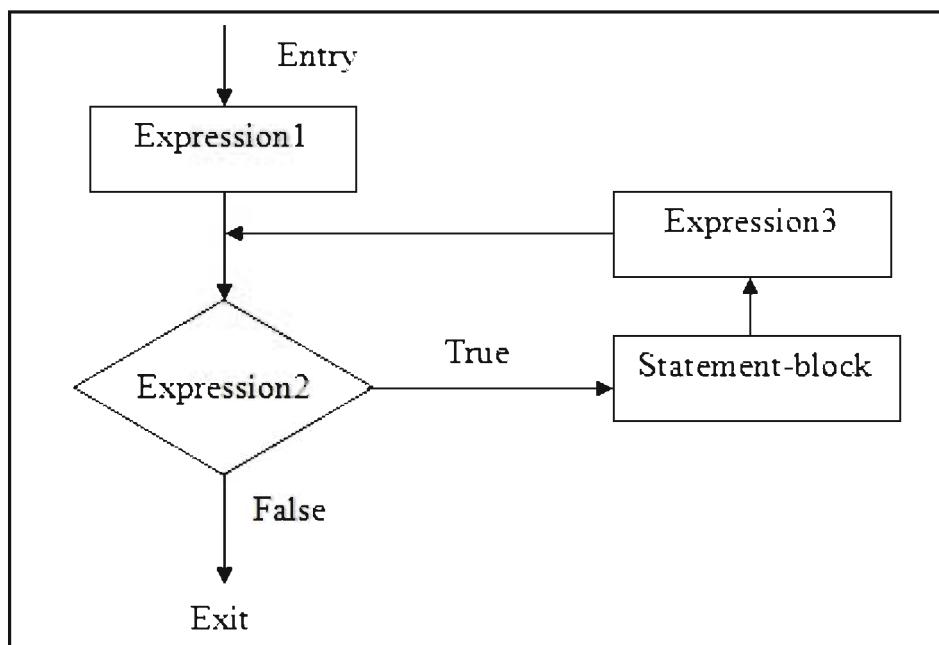


Figure 14.2 : Flowchart of for loop

As shown in figure14.2, the steps of executing for loop are as follows :

- Step 1** : Evaluate expresison1. The expression1 will be evaluated only once when the for loop starts.
- Step 2** : Evaluate expresison2. If its value is false then terminate the loop.
- Step 3** : If expression2 is evaluated to true then execute the statements in the body of loop.
- Step 4** : Evaluate expression3.
- Step 5** : Go to Step 2.

Note following important points related to for loop :

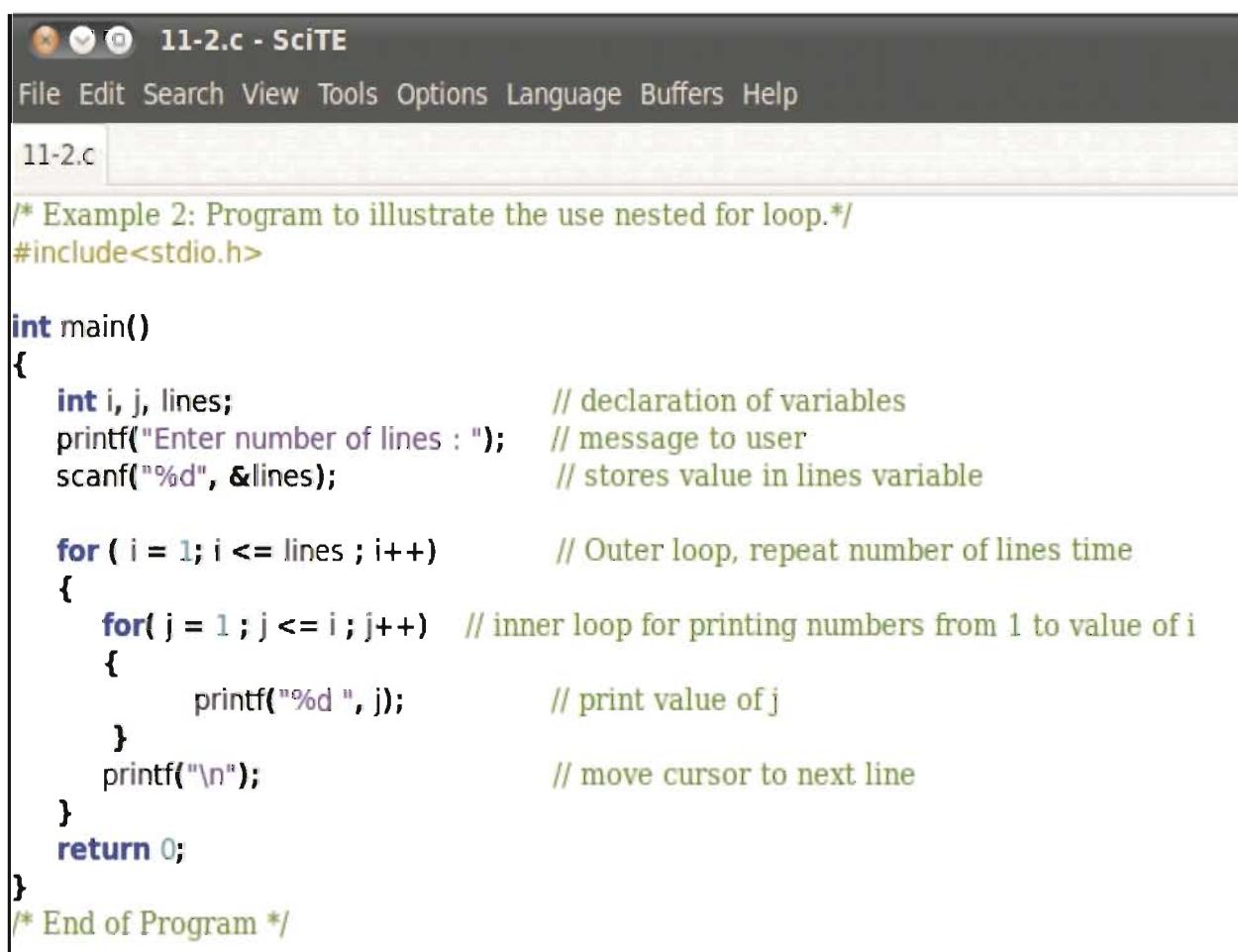
- expression1 works for initializing value of counter variable that is used to control the number of times a loop is to be executed. This variable is known as control variable.
- expression2 works as test condition. A control variable is used for checking loop terminating criteria.
- expression3 works for incrementing or decrementing value of control variable.

Nested for loop

Nested for loop means using one for loop within another for loop. Let us try to understand this concept using example14.2. This example prints the following pattern for given number of lines. For example if number of lines are 4 then display output as follow:

```
1  
1 2  
1 2 3  
1 2 3 4
```

Figure 14.3 gives the code listing of the example 14.2 while figure 14.4 shows its output.



The screenshot shows the SciTE IDE interface with the title bar "11-2.c - SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, and Help. The current file is "11-2.c". The code listed is as follows:

```
/* Example 2: Program to illustrate the use nested for loop.*/  
#include<stdio.h>  
  
int main()  
{  
    int i, j, lines; // declaration of variables  
    printf("Enter number of lines : "); // message to user  
    scanf("%d", &lines); // stores value in lines variable  
  
    for ( i = 1; i <= lines ; i++) // Outer loop, repeat number of lines time  
    {  
        for( j = 1 ; j <= i ; j++) // inner loop for printing numbers from 1 to value of i  
        {  
            printf("%d ", j); // print value of j  
        }  
        printf("\n"); // move cursor to next line  
    }  
    return 0;  
}  
/* End of Program */
```

Figure 14.3 : Code listing of Example 14.2

```

kpp@ubuntu: ~
File Edit View Terminal Help
kpp@ubuntu:~$ gcc 11-2.c
kpp@ubuntu:~$ ./a.out
Enter number of lines : 4
1
1 2
1 2 3
1 2 3 4
kpp@ubuntu:~$ 

```

Figure 14.4 : Output of Example 14.2

Explanation

The program contains outer for loop with control variable *i*, which is executed 'lines' number of times by varying value of *i* 1 by 1 up to the value of lines. For each *i* in the outer loop, inner loop with control variable *j* is executed *i* times, and display numbers varying 1 by 1 up to *i*. After printing *i* numbers in line *i*, printf("\n") statement of outer loop will bring cursor on the next line. This enables the next line number to be printed in new line.

Use of Comma Operator in for Loop

In for loop we can initialize more than one parameters using comma operator. Similarly we may use comma operator to increment or decrement various variables in a for loop. Let us try to understand the use of comma operator using example14.3. Figure 14.5 gives the code listing of the example 14.3 while figure 14.6 shows its output.

```

11-3.c - SciTE
File Edit Search View Tools Options Language Buffers Help
11-3.c

/*
 * Example 3: Program to illustrate the use comma operator in a for loop
 * to print the addition table.*/
#include<stdio.h>
int main()
{
    int i, j, number; // declaration of variables
    printf("Enter number of lines in a table: "); // message to user
    scanf("%d", &number); // stores value in number variable
    for ( i=0, j=10; i < number; i++, j- )
    {
        printf("%d + %d = %d\n", i, j, i + j );
    }
    return 0;
}
/* End of Program */

```

Figure 14.5 : Code listing of Example 14.3

```

kpp@ubuntu: ~
File Edit View Terminal Help
kpp@ubuntu:~$ gcc 11-3.c
kpp@ubuntu:~$ ./a.out
Enter number of lines in a table: 5
0 + 10 = 10
1 + 9 = 10
2 + 8 = 10
3 + 7 = 10
4 + 6 = 10
kpp@ubuntu:~$ 

```

Figure 14.6 : Output of Example 14.3

Note that in example 14.3 we have initialized two variables $i=0$ and $j=10$. Also we are incrementing i with value one and decrementing j with value 1 in the same *for* loop. Based on the input given we will get output as shown in the example.

The While Loop

When number of iteration cannot be pre-determined and when loop terminating condition is to be tested before entering the loop, at that time use of while loop is more suitable. For example,

- Find sum of all numbers entered until user enters zero
- Display menu options and take proper action until user select exit option

Syntax of While Loop

```

while (test expression)
{
    statement-block; /* body of while loop */
}

```

Flow chart of execution of while loop is shown in figure14.7.

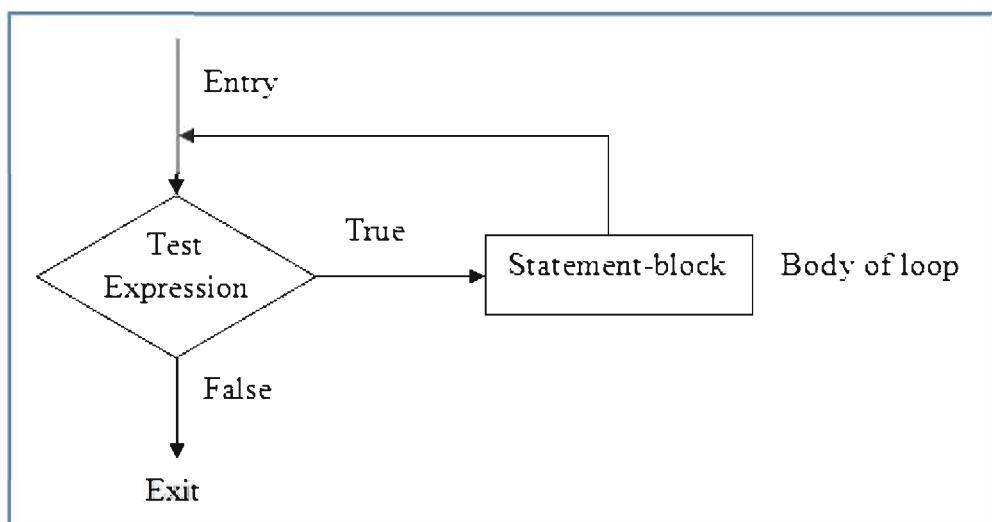
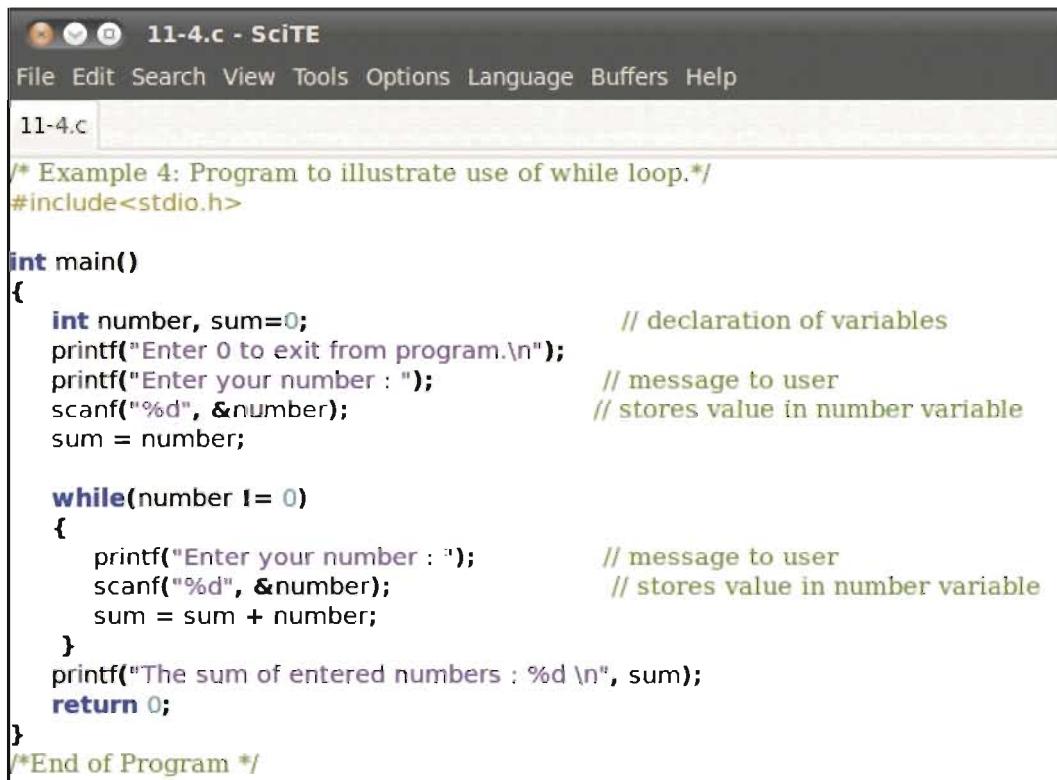


Figure14.7 : Flowchart of while loop

As seen in the figure14.7, it evaluates the test expression first. If test expression is evaluated to true, then body of loop containing statement-block is executed. Again program evaluates the test expression. This process is repeated until the test expression is evaluated to false. When test expression is evaluated to false then loop is terminated and control is passed to the next statement after the loop in a program. Body of loop may contain simple or compound statement.

Note that as we are checking condition at the entry point, it is known as *entry-controlled* loop. If test expression is evaluated to false, the body part will not be executed at all.

Let us try to understand the use of *while* loop using example 14.4. The program given in example14.4 finds sum of all numbers entered until user enter zero. Figure 14.8 gives the code listing of the example 14.4 while figure 14.9 shows its output.

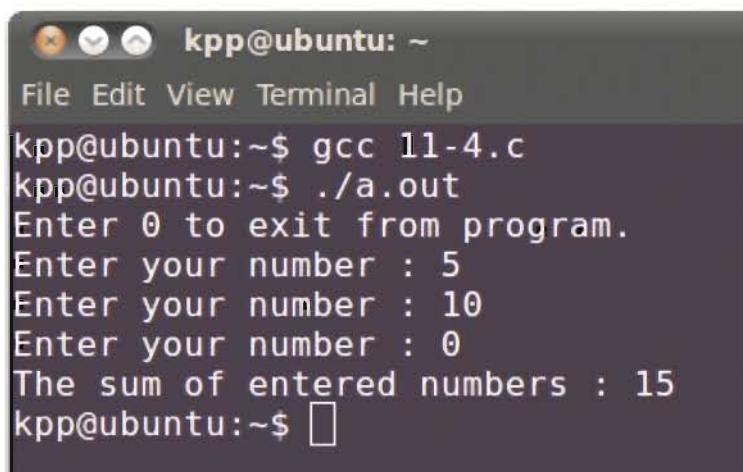


```
11-4.c - SciTE
File Edit Search View Tools Options Language Buffers Help
11-4.c
/* Example 4: Program to illustrate use of while loop.*/
#include<stdio.h>

int main()
{
    int number, sum=0;                                // declaration of variables
    printf("Enter 0 to exit from program.\n");
    printf("Enter your number : ");
    scanf("%d", &number);                            // message to user
    sum = number;                                    // stores value in number variable

    while(number != 0)
    {
        printf("Enter your number : ");
        scanf("%d", &number);
        sum = sum + number;
    }
    printf("The sum of entered numbers : %d \n", sum);
    return 0;
}
/*End of Program */
```

Figure 14.8 : Code listing of Example 14.4



```
kpp@ubuntu: ~
File Edit View Terminal Help
kpp@ubuntu:~$ gcc 11-4.c
kpp@ubuntu:~$ ./a.out
Enter 0 to exit from program.
Enter your number : 5
Enter your number : 10
Enter your number : 0
The sum of entered numbers : 15
kpp@ubuntu:~$
```

Figure 14.9 : Output of Example 14.4

Explanation

First statement after main function declares two variables required in the program. The second statement informs user about how to exit from the program. Third and fourth statement displays message and read a number from the user respectively. Fifth statement assigns the value of number variable to sum variable. In sixth statement, we check the test condition i.e. value stored in number variable is not equal to zero. When this test condition evaluates to true, statement number seven and eight displays message and read a number from the user respectively. The tenth number of statement will be adding the entered number into the sum variable. After this again test condition of *while* loop will be evaluated. When test condition is evaluated to false, the program executes the tenth statement displaying the sum of all the numbers entered by the user.

The do...while loop

We have seen that in *while* loop, the test expression is checked before executing body of loop. There are certain cases where we may want to execute body of loop before test expression. The do...*while* loop should be used when the test expression is to be checked after executing body of loop. As we are checking test condition at the end of loop, the do...*while* loop is a kind of *exit-controlled* loop. Note that in *do...while* loop, body of loop will be executed at least once.

Syntax of do...while loop

```
do
{
    statement-block;      /* body of loop */
}
while( test expression);
```

Flow chart of *do...while* loop is shown in figure14.10.

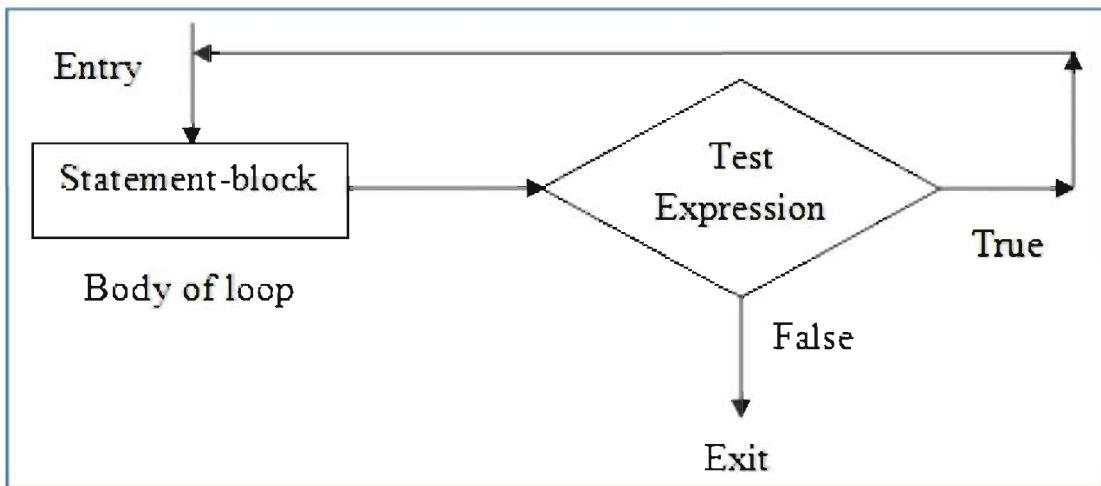


Figure 14.10 : Flowchart of do...while loop

As seen in the figure14.10 the body of loop containing statement-block is executed first. Then it evaluates the test expression. If the test expression is evaluated to true then body of loop containing statement-block is executed again. After executing statement-block, once again text expression is evaluated. This process is repeated until the test expression is evaluated to false. When test expression is evaluated to false then loop is terminated and control is passed to the next statement after the loop in a program. Body of loop may contain simple or compound statement.

Let us try to understand the use of `do...while` loop using example 14.5. Program given in the example 14.5 will find sum of all numbers entered by the user, until user enters zero using `do...while` loop. Figure 14.11 gives the code listing of the example 14.5 while figure 14.12 shows its output.

The screenshot shows the SciTE editor window with the title "11-5.c - SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, and Help. The file tab "11-5.c" is selected. The code listed is:

```
/* Example 5: Program to illustrate use of do...while loop.*/
#include<stdio.h>

int main()
{
    int number, sum=0; // declaration of variables
    printf("Enter 0 to exit from program.\n");
    do
    {
        printf("Enter your number : "); // message to user
        scanf("%d", &number); // stores value in number variable
        sum = sum + number;
    } while(number != 0);

    printf("The sum of entered numbers : %d \n", sum);
    return 0;
}
/*End of Program */
```

Figure 14.11 : Code listing of Example 14.5

The screenshot shows a terminal window titled "kpp@ubuntu: ~". The terminal prompt is "kpp@ubuntu:~\$". The user runs "gcc 11-5.c" and then "./a.out". The program asks for input, receives "3", "4", and "0", and then prints the sum "7".

```
kpp@ubuntu:~$ gcc 11-5.c
kpp@ubuntu:~$ ./a.out
Enter 0 to exit from program.
Enter your number : 3
Enter your number : 4
Enter your number : 0
The sum of entered numbers : 7
kpp@ubuntu:~$
```

Figure 14.12 : Output of Example 14.5

Explanation

First statement after main function declares two variables required in the program and initializes one of them. Second statement displays the message about how to exit from the program. At statement three the *do...while* loop starts. Statement fourth and fifth displays message and read a number from user respectively. Statement sixth adds number entered by user into sum variable. Statement seven evaluates the test condition. Statement fourth to sixth is executed repeatedly every time when test expression is evaluated to true. When test expression is evaluated to false, the program executes the eighth statement displaying the sum of all the numbers entered by the user.

Nested Loops

Nesting of different types of loops is possible by using one loop control structure within another, irrespective of the type of loop control structures used. For example, in *for* loop, *while* or *do...while* loop can be used. For example, display all prime numbers between 1 and 100. Loop for numbers can be written using *for* loop. Within this loop, while loop can be used to determine whether number is prime or not.

Selecting a Loop

So far in this chapter, we have discussed three looping construct for, while and do..while of C programming language. To select the more appropriate loop for our program, we may use the following steps :

- For a given problem, identify whether entry-controlled or exit-controlled loop is required.
- For entry controlled loop, we may use *for* or *while* loop construct. Use of *for* loop is advisable for counter based exit condition.
- For exit controlled loop, we have choice of *do..while* construct.

Also while using any loops in our program following three important points are to be considered carefully to avoid miscellaneous behavior of a loop:

- Initialization of loop counter
- To test exit condition to come out from the loop
- Incrementing or decrementing loop counter

Skipping a Part of Loop

During execution of a program, sometimes we may want to skip some of the statements or iterations of the loop. This is possible in C language by using following statements :

- (i) break statement
- (ii) continue statement

The break Statement

Sometimes during execution of any loop, we may want to terminate loop instantly without even execution of other statements of loop. This is possible using the break statement. The break statement in C programming language has following usages :

- When the break statement is encountered inside the loop, the loop is immediately terminated and program control executes the next statement following the loop.
- The break statement can also be used to terminate a case in the *switch* statement.

In case of nested loop, the *break* statement will stop the execution of currently executing loop and starts execution of next statement after the block of code.

Example14.6 shows how the *break* statement works in a program. Figure 14.13 gives the code listing as well as output of example 14.6.

The screenshot shows the SciTE IDE interface. The title bar says "11-6.c * SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, Help. The buffer tab shows "11-6.c *". The code area contains:

```
/* Example 6: Program to illustrate use of the break statement.*/
#include <stdio.h>

int main ()
{
    int number = 1;
    while( number < 10 )
    {
        printf("Value of number is %d\n", number);
        number = number + 1;
        if( number > 5)
        {
            break; // statement will terminate the loop.
        }
    }
    return 0;
}
/* End of Program*/
```

To the right of the code, the output window shows the command line and the program's execution:

```
gcc 11-6.c
>gcc 11-6.c
>Exit code: 0
./a.out
>./a.out
Value of number is 1
Value of number is 2
Value of number is 3
Value of number is 4
Value of number is 5
>Exit code: 0
```

Figure 14.13 : Code listing and output of Example 14.6

Explanation

First statement after main function initializes an integer variable. The test condition in *while* statement is written to execute loop nine times. The *printf* statement inside loop will print the value of number variable. After printing value of number variable it is incremented by 1. Next, *if* statement will be executed. But during execution of program when *if* statements is evaluated to true, the *break* statement executes; causing termination of while loop. Hence in our program, *printf* statement inside the while loop will be executed only five times.

The continue Statement

The *continue* statement works to some extent like the *break* statement. Instead of forcing termination of loop, however, *continue* statement forces the next iteration of the loop to take place, skipping any code in between.

For example, in for loop, *continue* statement causes the conditional test and increment portions of the loop to execute. For the while and do...while loops, *continue* statement causes the program control passes to the conditional tests.

Let us see example14.7 which shows how *continue* statement works in a program. Figure 14.14 gives the code listing and output of example 14.7.

The screenshot shows the SciTE IDE interface. The title bar says "11-7.c - SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, Help. The main window contains the C code for Example 7. The code uses a continue statement to skip the current iteration of a while loop when the number is 3. The output window shows the command "gcc 11-7.c" being run, followed by the program's output: "Value of number is 1", "Value of number is 2", "Value of number is 4", "Value of number is 5", and "Value of number is 6". The command ">>./a.out" is also shown.

```

/*
 * Example 7: Program to illustrate the use of continue statement.
 */
#include <stdio.h>

int main ()
{
    int number = 1;
    printf("Value of number is %d\n", number);

    while(number < 6)
    {
        number = number + 1;
        if( number == 3)
        {
            continue; // statement skip the current iteration of the loop
        }
        printf("Value of number is %d\n", number);
    }
    return 0;
}
/* End of Program*/

```

Figure 14.14 : Code listing and output of Example 14.7

Explanation

First statement declares and initializes a number variable required in the program. Second statement prints value of number variable. The test condition in while statement is written to execute loop five times. Inside while loop, the number is incremented by value 1. After this the if statement checks the test condition. When it is evaluated to true, the continue statement executes, which terminates the current iteration of while loop. Hence value of number 3 will not be printed. Program executes next iteration of loop causing program to print value of number variable 4, 5 and 6.

The Infinite Loop

Any loop in a program becomes infinite loop if it runs forever and program control never comes out of it. The loop becomes infinite due to non availability of exit condition in the logic of loop. Observe the program code logic of example 14.8 shown in figure 14.15.

The screenshot shows the SciTE IDE interface. The title bar says "11-8.c - SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, Help. The main window contains the C code for Example 8. It features a for loop with no termination condition, which results in an infinite loop. The output window shows the repeated message "This for loop will run forever....." indicating the loop is running indefinitely.

```

/*
 * Example 8: Program to illustrate infinite loop.
 */
#include <stdio.h>

int main ()
{
    for( ; ; )
    {
        printf(" This for loop will run forever....\n ");
    }
    return 0;
}
/* End of program*/

```

Figure 14.15 : Code listing and output of Example 14.8

Explanation

When we compile and execute this code it will runs forever. In C program *for* loop all three expressions are optional. When the conditional expression is absent inside *for* loop, it is assumed to be true. We may have initialization expression and increment expression inside for loop, but C program runs forever when we use `for(; ;)` construct. Note that you may use CTRL + C keys to terminate your infinite loop.

Summary

In this chapter we had learnt that how C languages offers **loop control structure** allowing programmers to execute a statement or group of statements multiple times. We had learnt three basic loop control structure (for, while and do...while) provided by the C language. We had discussed that how entry controlled and exit controlled loops are useful in our programs. We had also learnt that how nesting of various loops can be performed. At the end of chapter we had learnt that how during execution of a program; we can skip some of the statements or iterations of the loop using break and continue statements.

EXERCISE

1. Explain the syntax of for loop construct with suitable examples.
2. What do you mean by looping statement ? State use of looping statement in a C program.
3. What is the difference between while and do-while loop ?
4. Describe how to skip part or a loop using break and continue statement.
5. What do you mean by entry controlled and exit controlled loop ? Give suitable example for the same.
6. Fill in blanks :
 - (a) Depending on the place of control statement in loop, it can be classified as _____ controlled and _____ controlled loop.
 - (b) The for loop is example of _____ controlled loop.
 - (c) The do-while loop is example of _____ controlled loop.
 - (d) In exit controlled loop, body will be executed _____ time before exiting from the loop.
 - (e) The _____ statement causes immediate exit from the loop control structure.
 - (f) The _____ statement skips the subsequent statements of the loop and continues the next iteration of the loop.
7. State True or False :
 - (a) Within for loop do...while loop cannot be used.
 - (b) Execution of break statement inside the loop immediately terminates the execution of program.

- (c) Initialization, test condition and increment parts are optional in a for loop header.
- (d) The break statement cannot be used to terminate a case in the switch statement.
- (e) Initialization, test condition and increment parts are separated by a commas.
- (f) For every while clause, there must be a do.
- (g) For every do clause, there must be a corresponding while.

8. Choose the correct option from the following :

- (1)** How many times printf statement is executed in the following program segment ?

```
int num1 = 3, num2 = 6;
```

```
while(num1 < num2) {
```

```
    printf(" Hello Students...");
```

```
    num1 = num1 + 1;
```

```
}
```

(a) 1

(b) 2

(c) 3

(d) 6

- (2)** What is the output of following program segment ?

```
int a = 5, b = 10;
```

```
do
```

```
{
```

```
    a = a + 1;
```

```
}while( a <= b);
```

```
printf("%d", a);
```

(a) 10

(b) 9

(c) 11

(d) 15

- (3)** What is the output of following program segment ?

```
int main(){
```

```
    int i;
```

```
    for(i = 0; i < 10; i++);
```

```
    printf("%d", i);
```

```
}
```

(a) 0123456789

(b) 9

(c) 10

(d) Error

- (4) What is the value of a number variable after execution of following code ?

```
int number = 1;  
while(number < 5){  
    number = number + 1;  
    if( number == 3) {  
        continue;  
    }  
}
```


LABORATORY EXERCISE

- 1.** Change the following for loop into its equivalent while loop

- ```
(a) for(number=0; number <5; number++)
 printf("%d", number);
}
```

- (b) `for(number=5; number > 0; number- -) {  
 printf("%d", number);  
}`

- 2.** Write a program to demonstrate use of the compound statement within a while loop.

3. Write a program to find the sum of first 100 integers using for loop.

4. Demonstrate the use of while loop to find the sum of first 100 even integers.

- 5.** Write a program to print following pattern for a given number of lines. For example, if number of lines entered are 4 then out is as under:

\* \* \* \*

6. Write a for loop program for printing following series of numbers on screen.



7. Write a program to display ASCII value of uppercase and lowercase letters.

- 8.** Write a program to count and display the number of characters entered by user until user enters 'Z';





## Arrays

In C programming, so far we have used only basic data types, namely int, float, char, double, etc. These types of variables can store only one value at a time. To process one value we need one variable in a program. If we want to process five different values, we need five different variables. Now imagine the program where we want to handle very large amount of data. To handle such kind of situations, C supports a special kind of data type called an array. In this chapter we will learn how to use different types of an array available in C language.

### Need of Arrays

Imagine some of the following real life situations for which we want to write a C program :

- Maintain Marks of all the students of class
- Maintain Cost of all the products of Super Market
- List of employees and their contact numbers
- Maintain height and weight of each student of class

To handle marks of each student, instead of declaring different variables, such as marks0, marks1, marks2, marks3, marks4, .... and marks59, we may declare one array variable such as marks[60]. To refer individual student's mark now we can use variable like marks[0], marks[1], marks[2] ... and so on. In computer memory array of marks[59] will look like the one shown in figure 15.1.

| First element | Second element | Third element | .....    | ..... | Last element |
|---------------|----------------|---------------|----------|-------|--------------|
| marks[0]      | marks[1]       | marks[2]      | marks[3] | ..... | marks[59]    |

**Figure 15.1 : Sample Structure of Array in Memory**

Note following important points related to an arrays.

- It is a collection of elements having same data type.
- It is a fixed-size sequential collection of elements. An array element occupies contiguous memory locations.
- The element of an array is accessed by an index number. The index number contained within square brackets is also known as subscript.
- A subscript number must be an integer or an integer expression.
- The subscript number starts with zero.

### Types of Arrays

The arrays in C language can be classified in to following two categories :

- (i) Single or One dimensional array
- (ii) Multidimensional array

The single dimensional array may have one row or one column while multidimensional array has one or more rows and one or more columns. The number of rows and columns are specified by user as per the program requirement. We will discuss only one and two dimensional arrays offered by the C language.

### Declaration of Single Dimensional Array

To declare an array in C language, we specify the type of data which we want to store in the elements and the number of elements to be stored in an array as follows:

***datatype arrayname [ size ];***

- ***datatype:*** It specifies the type of elements that an array stores. If we want to store characters in an array then type of an array is 'char'. In the same way to store integers, type of an array is 'int'. The datatype can be any valid C language data type.
- ***arrayname:*** This is the name given to an array variable by the programmer. It can be any string but it is usually suggested that some meaningful name should be given to an array. The array name should be in context with what is being stored in that array in a program.
- ***size:*** The value of size indicates the number of elements the array can store. The size must be an integer constant greater than zero.

For example, to store marks of 60 students, we may use following statement :

***int marks[60];***

Note that the marks array declared here will be able to contain only 60 integer values. The example of arrays that can store characters, floating point values and very large values (double) is shown in the declarations below:

***char strings[20];***

***float percentages[20];***

***double numbers[20];***

### Assigning Values to Single Dimensional Arrays

C language allows array initialization in two different ways:

- Compile time array initialization
- Runtime array initialization

### Compile Time Array Initialization

An array can be assigned values similar to a normal variable at the time of declaration, the general syntax to set the values of various array elements is as follows:

***datatype arrayname[size] = { value1, value2, .... , valueN };***

Here *value1*, *value2*, ..., *valueN* provide the initial values for successive elements of the array. Following is the example where we specify size and set values of all elements of arrays.

***int marks[5] = {60,65,70,75,80};***

This will declare an array variable marks that can store five elements with the values set as: marks[0]=60, marks[1]=65, marks[2]=70, marks[3]=75 and marks[4]=80. During such declaration, if list of values are less than the size of an array, then only specified number of elements will be initialized and remaining elements will be initialized to zero. For example, the declaration

```
int marks[5] = {60,65,70};
```

will initialize first three array elements as marks[0] = 60, marks[1] = 65 and marks[2] = 70. The remaining two array elements marks[3] and marks[4] will be initialized to value zero.

If the array elements are assigned value at the time of declaration itself then specifying size is optional. If we don't specify the size, compiler will automatically calculate the size by counting total number of values written inside the curly bracket { }. For example, the following statement

```
int marks[] = {60,65,70,75,80};
```

will declare the marks array containing five elements whose values are initialized with the values given in the curly bracket.

Let us learn the concept of compile time initialization of one dimensional array with example15.1. Figure 15.2 gives the code listing and output of the example 15.1.

The screenshot shows the SciTE IDE interface. The title bar says "12-1.c - SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, Help. The main window displays the C code for Example 15.1. The code initializes an array 'number' with values 10, 20, 30, 40, 50, and then prints each element. To the right, a terminal window shows the command "gcc 12-1.c" being run, followed by the output of the program which prints the numbers 10, 20, 30, 40, and 50, and ends with "Exit code: 0".

```
/* Example 1: Program to illustrate compile time initialization of
one dimensional array*/

#include<stdio.h>
int main()
{
 int number[5] = {10,20,30,40,50}; // Compile time initialization
 int i;
 for(i = 0; i < 5 ; i++) // Loop to display array elements
 {
 printf(" %d \n", number[i]);
 }
 return 0;
}
/* End of Program*/
```

```
gcc 12-1.c
>gcc 12-1.c
>Exit code: 0
./a.out
>./a.out
10
20
30
40
50
>Exit code: 0
```

Figure 15.2 : Code listing and output of Example 15.1

### Explanation

The first line after main function declares an array called number and initializes all array elements with list of values given in curly bracket at compile time. The for loop will display the individual elements of a number array.

### Runtime Array Initialization

When we need to read data from user while executing our program, we can use runtime array initialization. The code given in example15.2 will allows user to enter marks of five different students at runtime of the program. The program also demonstrates how to use stored values from the array.

Figure 15.3 gives the code listing of the example 15.2 while figure 15.4 shows its output.

The screenshot shows the SciTE code editor with the file '12-2.c' open. The code is a C program that reads marks of five students and calculates the average. It includes comments explaining the purpose of each section of code. The code uses standard C syntax with `#include <stdio.h>`, `int main()`, and loops to read and sum marks.

```
/* Example 2: Program to illustrate runtime array initialization and accessing values from one dimensional array. Program will read marks of five students and display the average marks.*
```

```
#include<stdio.h>
int main()
{
 int marks[5], i, total=0;
 for(i = 0; i < 5; i++)
 {
 printf("Enter marks[%d] : ", i);
 scanf("%d", &marks[i]);
 }
 for(i = 0; i < 5; i++)
 {
 total = total + marks[i];
 }
 printf("\nThe average marks of students are %d. \n", total/5);
 return 0;
}
/* End of Program*/
```

Figure 15.3 : Code listing of Example 15.2

The screenshot shows a terminal window on Ubuntu. The user runs `gcc 12-2.c` and then executes the program with `./a.out`. The program prompts for five student marks (5, 10, 15, 20, 25) and then displays the average mark (15).

```
kpp@ubuntu: ~$ gcc 12-2.c
kpp@ubuntu: ~$./a.out
Enter marks[0] :5
Enter marks[1] :10
Enter marks[2] :15
Enter marks[3] :20
Enter marks[4] :25

The average marks of students are 15.
kpp@ubuntu: ~$
```

Figure 15.4 : Output of Example 15.2

### Explanation

The first `for` loop in the program will read marks of five students. The second `for` loop will find the total marks by reading individual elements of marks array. The last `printf` statement will display the average marks of the students.

### String as a Character Arrays

A string is a series of characters that is normally treated as a single block. C language does not

provide the *string* as inbuilt data type. However it allows us to represent string as a character array. A string variable is any valid C variable name and it is always declared as the character array. The general form of string declaration is:

```
char stringname[size];
```

Here the size indicates the number of characters in the variable called stringname. Followings are some of the valid string declarations:

```
char student_name[20];
```

```
char city[50];
```

```
char state[20];
```

Note that strings in a C language are terminated by the special character called null character ('\0'); this helps the program to identify the end of the string. Since strings are terminated by the null character '\0', we require one extra storage location in the character array to accommodate all the characters of strings.

Similar to numeric arrays, character arrays can also be initialized at compile time and at run time of a program. Character array initialization can be possible in following different manners:

```
char student_name[6] = "PURVA";
```

```
char student_name[6] = {'P', 'U', 'R', 'V', 'A', '\0'};
```

The reason behind keeping 6 elements in student\_name array is that the string PURVA contains five characters and one element is used for null character '\0'. Note that when we initialize the character array by listing its elements, it is necessary to specify null character '\0' at the end of string. We can also declare character arrays without specifying size as follow:

```
char state[] = {'G', 'U', 'J', 'A', 'R', 'A', 'T', '\0' };
```

In this case the character array state will be declared and initialized with eight elements. Using following statements we can refer previously discussed character arrays.

```
printf("The name of student is %s", student_name);
```

```
printf("The name of state is %s", state);
```

```
printf("The first character of your name is %c", student_name[0]);
```

## Multidimensional Array

The single dimensional array keeps track of information in linear order. However, the data associated with certain real life systems (such as digital image, a chess game board, matrix, etc.) are available in two dimensions. To think about this type of system in a program, we need a multi-dimensional data structure.

C language allows us to create multidimensional array. As mentioned earlier, multidimensional arrays have more than one rows and columns. We can create two dimensional, three dimensional, or N-dimensional array in C. The simplest form of the multidimensional array is the two-dimensional array. The two dimensional array have row index as well as column index. We will discuss only the two dimensional arrays in this chapter.

## Declaration of Two Dimensional Arrays

So far we had discussed one dimensional array that can store a list of values in it. There are many situations in real life where we may want to store data tables. Consider the case given in the table 15.1. It shows the details about sale of an item by different salesmen from Monday to Friday.

|           | Monday | Tuesday | Wednesday | Thursday | Friday |
|-----------|--------|---------|-----------|----------|--------|
| Salesman1 | 100    | 150     | 200       | 250      | 200    |
| Salesman2 | 200    | 250     | 300       | 350      | 300    |
| Salesman3 | 150    | 200     | 250       | 300      | 250    |

**Table 15.1 : Sales details**

The table contains a total 15 different values. Here use of one dimensional array is not advisable. Better option is to use two dimensional arrays. The table 15.1 can easily be represented by using two dimensional arrays of C. The two dimensional arrays are declared as follows:

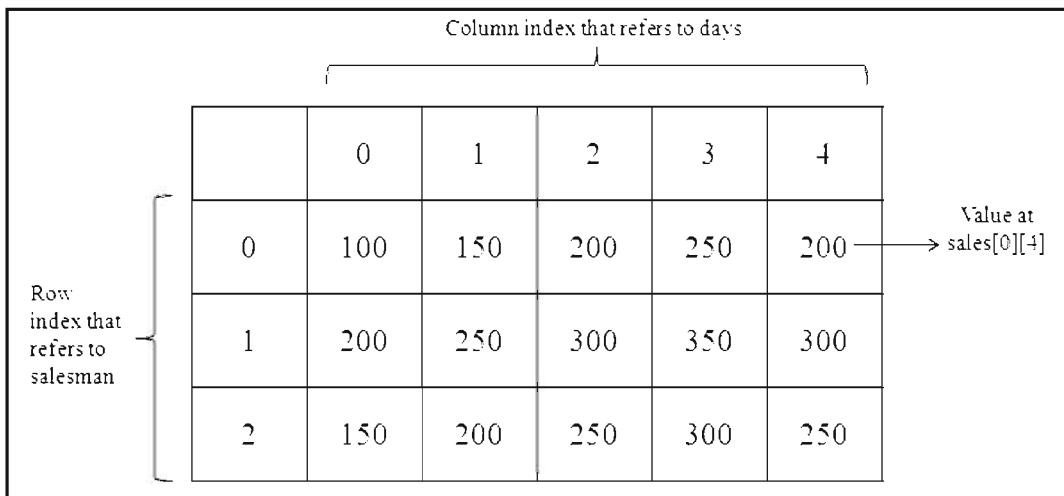
***datatype arrayname [row size ][column size];***

The *row size* and *column size* must be an integer constant greater than zero and datatype can be any valid C language data type. Note that in C language, the multidimensional arrays are row major. In other words, the first bracket in the declaration of an array specifies number of rows.

To store the data shown in table 15.1 we can declare a two dimensional array as under:

***int sales[3][5];***

Here row size=3 which is used to refer salesman (Salesman1 = 0, Salesman2 = 1, and Salesman3 = 2) while column size=5 which is used to refer days. (Monday = 0, Tuesday = 1, Wednesday = 2, Thursday = 3 and Friday = 4) We have assumed that unit sales values are in integer only. Figure 15.5 shows the sample memory layout of the Sales details.



**Figure 15.5 : Memory layout of Sales details**

Now to check the sales of item done by Salesman1 on Monday we may use array element sales[0][0]. The value stored at sales[0][0] is 100. Similarly to check the sales of item done by Salesman2 and Salesman3 on Monday, we may use array element sales[1][0] and sales[2][0] respectively. As shown in figure 15.5 the value stored at sales[0][4] is 200 and it refers to the sales of item done by Salesman1 on Friday.

## Initializing Two Dimensional Arrays

Like single dimensional array, we can also initialize two dimensional arrays at compile time and runtime of a program.

### Compile Time Initialization of Two Dimensional Arrays

The compile time array initialization of table 15.1 can be performed in a program as under:

```
int sales[3][5]={
 {100, 150, 200, 250, 200},
 {200, 250, 300, 350, 300},
 {150, 200, 250, 300, 250}
};
```

The nested inner curly braces, which indicate the intended salesman row, are optional. The following initialization is equivalent to previous example:

```
int sales[3][5]={100, 150, 200, 250, 200, 200, 250, 300, 350, 300, 150, 200, 250, 300, 250};
```

### Run Time Initialization of Two Dimensional Arrays

The run time array initialization of data shown in table 15.1 can be performed using program segment given in code listing 15.1.

```
int sales[3][5], row, column;
```

```
for(row = 0; row < 3; row++) {
 for (column = 0; column < 5; column++){
 printf("Enter sales item [%d][%d]:", row, column);
 scanf("%d", &sales[row][column]);
 }
}
```

#### Code Listing 15.1 : Program segment to add data at runtime

The program segment declares two dimensional array sales with three rows and five columns. The logic given in for loops will read total fifteen different values for sales array. Individual elements of sales array can be displayed on screen using the program segment given in code listing 15.2.

```
for(row = 0; row < 3; row++) {
 for (column = 0; column < 5; column++){
 printf("[%d][%d] := %d \n", row, column, sales[row][column]);
 }
}
```

#### Code Listing 15.2 : Program segment to show contents of array

## Example of Two Dimensional Arrays

Let us now understand a program to illustrate runtime array initialization and accessing values from one and two dimensional arrays. The program will declare two dimensional arrays for maintaining marks of students. The code of this program is given in code listing 15.3, along with its output.

```
/* Example 3: Program to illustrate use of one and two dimensional array. The program will
 maintain marks of two students in three quizzes using two dimensional arrays. The total marks
 of each student will be displayed.*/

#include<stdio.h>
#define STD 2 /*Number of rows for student data*/
#define QUIZ 3 /*Number of columns for quiz data*/

int main(){
 int marks[STD][QUIZ]; /* Marks array for 2 students and 3 quizzes */
 int i, j, total_quiz[STD] = {0};

 for(i = 0; i < STD; i++) /* Loop for student row */
 {
 for(j = 0; j < QUIZ; j++) /* Loop for quiz column */
 {
 printf("Enter marks of student %d in quiz %d: ", i+1, j+1);
 scanf("%d", &marks[i][j]); /* Reading marks of quiz */
 total_quiz[i] = total_quiz[i] + marks[i][j];
 }
 }

 /* Logic to print student no., quiz marks and total... */

 for (i = 0 ; i < QUIZ; i++) /* Loop for printing quiz number... */
 {
 printf("\tQuiz %d", i+1);
 }

 printf(" Total \n");
 for(i = 0; i < STD; i++) /* Loop for student row */
 {
 printf("Student %d:", i+1);
 for(j = 0; j < QUIZ; j++) /* Loop for quiz column */
 {
 printf("%d\t", marks[i][j]);
 }
 printf("%d \n", total_quiz[i]);
 }
 return 0;
}
/* End of Program*/
```

```

```

Output:

```
Enter marks of student 1 in quiz 1: 20
Enter marks of student 1 in quiz 2: 30
Enter marks of student 1 in quiz 3: 40
Enter marks of student 2 in quiz 1: 50
Enter marks of student 2 in quiz 2: 40
Enter marks of student 2 in quiz 3: 20
```

|            | Quiz1 | Quiz2 | Quiz3 | Total |
|------------|-------|-------|-------|-------|
| Student 1: | 20    | 30    | 40    | 90    |
| Student 2: | 50    | 40    | 20    | 110   |

```

```

### Code Listing 15.3 : Code and output of example 15.3

#### Explanation

Here we have used two symbolic constants STD and QUIZ in our program. STD refers to the number of students to be maintained in a program. QUIZ refers to the number of quizzes to be maintained in a program. We have declared two dimensional array marks to maintain quiz marks of students. One dimensional array total\_quiz is declared for storing total marks of all quizzes by a student. At the time of declaration of total\_quiz, all elements are initialized to zero. Two counter variables i and j are declared to repeat *for* loop body statements as per our requirement.

A nested for loop is used to read quizzes marks of students. Within the loop we had calculated total marks of all quizzes by a student. The remaining for loops are used to display output appropriately on the screen. The given program will be able to maintain two students' marks for three quizzes only. But we can make our program more dynamic by changing values of STD and QUIZ variable in our program.

#### No Array Bound Check

Arrays are one of the strength of C language but it is important to note that there is no array bound check in C language. Array bound check means checking the boundaries of array declared in a program. Consider the following array declaration statement:

```
int number[5];
```

This will declare an array variable number capable of storing five integer values. We can access individual array elements by using number[0], number[1].....number[4]; C provides power to the programmer to write any index value within the [ ] of an array. This means we can access number[-1] and also number[6] or any other illegal index location. As there is no bound check of array index, we will not get any error but the program will give us the garbage data or it will crash.

## Summary

In this chapter we had learnt important characteristics related to use of one dimensional and two dimensional arrays in C. We need to remember that the array datatype can be any valid C language data type. In array declaration the subscript number must be an integer or integer expression. An array index number starts with zero. Character arrays (string) are terminated by the special null character '\0'. There is no array boundary check in C language. It is programmer's responsibility to refer correct array index number in a program.

## EXERCISE

1. Give the advantages of using one dimensional array in a program.
2. State the difference between one and two dimensional arrays.
3. What do you mean by runtime and compile time array initialization of an array?
4. Find errors, if any in the following code, correct it and write the output.

(a) #include <stdio.h>

```
int main() {
 int num[3][3] = {{1, 2}, {3, 4}};
 printf("%d \n", number[1][1]);
 return 0;
}
```

(b) #include<stdio.h>

```
void main()
{
 char str[] = 'INDIA'
 printf("%c %c %c %c c", str[0], str[1], str[2], str[0], str[4]);
}
```

5. Fill in blanks :

- (a) The two-dimensional arrays have rows and \_\_\_\_\_.
- (b) An array element occupies \_\_\_\_\_ memory locations.
- (c) Index number contained within square brackets of an array is also known as \_\_\_\_\_.
- (d) A subscript number must be an integer or \_\_\_\_\_ expression.
- (e) Strings in a C language are terminated by the special character \_\_\_\_\_.

**6. State True or False :**

- (a) An array can be initialized only at runtime of a program.
  - (b) The C array index number starts with one.
  - (c) The simplest form of multidimensional array is one-dimensional array.
  - (d) C language provides the string as inbuilt data type.
  - (e) First bracket in the declaration of two dimensional arrays specifies number of rows.
  - (f) The statement: int temperature[6] = {35, 32, 42}; is valid array initialization.
  - (g) The statement: int num[ ] = { {1,1}, {2,2} }; is valid initialization.

**7.** Choose the correct option from the following :

- (1) What happens if a C program tries to access a value from an array element whose subscript exceeds the size of array ?

  - (a) The element will be set to 0.
  - (b) The compiler would report an error.
  - (c) The program may crash or gives garbage data.
  - (d) The array size increases automatically.

(2) What will be the output of following program segment?

```
int num[5] = {1, 2, 3, 4, 5};
```

```
int i, j;
```

```
i = num[1];
```

```
j = num[2];
```

```
printf("%d, %d, %d", i, j, num[0]);
```



- (3) What will be the output of following program segment ?

```
int num[5] = {1, 2, 3};
```

```
printf("%d, %d", num[0], num[3]);
```

## LABORATORY EXERCISE

Write a C program for performing followings tasks :

1. Read two five by five matrices from the user. Add these two matrices into third matrix and display the answer on screen.
2. Read marks of 20 students using an array in a program as shown below:

| Roll No | Marks |
|---------|-------|
| 1       | 60    |
| 2       | 70    |
| ...     | ....  |
| 20      | 75    |

- Display the Roll No. of students having highest marks amongst all.
- Display total number of students having marks greater than 50.
- Display total number of students having marks less than 50.
3. Read 10 integer numbers in one dimensional array. Display the array elements into reverse order. For example, if user input is 10, 20, 30, 40 then out output is 40, 30, 20, 10.
4. Find maximum, minimum and average of N elements of an array.
5. Read integer values for 3 X 3 Matrix from the user. Calculate and display sum of all elements of a Matrix.



## Function

One of the strength of C language is its ability to define and use functions. Function is a group of statements that performs a specific task. The function is also known as *method*, *sub-routine* or *procedure*. So far we have used number of C functions in our programs. We are familiar with the use of `main()`, `printf()` and `scanf()`. They are examples of C functions. Every executable C program has at least one `main()` function.

The use of function in a program makes it modular. Modularity means partitioning a complex problem into small sub-problems which are easy to understand and maintain. Once a problem is divided into small sub-problem, we can write a function for each sub-problem. The problem is then solved by using all functions together in a program. In this chapter we will discuss various types of C functions. We will also learn how function can be defined and used in a program.

There are two categories of functions in C.

- (1) Library functions or System defined functions
- (2) User defined functions

### Library Functions

The C language standard library provides many built-in functions that our program can use. They are also known as system defined functions. For example `printf()`, `scanf()`, `sqrt()` and `cos()` are examples of library functions. Library functions are not required to be written by normal user. These functions are available in compiled form in C library. Library functions are placed in various header files. For example, `sqrt()` and `cos()` are related to mathematics hence they are placed in `<math.h>` header file. To use any library function, we have to include its corresponding header file in our program. The list of some known header files and functions in that file are given in Appendix IV of this book.

Let us understand the use of one of the library function through example 16.1. The program uses inbuilt library function `pow()` that returns  $x$  raised to the power of  $y$ . Figure 16.1 gives the code listing and output of the example 16.1.

In the example 16.1, we had used `printf()` function which is defined in `<stdio.h>` header file. We had also used `pow()` function which returns the value of 8 raised to power of 2. The `pow()` function is defined in `<math.h>` file.

The screenshot shows the SciTE IDE interface. The title bar says "13-1.c - SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, Help. The code editor window contains the following C code:

```
13-1.c
/*
 * Example 1: Program to illustrate use of pow() library
 * function */
#include<stdio.h>
#include<math.h>
int main()
{
 float answer;

 answer = pow(8,2);
 printf("The value of 8 raised to 2 is %f\n",answer);

 return 0;
}
```

To the right of the code editor, a terminal window shows the command line and its output:

```
>gcc -pedantic -Os -std=c99 13-1.c -o 13-1
>Exit code: 0
>/13-1
The value of 8 raised to 2 is 64.000000
>Exit code: 0
```

Figure 16.1 : Code listing and output of Example 16.1

## User Defined Functions

The functions developed by user are known as user defined functions. The main( ) is special type of user defined function in C. The program execution starts from main( ) function. If we write our program using only main( ) function then it may become large. Testing and debugging of such program may become difficult for programmer. If program is divided into small subprograms (functions) then it is much easier to understand, test and debug it.

## Advantages of Functions

Followings are the advantages of using functions in our program :

- Dividing program into functions makes understanding and maintenance of the program easier.
- Functions written once can be called many times in our program. This reduces the actual length of program and thus saves memory.
- Functions help us to avoid redundant programming of similar tasks.
- A function written for one program may be used in other program.
- Same function can be used with different values of arguments.

Let us now try to understand how to use different functions in our program. Figure 16.2 shows control flow of a program that uses multiple functions.

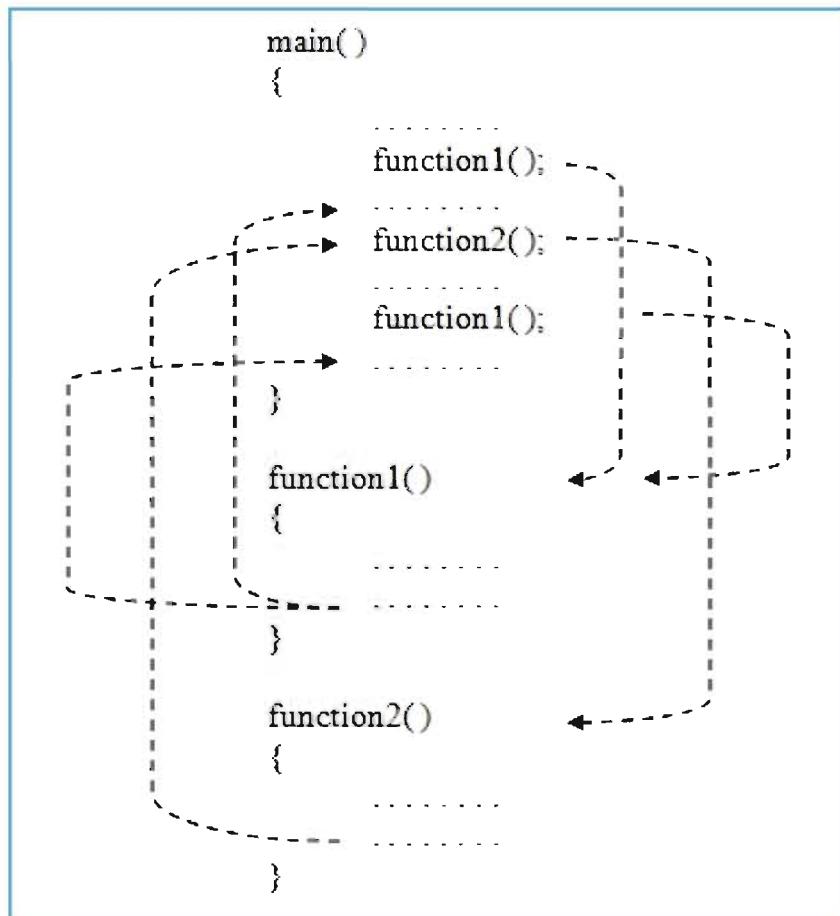


Figure 16.2 : Flow of control in a program with multiple functions

Figure 16.2 shows that in main( ), function1( ) is called twice and function2( ) is called once. During execution of statements in main( ) when control reaches to statement function1( ), then control is transferred to the body of function1( ). After executing all the statements inside body of function1( ) again control is returned to statement after function1( ) in the main ( ) function. In the same way again function2( ) and then function1( ) is called from the main program.

## Function Definition

If we want to use user defined function in our program then we have to inform this to compiler. We can inform the compiler by writing function definition. A function cannot be used without its definition. In C language, function can be defined using following syntax.

```
return_data_type function_name (arguments)
{
 function statements;
}
```

Here return\_data\_type specifies the type of information return by the function. If the return value from the function is integer value then return\_data\_type is int. If function does not return any value then the return\_data\_type is void.

The function\_name specifies the name of user defined function. The rules for giving name to functions are same as rules of naming variable. We should give meaningful function name. For example add( ) for performing addition, avg( ) for performing average, etc. The arguments show the input values provided to the function along with their data types. For more than one argument, all arguments are separated by commas in the function definition. Statements between opening and closing curly brace are known as function body. Based on requirements in a program, function body may contains local variable declarations and return statements. We will discuss about variable declaration in function and return statements later in this chapter.

Note that there is no need to define library function as they are already defined. All user defined functions needs to be defined as function prototype before being used in a program. Function prototype means declaration of a function before their use in a main ( ) function.

## Function Call

As stated earlier, every C program starts with function main( ). From main( ) functions we are calling other library or user defined functions. To call a function, use function name with required number of parameters. Let us look at the example 16.2 which illustrate how to call a function from the main( ). Figure 16.3 gives the code listing of the example 16.2, its output is shown in figure 16.4.

The screenshot shows the SciTE IDE interface with the title bar "13-2.c - SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, and Help. The code editor window contains the following C program:

```
/* Example 2: Program to illustrate use of function which returns no value.*/
#include<stdio.h>

void print_line(); // Function Prototype Declaration

void main()
{
 print_line(); // Function call
 printf("...Wel Come to the World of C Function...\n");
 print_line(); // Function call
}
// End of main() function

void print_line()
{
 printf("=====\\n");
}
// End of print_line() function
```

Figure 16.3 : Code listing of Example 16.2

```
gcc 13-2.c
>gcc 13-2.c
>Exit code: 0
./a.out
>./a.out
=====
...Wel Come to the World of C Function...
=====
>Exit code: 45
```

Figure 16.4 : Output of Example 16.2

### Explanation

In the program one user defined function `print_line()` is used. The `print_line()` statement within the `main` function calls this function. The control moves from `main()` to `print_line()` function where it prints the line pattern, then control moves back to the `main()`. The next `printf` statement in `main` prints the message. Next statement in `main()` once again calls `print_line()` function. In the example `print_line()` function is called twice. As this function does not return any value to the `main()` hence its return type is `void`.

### The Return Statement

Userdefined functions may or may not return any value to the calling function. To return any value to the calling function, we may use return statement inside function body. The format of return statement is as under:

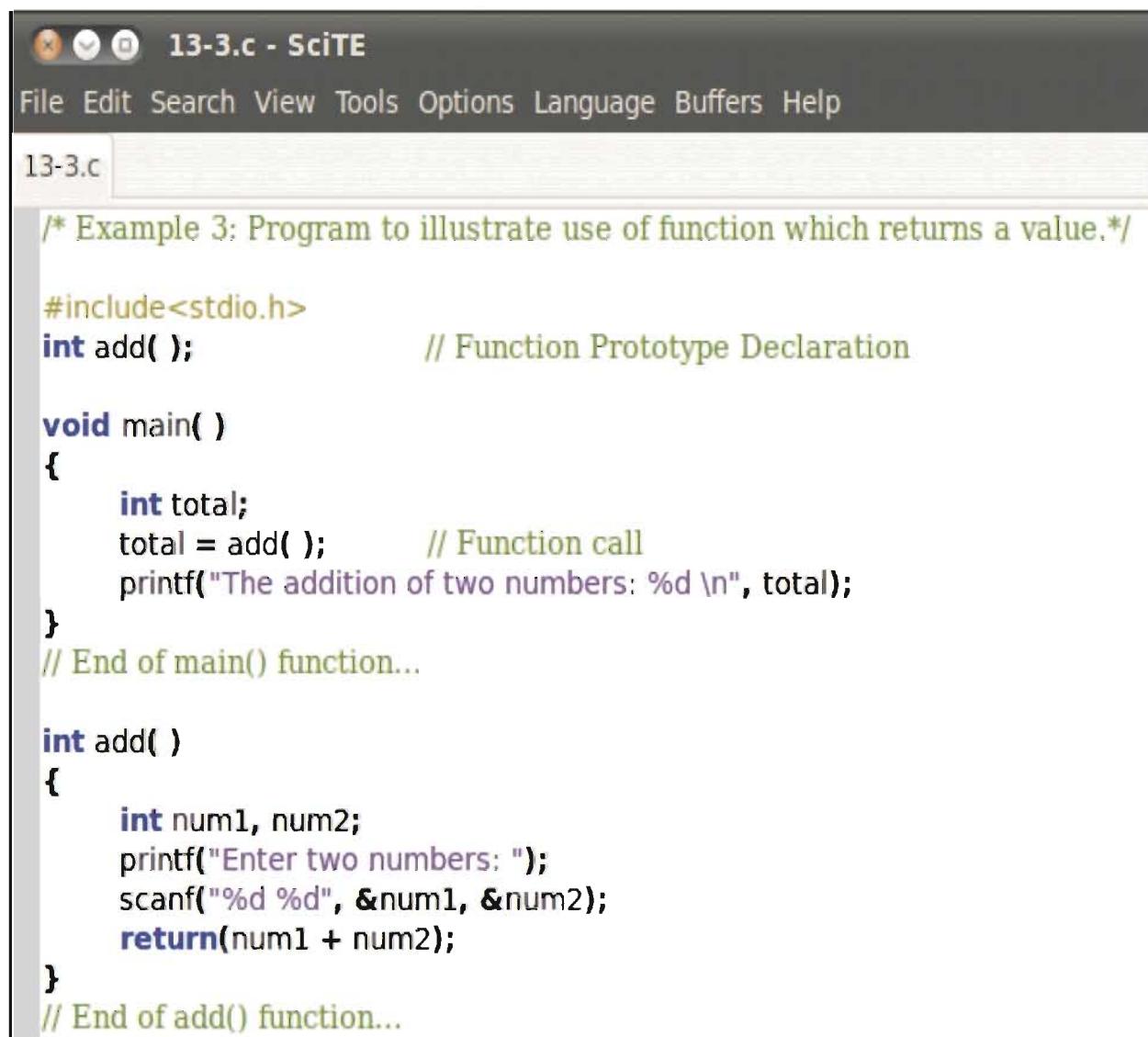
return;    or    return (expression);

The first form of return does not return any value. It returns only control back to the calling function. When return type of function is void then there is no need to use return statement at the end of function body.

Look at figure 16.5 that shows the code listing of example 16.3 to understand how the function returns a value to the calling function. The output is shown in figure 16.6.

### Explanation

In figure 16.5 we have used a user defined function add( ); In main( ) function we have declared a variable called total. When we call the add( ) function from the main(), the control is transferred to the add() function. Two local variables are declared namely num1 and num2 in function add. The next statements within add( ) displays message and read two numbers using printf and scanf statement respectively. Last statement returns addition of two numbers which is stored in the total variable of main( ) function. The total is then printed on screen.



The screenshot shows the SciTE IDE interface with the title bar "13-3.c - SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, and Help. The current file is "13-3.c". The code editor contains the following C program:

```
/* Example 3: Program to illustrate use of function which returns a value.*/

#include<stdio.h>
int add(); // Function Prototype Declaration

void main()
{
 int total;
 total = add(); // Function call
 printf("The addition of two numbers: %d \n", total);
}

// End of main() function...

int add()
{
 int num1, num2;
 printf("Enter two numbers: ");
 scanf("%d %d", &num1, &num2);
 return(num1 + num2);
}

// End of add() function...
```

Figure 16.5 : Code listing of Example 16.3

```

kpp@ubuntu: ~
File Edit View Terminal Help
kpp@ubuntu:~$ gcc 13-3.c
kpp@ubuntu:~$./a.out
Enter two numbers: 20 10
The addition of two numbers: 30
kpp@ubuntu:~$

```

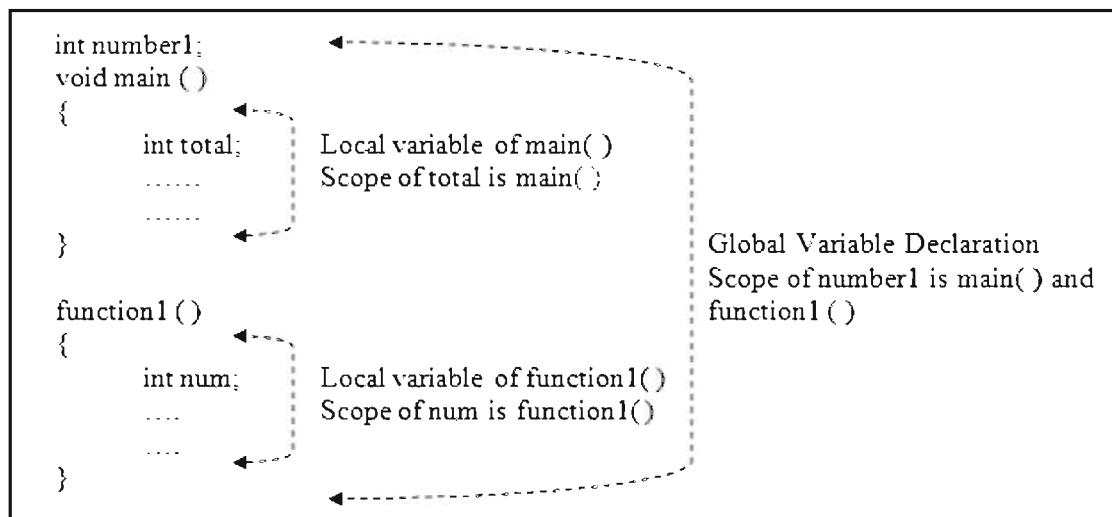
**Figure 16.6 : Output of Example 16.3**

### Scope of Variables

By scope of variable we mean the part of program where the variable is accessible. In C scope of variables are of two types:

- (i) Global variables
- (ii) Local variables

Figure 16.7 explains the scope of variable in a C program.



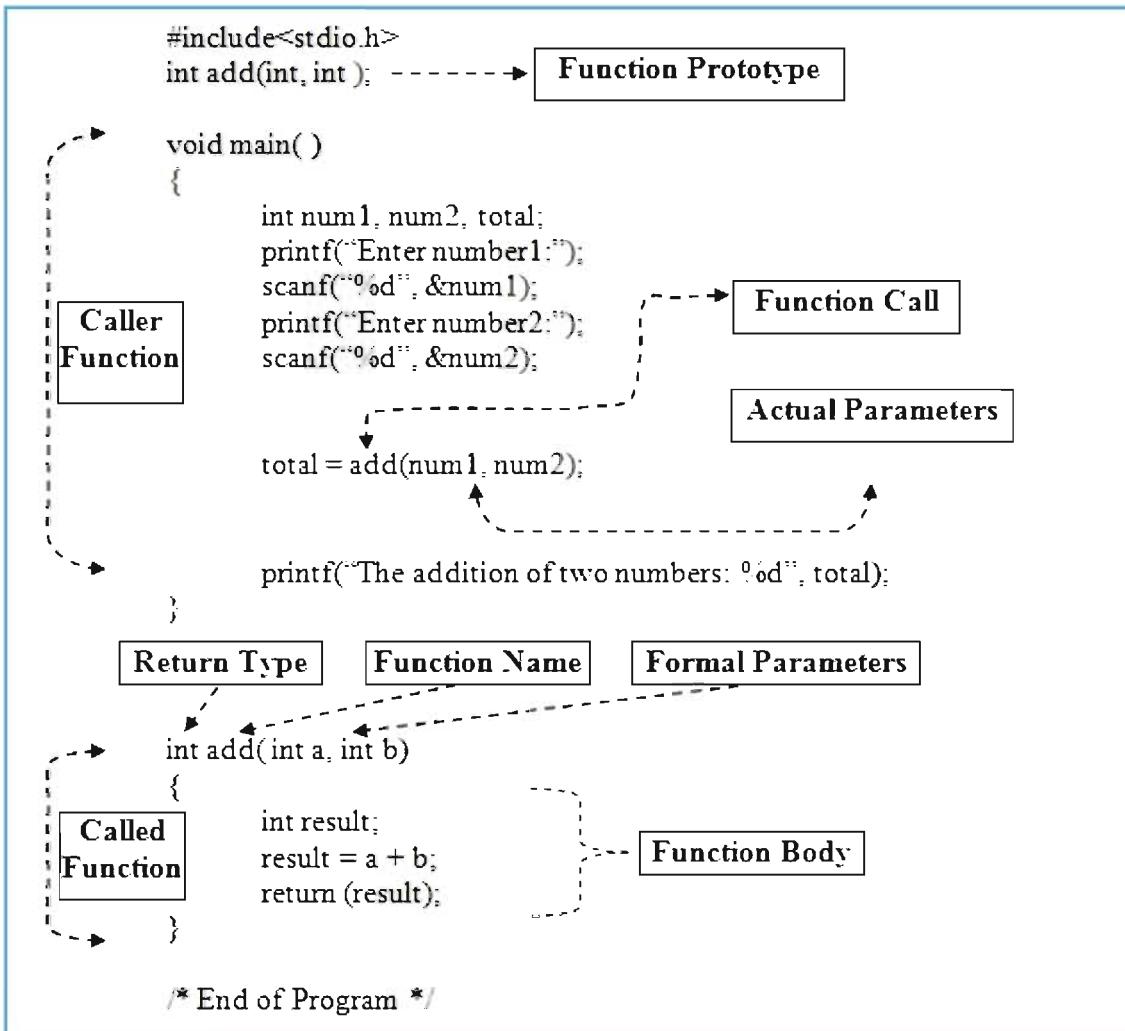
**Figure 16.7 : Scope of Variable**

Here the variable *number1* as it is defined above function *main* is known as global variable. The global variable *number1* is accessible in both *main()* and *function1()*. The local variable *int total* variable is accessible only within *main()* and not in *function1()*. Similarly the local variable *int num* is accessible only within *function1()* and not in *main()*.

Before getting in to the detail characteristics of C function, let us see the figure 16.8 which shows various components of the function.

As shown in figure 16.8 initially we have defined function prototype as *int add(int, int)*; this shows that there are two arguments of integer types. Function prototype is discussed later in the chapter. The *void main() { .... }* is the caller function which calls *add()* function with two actual parameters *num1* and *num2*. Here user defined function *int add()* indicates that it returns integer value to the caller function *void main()*. Two integer formal parameters "a" and "b" are defined in *add()* function.

These formal parameters accept the value sent at the time of calling function from the main( ). The integer variable result returns the addition of two numbers to the caller function which is stored in total variable of main( ).



**Figure 16.8 : Components of the C functions**

### Function Prototype

The function prototype is needed when the function is defined after the main( ) in a program. When we call any function in our program, compiler searches for a corresponding function definition. This is done to verify whether the function call is correct or not. If function call matches the header of the function in terms of number of arguments with their data types, it is valid otherwise compiler issues an error. For example

*int add(int, int); which is a prototype of function int add( int a, int b)*

Here prototype shows that add( ) is a function having two integer arguments and returns integer value. The function prototypes are written for each user defined function in beginning of the program.

### Function Arguments or Parameters

The basic idea behind using arguments or parameters in functions is to pass data between the calling and the called function. When the calling function sends the data to the called function it is called as argument or parameter passing.

The arguments used at the time of function definition are known as *formal* arguments. The call to such function needs same number of arguments to be passed to it at the time of function call. These arguments are known as *actual* arguments. The formal and actual arguments should match in number, type and order. If actual arguments are more than formal arguments, the additional actual arguments are simply discarded. When actual arguments are less than formal arguments, the unmatched formal arguments are initialized to garbage value. Also any mismatch in data types of arguments will result in garbage initialization.

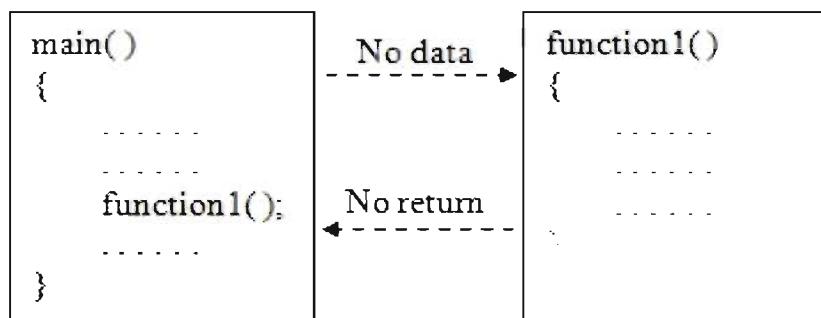
### Types/Categories of Functions

Depending on presence of arguments and return value in a function, they fall under following categories:

- (i) Function with no arguments and no return values
- (ii) Function with arguments and no return values
- (iii) Function with arguments and return values

### Function with No Arguments and No Return Values

When function has no arguments and no return value, it falls under this category. The called function does not receive any data value from calling function. Similarly due to no return value, the calling function does not receive any data from called function. In short there is no data transfer between calling and called functions. This is shown in figure 16.9. The dotted line in the figure indicates no data transfer but only transfer of control.

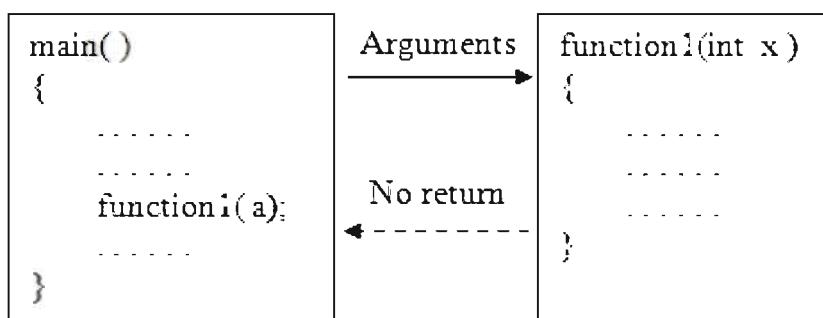


**Figure 16.9 : Functions with no data communications**

Example 16.2 explained earlier in this chapter is the example of category function with no arguments and no return values.

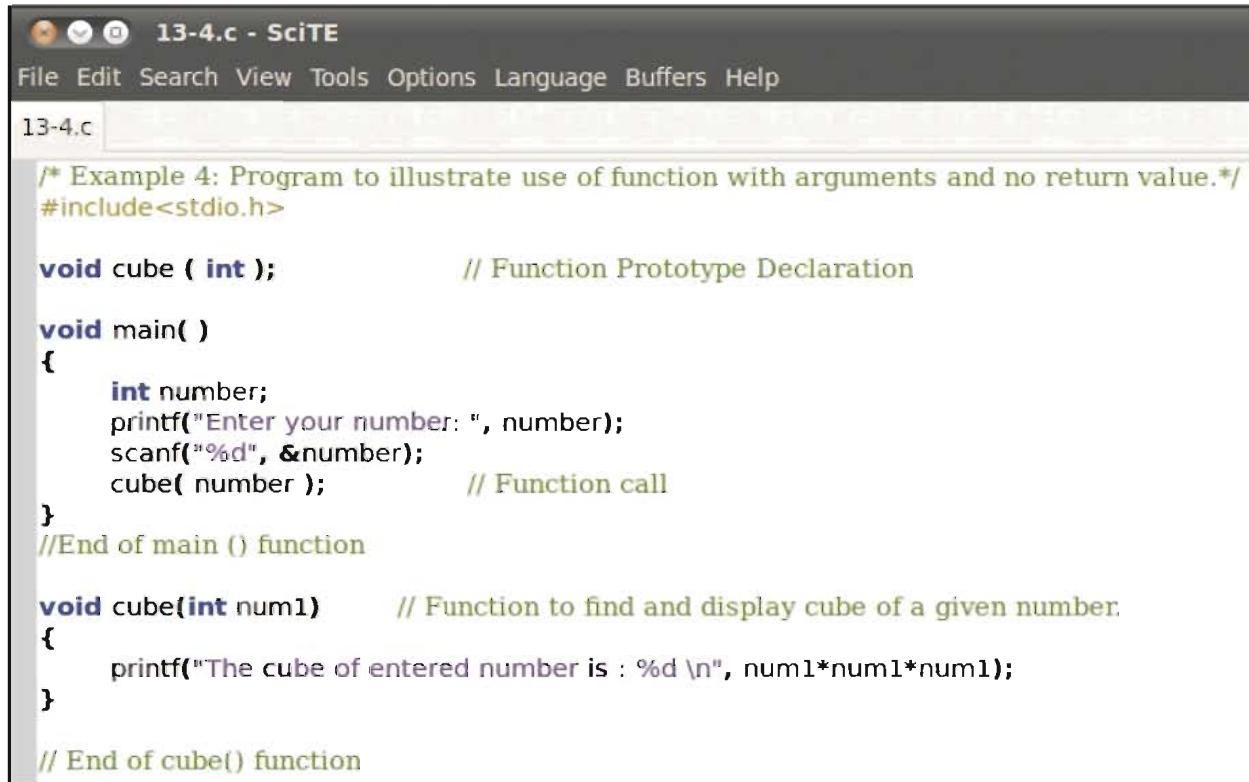
### Function with Arguments and No Return Values

The communication between the calling function and the called function with the arguments but no return value is shown in figure 16.10.



**Figure 16.10 : Function with one way data communication**

Let us try to understand the example 16.4 which will read a number from user and find its cube using function. We will send a number from calling function main( ) to called function cube( ) as an argument. The called function finds the cube and displays it on screen. Figure 16.11 gives the code listing of the example 16.4 while figure 16.12 shows its output.



```

13-4.c - SciTE
File Edit Search View Tools Options Language Buffers Help
13-4.c
/* Example 4: Program to illustrate use of function with arguments and no return value.*/
#include<stdio.h>

void cube(int); // Function Prototype Declaration

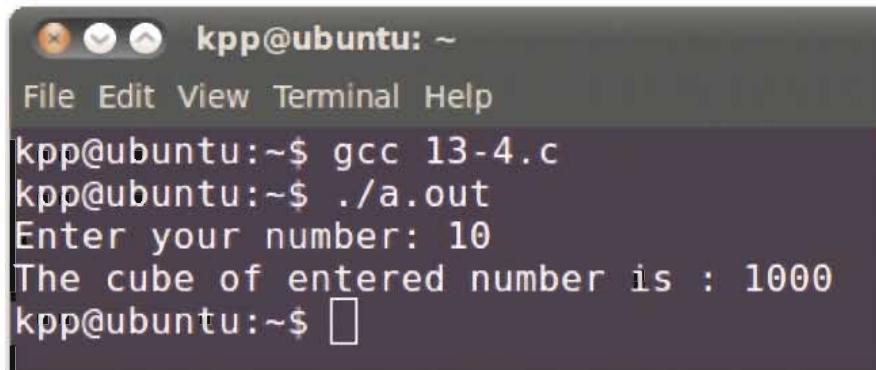
void main()
{
 int number;
 printf("Enter your number: ", number);
 scanf("%d", &number);
 cube(number); // Function call
}
//End of main () function

void cube(int num1) // Function to find and display cube of a given number.
{
 printf("The cube of entered number is : %d \n", num1*num1*num1);
}

// End of cube() function

```

**Figure 16.11 : Code listing of Example 16.4**



```

kpp@ubuntu: ~
File Edit View Terminal Help
kpp@ubuntu:~$ gcc 13-4.c
kpp@ubuntu:~$./a.out
Enter your number: 10
The cube of entered number is : 1000
kpp@ubuntu:~$

```

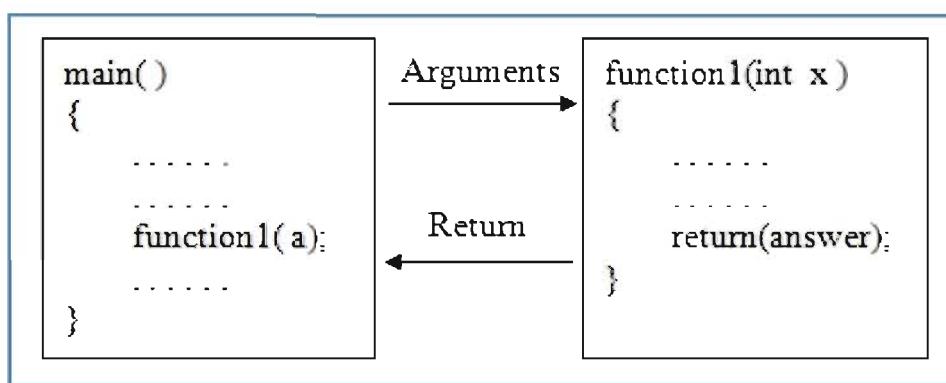
**Figure 16.12 : Output of Example 16.4**

### Explanation

Here main( ) function accepts a number from the user which is stored in to a number variable. This number variable is passed as an actual argument to the cube( ) function. In cube( ) function formal argument num1 gets the value of number variable. The cube( ) function then calculate and display the cube of given number using printf statement.

### Function with Arguments and Return Values

Let us understand the two way communication between the calling function and the called function. The function with arguments and return value is shown in figure 16.13.



**Figure 16.13 : Function with two way data communication**

Let us try to understand the example 16.5 which will read a number from user and find its square using `sqr()` function. We will send a number from calling function `main()` to called function `sqr()` as an argument. The called function finds the square of number and returns it to the calling function. Figure 16.14 gives the code listing of the example 16.5 while figure 16.15 shows its output.

```

13-5.c - SciTE
File Edit Search View Tools Options Language Buffers Help
13-5.c
/*
 * Example 5: Program to illustrate use of function with arguments and return value.*/
#include<stdio.h>

int sqr(int); // Function Prototype Declaration

void main()
{
 int number, result;
 printf("Enter your number: ", number);
 scanf("%d", &number);
 result = sqr(number); // Function call with argument
 printf("The square of %d is %d. \n", number, result);
}
//End of main() function

int sqr(int num1) // Function to find square of a number and return the answer
{
 int answer;
 answer = num1 * num1;
 return (answer);
}
//End of sqr() function

```

**Figure 16.14 : Code listing of Example 16.5**

```

kpp@ubuntu: ~
File Edit View Terminal Help
kpp@ubuntu:~$ gcc 13-5.c
kpp@ubuntu:~$./a.out
Enter your number: 10
The square of 10 is 100.
kpp@ubuntu:~$

```

**Figure 16.15 : Output of Example 16.5**

## Explanation

The main( ) function accepts a number from the user which is stored in to a number variable. Then number variable is passed as an actual argument to the sqr( ) function. In sqr( ) function formal argument num1 gets the value of number variable. The sqr( ) function then calculates the square and stores it into answer variable. The last statement in sqr( ) return the value stored in answer variable to the calling function. The caller function accepts the return value and stored it into result variable. Last statement in the main( ) displays the square of given number on screen.

## Summary

In this chapter we had discussed basic concepts related to the C functions. We had seen that use of function in a program makes it modular. We had learnt that how to use Library and User defined functions in our program. We had learnt that how to use global and local variables in our program. We had seen that how function can return a value to the calling function. The basic idea behind using arguments or parameters in the functions is discussed.

## EXERCISE

1. What do you mean by a function ? State advantages of using a function in a program.
2. State the advantages of using library function in a program.
3. What do you mean by scope of a variable ?
4. State how data/information can be passed to a function.
5. What is the meaning of function prototype ?
6. Differentiate between followings :
  - (a) Global and Local variables
  - (b) Formal and Actual parameters
  - (c) User defined function and Library function
7. Find errors (if any) in the following code; correct it and write the output :
  - (a)

```
#include<stdio.h>
void draw_line();
void main()
{
 draw_line;
 printf("...Wel Come to the C Function...\n");
 draw_line;
}
void print_line()
{
 printf("= = = = = = = = = = = = = = = = = =\n");
}
```

(b)

```
#include<stdio.h>

int sqr (int);
void main()
{
 int number = 5, result;
 result = sqr();
 printf("The square of 5 is %d.", result);
}

int sqr(int num1)
{
 int answer;
 answer = num1 * num1;
 print (answer);
}
```

8. Fill in blanks :

- (a) The \_\_\_\_\_ statement returns the information from a function.
- (b) If function does not return any value then its return data type is \_\_\_\_\_.
- (c) The library function sqrt( ) is defined in \_\_\_\_\_ header file.
- (d) The variable declared above main( ) function are called \_\_\_\_\_ variables.

9. State True or False :

- (a) C program functions can pass only one value.
- (b) Function cannot send multiple arguments together.
- (c) Same user defined function cannot be called multiple times from main( ).
- (d) The main( ) is also one type of function in C program.
- (e) The function prototype declarations for library functions are not required in the beginning of a program.

10. Choose the correct option from the following :

- (1) The main() function in C language is a
- (a) User defined function
  - (b) Library function
  - (c) Keyword
  - (d) Reserved function

- (2) By default the main() function returns
- (a) Char value
  - (b) Float value
  - (c) Integer value
  - d. Double value
- (3) Which keyword is used as a return type when function does not return any value?
- (a) int
  - (b) main
  - (c) void
  - (d) auto
- (4) Which of the following is not an inbuilt library function ?
- (a) pow()
  - (b) printf()
  - (c) sum()
  - (d) sqrt()
- (5) What will be the value of variable number2 after execution of following code :
- ```
int main(){
    int number1 = 2, number2;
    number2 = sqr(number1);
    printf("number2 = %d",number2);
    return 0;
}
int sqr(int n){
    return(n*n);
}//End of program
```
- (a) number2 = 4
 - (b) number2 = 6
 - (c) number2 = 8
 - (d) number2 = 18

LABORATORY EXERCISE

Write a C program to perform following tasks :

1. Read an integer number N in main() function. Pass the value of N to the function as an argument. Display the sum of following series using a function for a given N number.

$$1 + 2 + 3 + 4 + \dots + N$$
2. Write a user defined function which will display Welcome message 10 times.
3. Define a function which will calculate the x to the power n.
4. Write a user defined function to find maximum from the given two numbers.
5. Read a number from main(), pass it to the user defined function as an argument and display whether given number is positive or negative.
6. Read a number in your program and apply library function abs(), sin(), cos() and log() on it and display the result.
7. Accept a character arrays (string) from user and display the string in reverse order.



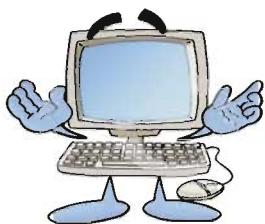
Appendix I



List of decimal numbers with corresponding octal and hexadecimal numbers

Decimal Number (0,1,2,...,9)	Binary Number (0,1)	Octal Number (0,1,2,...,7)	Hexadecimal Number (0,1,2,...,9,A,B,C,D,E,F)
1	00001	1	1
2	00010	2	2
3	00011	3	3
4	00100	4	4
5	00101	5	5
6	00110	6	6
7	00111	7	7
8	01000	10	8
9	01001	11	9
10	01010	12	A
11	01011	13	B
12	01100	14	C
13	01101	15	D
14	01110	16	E
15	01111	17	F
16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13
20	10100	24	14

Appendix II



ASCII Values of Characters

ASCII		ASCII		ASCII		ASCII	
Value	Character	Value	Character	Value	Character	Value	Character
000	NUL	032	blank	064	@	096	
001	SOH	033	!	065	A	097	a
002	STX	034	"	066	B	098	b
003	ETX	035	#	067	C	099	c
004	EOT	036	\$	068	D	100	d
005	ENQ	037	%	069	E	101	e
006	ACK	038	&	070	F	102	f
007	BEL	039	'	071	G	103	g
008	BS	040	(072	H	104	h
009	HT	041)	073	I	105	i
010	LF	042	*	074	J	106	j
011	VT	043	+	075	K	107	k
012	FF	044	,	076	L	108	l
013	CR	045	-	077	M	109	m
014	SO	046	.	078	N	110	n
015	SI	047	/	079	O	111	o
016	DLE	048	0	080	P	112	p
017	DC1	049	1	081	Q	113	q
018	DC2	050	2	082	R	114	r
019	DC3	051	3	083	S	115	s
020	DC4	052	4	084	T	116	t
021	NAK	053	5	085	U	117	u
022	SYN	054	6	086	V	118	v
023	ETB	055	7	087	W	119	w
024	CAN	056	8	088	X	120	x
025	EM	057	9	089	Y	121	y
026	SUB	058	:	090	Z	122	z
027	ESC	059	;	091	[123	{
028	FS	060	<	092	\	124	
029	GF	061	=	093]	125	}
030	RS	062	>	094	?	126	~
031	US	063	?	095	-	127	DEL

The characters 33 to 126 are printable. The other characters are control characters; they cannot be printed.




Appendix III

Summary of C Operators

Operator	Operation Used for	Associativity	Precedence
()	Function call	Left to Right	First
[]	Array expression		
→	Structure operator		
.	Structure operator		
+	Unary plus	Right to Left	Second
-	Unary minus		
++	Increment		
--	Decrement		
!	Logical NOT		
~	Bitwise NOT		
*	Pointer operator		
&	Address operator		
sizeof()	Size of operand	Left to Right	Third
*	Multiplication		
/	Division		
%	Modulo division		
+	Binary addition	Left to Right	Fourth
-	Binary subtraction		
<<	Left shift		
>>	Right shift		
<	Less than	Left to Right	Sixth
<=	Less than equal to		
>	Greater than		
>=	Greater than equal to		
==	Equal to	Left to Right	Seventh
!=	Not equals to		
&	Bitwise AND		Eighth
^	Bitwise XOR		Ninth
	Bitwise OR	Left to Right	Tenth
&&	Logical AND		Eleventh
	Logical OR		Twelfth
?:	Conditional operator		Thirteenth
=, *=, -=, &=, +=,		Right to Left	
^=, =, <<=, >>=	Assignment Operator		Fourteenth
,	Comma operator		Fifteenth



Appendix IV

Some of the Regularly Used Header Files

As we already know that in C Language functions are very important. C has a collection of inbuilt functions to perform various operations. These functions are grouped and stored in header files. Collection of such header files are called C library. Following is the list of some of the header files that programmers might like to use.

File name	Purpose
<stdio.h>	Standard I/O header file
<ctype.h>	Character testing and conversion functions
<math.h>	Mathematical functions
<stdlib.h>	Different types of functions used for string conversions, memory allocations, random number generation.
<string.h>	String manipulation functions

The some of the functions available in the above header files are shown below. At times function need to be provided an input. The input is part of function definition and is known as arguments. The arguments may have different data types. To identify the arguments we have used the character codes given below :

c	- argument of type char
d	- argument of type double
f	- argument of type file
i	- argument of type int
l	- argument of type long
p or *	- argument of type pointer
s	- argument of type string
u	- argument of type unsigned integer

The functions at times are also capable of returning a value of different data types. The list below gives idea about different function, its argument type and the type of value it returns.

Functions in <stdio.h> file		
Function	Return Value	Description
getc(f)	int	Enter a single character from file f.
getchar(void)	int	Enter a single character from the standard input device.
gets(s)	char*	Enter string s from the standard input device.
printf(...)	int	Send data items to the standard output devices.
putc(c,f)	int	Send a single character to file f.
putchar(c)	int	Send a single character to the standard output device.
puts(s)	int	Send string s to the standard output device.
scanf(...)	int	Enter data items from the standard input device.

Functions in <ctype.h> file		
Function	Return Value	Description
isalnum(c)	int	Determine if argument is alphanumeric. Returns nonzero value if true; 0 otherwise.
isalpha(c)	int	Determine if argument is alphabet. Returns nonzero value if true; 0 otherwise.
isascii(c)	int	Determine if argument is an ASCII character. Returns non zero value if true; 0 otherwise.
iscntrl(c)	int	Determine if argument is an ASCII control character. Returns nonzero value if true; 0 otherwise.
isdigit(c)	int	Determine if argument is a decimal digit. Returns nonzero value if true; 0 otherwise.
islower(c)	int	Determine if argument is lowercase. Returns nonzero value if true; 0 otherwise.
isodigit(c)	int	Determine if argument is an octal digit. Returns nonzero value if true; 0 otherwise.
isprint(c)	int	Determine if argument is a printable ASCII character. Returns nonzero value if true; 0 otherwise.
ispunct(c)	int	Determine if argument is a punctuation character. Returns nonzero value if true; 0 otherwise.
isspace(c)	int	Determine if argument is whitespace character. Returns non-zero value if true; 0 otherwise.
isupper(c)	int	Determine if argument is uppercase. Returns nonzero value if true; 0 otherwise.
isxdigit(c)	int	Determine if argument is a hexadecimal digit. Returns nonzero value if true; 0 otherwise.
toascii(c)	int	Convert value of argument to ASCII.
tolower(c)	int	Convert letter to lowercase.
toupper(c)	int	Convert letter to uppercase.

Functions in <math.h> file		
Function	Return Value	Description
acos(d)	double	Returns the arc cosine of d.
asin(d)	double	Returns the arc sine of d.
atan(d)	double	Returns the arc tangent of d.
atan2(d1, d2)	double	Returns the arc tangent of d1/d2.
ceil(d)	double	Returns a value rounded up to the next higher integer.
cos(d)	double	Returns the cosine of d.
cosh(d)	double	Returns the hyperbolic cosine of d.
exp(d)	double	Raises e to the power d.
fabs(d)	double	Returns the absolute value of d.
floor(d)	double	Returns a value rounded down to the next lower integer.
fmod(d1,d2)	double	Returns the remainder of d1/d2 (with same sign as d1).
labs(1)	long int	Returns the absolute value of 1.
log(d)	double	Returns the natural logarithm of d.
log10(d)	double	Returns the logarithm (base 10) of d.
pow(d1,d2)	double	Return d1 raised to the d2 power.
sin(d)	double	Returns the sine of d.
sinh(d)	double	Returns the hyperbolic sine of d.
sqrt(d)	double	Returns the square root of d.
tan(d)	double	Returns the tangent of d.
tanh(d)	double	Returns the hyperbolic tangent of d.

Functions in <stdlib.h> file

Function	Return Value	Description
abs(i)	int	Returns the absolute value of i.
atof(s)	double	Converts string s to a double-precision quantity.
atoi(s)	int	Converts string s to an integer.
atol(s)	long	Converts string s to a long integer.
calloc(u1,u2)	void*	Allocates block of memory having u1 elements each of length u2 bytes. Returns a pointer to the beginning of the allocated space.
exit(u)	void	Closes all files and buffers and terminate the program. (value of u is assigned by the function to indicate termination status)
free(p)	void	Frees a block of allocated memory whose beginning is indicated by p.
malloc(u)	void*	Allocates u bytes of memory. Returns a pointer to the beginning of the allocated space.
rand(void)	int	Returns a random positive integer.
realloc(p,u)	void*	Allocates u bytes of new memory to the pointer variable p. Returns a pointer to the beginning of the new memory space.

Functions in <string.h> file

Function	Return Value	Description
strcmp(s1, s2)	int	Compares two strings lexicographically. Returns a negative value if s1 < s2; 0 if s1 and s2 are identical; and a positive value if s1 > s2.
strcmpl(s1,s2)	int	Compares two strings lexicographically irrespective of case. Returns a negative value if s1 < s2; 0 if s1 and s2 are identical; and a value if s1 > s2
strcpy(s1, s2)	char*	Copy string s2 to s1.
strlen(s)	int	Returns the number of character in string s.

