Getting Started with Ch

Harry H. Cheng

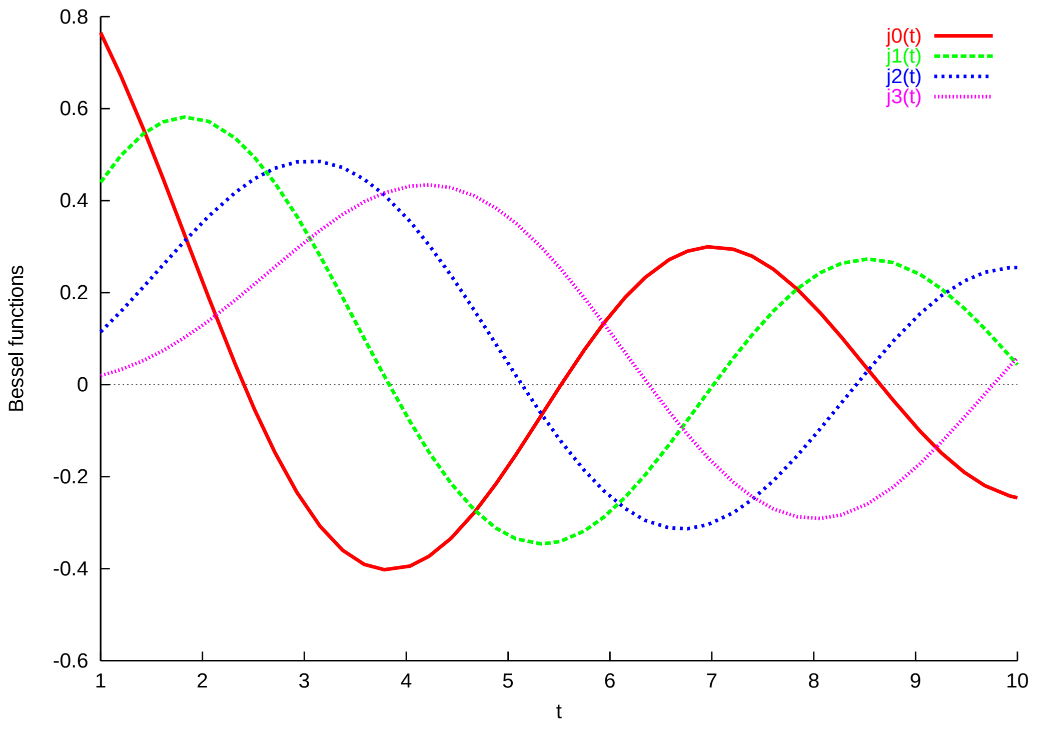


Table of Contents

Getting Started with Ch

Ch is a C/C++ interpreter. It is a superset of C with classes in C++ and other high-level extensions. Possible uses for Ch include but are not limited to cross-platform scripting, shell programming, 2D/3D plotting, numerical computing, and embedded scripting. Because Ch is interpretive and does not require tedious edit/compile/link/debug cycles, it is more suitable for interactive classroom presentations in teaching and for students learning C and C++. The latest version of Ch for different platforms can be downloaded from the internet at `http://www.softingeration.com`. This document presents a quick introduction on how to use this C/C++ interperter for learning C and C++.

1 Getting Started with an IDE

An Integrated Development Environment (IDE) can be used to develop C and C++ programs. It can typically be used to edit programs with added features of automatic syntax highlighting and run the programs within the IDE. ChSciTE is an open source Ch IDE for developing and running C/Ch/C++ programs in Windows, Mac OS X, and Unix. With Ch installed on the machine, the edited program can readily be executed in ChSciTE like in any other IDE. The source files are executed interpretively without compilation.

The latest version of ChSciTE for different platforms can be downloaded from the internet at `http://chscite.sourceforge.net`. ChSciTE can be launched by running the command `chscite`. In Windows, ChScitTE can also be conveniently launched by double clicking its icon shown in Figure 1 on the desktop.

Text editing in ChSciTE works similarly to most Macintosh or Windows editors such as Notepad with the additional feature of automatic syntax styling. The user interface can be in one of 30 local languages such as German, French, Chinese, and Korea. ChSciTE can hold multiple files in memory at one time but only one file will be visible. By default, ChSciTE allows up to 20 files to be in memory at once.

As an example, open a new document, and type

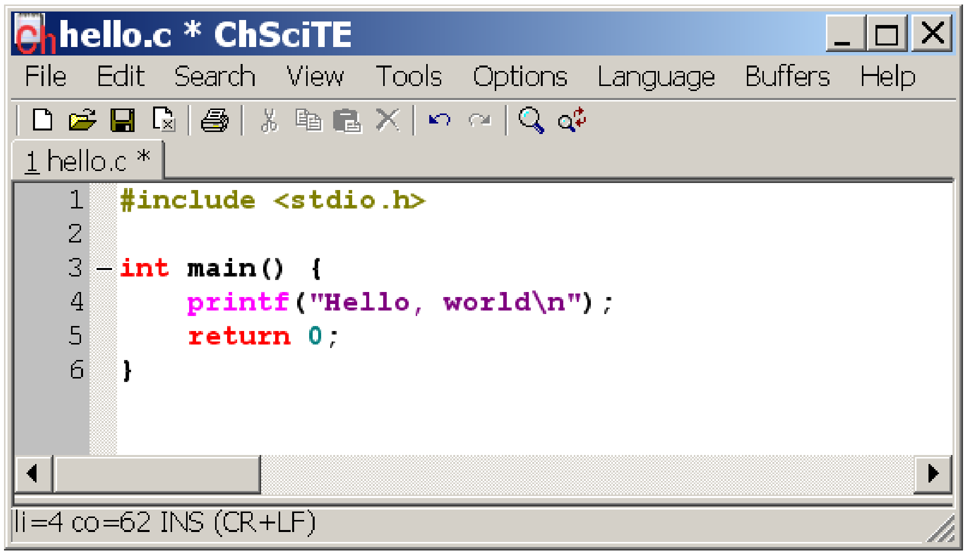
```
#include <stdio.h>

int main() {
    printf("Hello, world!\n");
    return 0;
}
```

in the text as shown in Figure 2 in the editing pane. The program appears colored due to syntax highlighting.

Save the document as a file named `hello.c` as shown in Figure 3. The program `hello.c` in `CHHOME/demos/bin/hello.c`, where `CHHOME` is the home directory for Ch such as `C:/Ch` in Windows for the directory `Ch` in the `C:` drive, can also be loaded using `File | Open` command.

Figure 1. A ChSciTE icon on a desktop in Windows.

.



```c
#include <stdio.h>

int main() {
    printf("Hello, world\n");
    return 0;
}
```

li=4 co=62 INS (CR+LF)

Figure 2. The program edited inside the editing pane in ChSciTE.

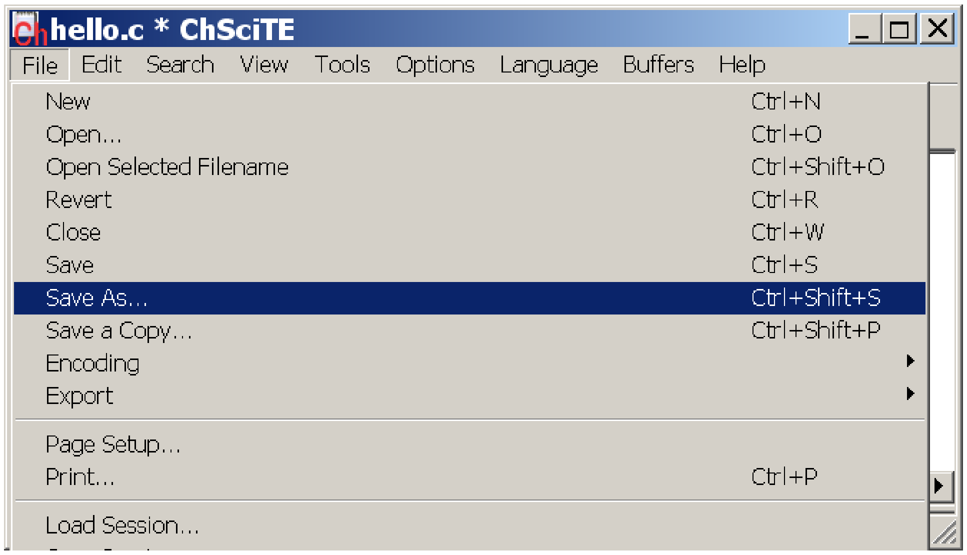
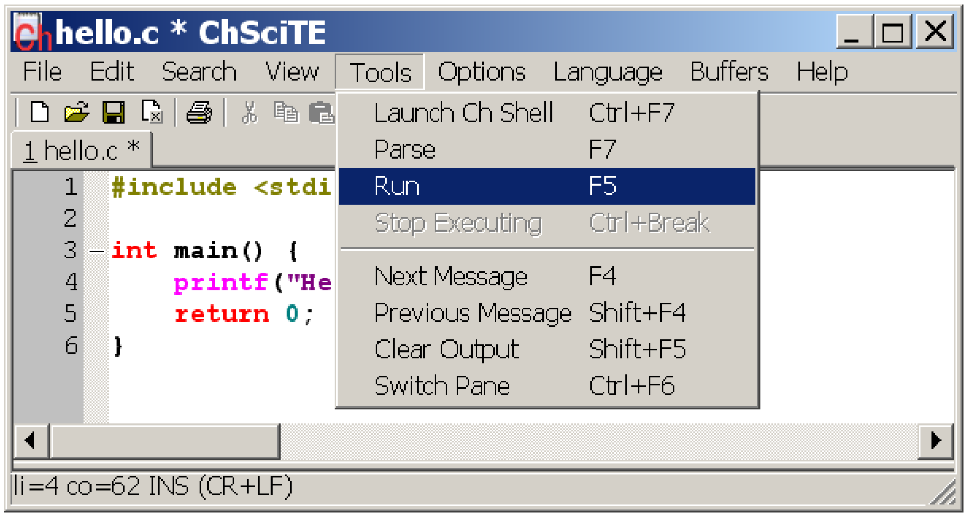


Figure 3. Save the edited program in ChSciTE.



```
#include <stdi

int main() {
    printf("He
    return 0;
}
```

Figure 4. Run the program inside the editing pane in ChSciTE.

.

Perform the `Tools | Run` as shown in Figure 4 to execute the program `hello.c`. Instead of performing the `Tools | Run` command, pressing function key `F5` will have the same effect of executing the program.

There are two panes in ChSciTE, the editing pane and the output pane. The output pane is located either below the editing pane or on the right. Initially, it is of zero size, but it can be made larger by dragging the divider between it and the editing pane. By default, the output from the program are directed into the output pane below the editing pane. The `Options | Vertical Split` command can be used to move the output pane to the right of the editing pane.

When the program `hello.c` is executed, the output window will be made visible if it is not already visible and will display

```
ch -u hello.c
Hello, world
Exit code: 0
```

as shown in Figure 5. The first blue line from ChSciTE shows the command it will use to run the program. The black line is the output from running the Ch program. The last blue line is from ChSciTE showing that the program has finished and displays its exit code. An exit code of 0 indicates that the program is terminated successfully by the statement

```
return 0;
```

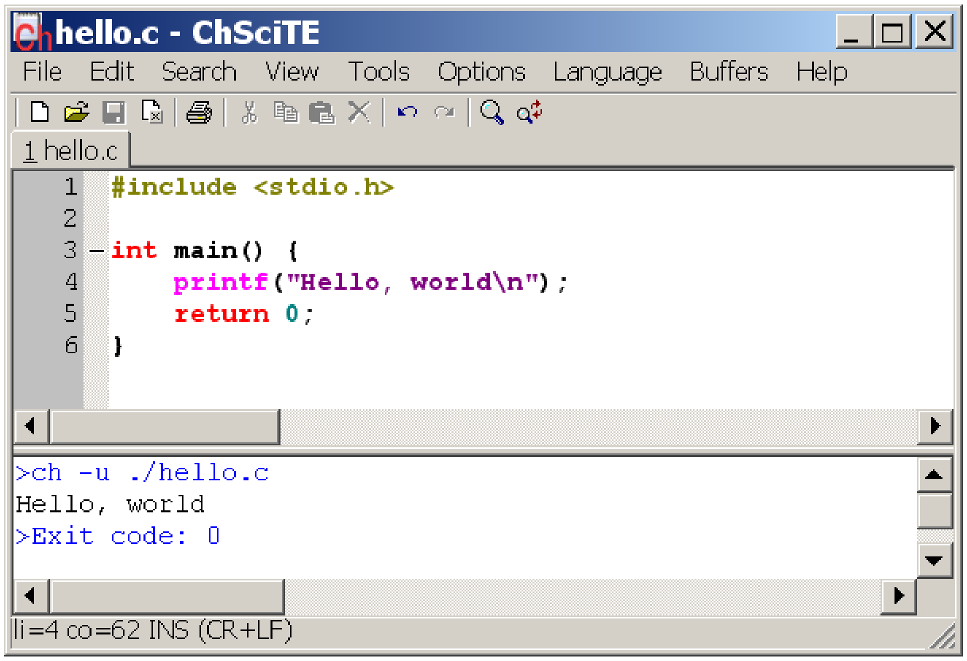
If a failure had occurred during the execution of the program or the returned value was non-zero, the exit code would be -1.

ChSciTE understands the error messages produced by Ch. To see this, add a mistake to the program by changing the line

```
printf("Hello, world");
```

to

```
printf("Hello, world";
```



```
#include <stdio.h>

int main() {
    printf("Hello, world\n");
    return 0;
}
```

```
>ch -u ./hello.c
Hello, world
>Exit code: 0
```

Figure 5. The output from executing program hello.c.

.

Perform the `Tools | Run` the modified program. The results should look below

```
ERROR: syntax error before or at line 4 in file hello.c
  ==>:    printf("Hello, world\n";
  BUG:    printf("Hello, world\n"; <== ???
ERROR: cannot execute command 'hello.c'
```

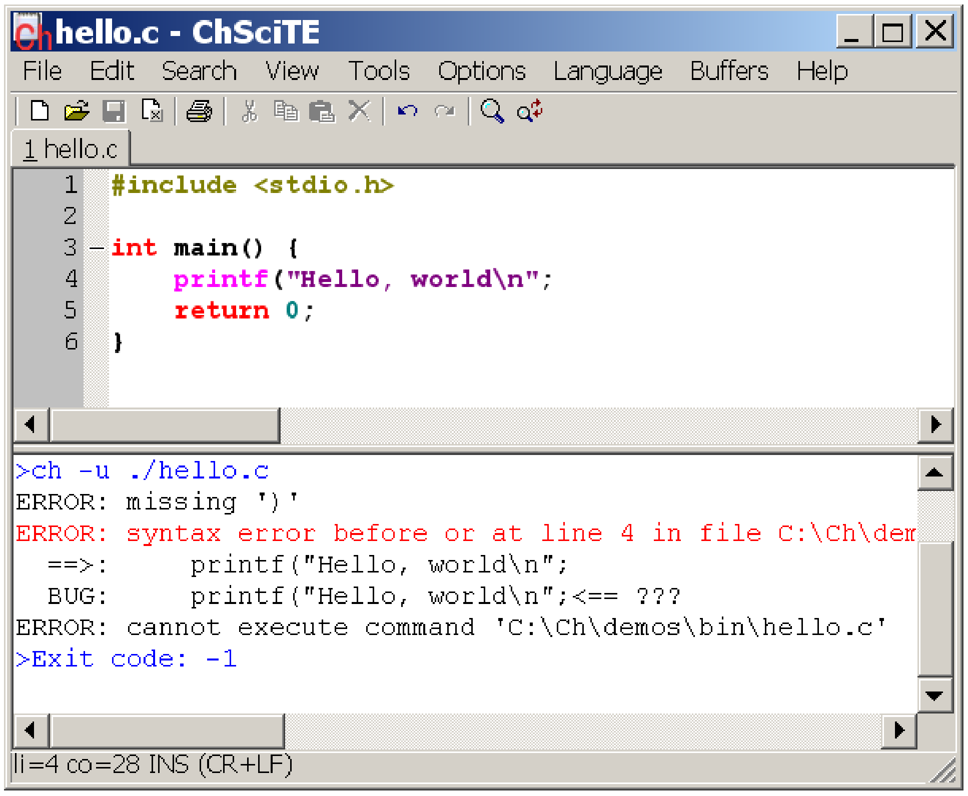
as shown in Figure 6. Because the program fails to execute, the exit code -1 is displayed at the end of the output pane as

```
Exit code: -1
```

Click the red colored output line in Figure 6. the line with incorrect syntax will be highlighted as shown in Figure 7. When the error message in the output pane with a line number is clicked using the left button of the mouse, the error message in the output pane and the appropriate line in the editing pane are highlighted with a yellow background as shown in Figure 7. The caret is moved to this line and the pane is scrolled if needed to show the line. While it is easy to see where the problem is in this simple case, with a large file, the `Tools | Next Message` command can be used to view each of the reported errors. Upon performing `Tools | Next Message`, the first error message in the output pane and the appropriate line in the editing pane are also highlighted with a yellow background.

If command execution has failed and is taking too long to complete, then the `Tools | Stop Executing` command can be used.

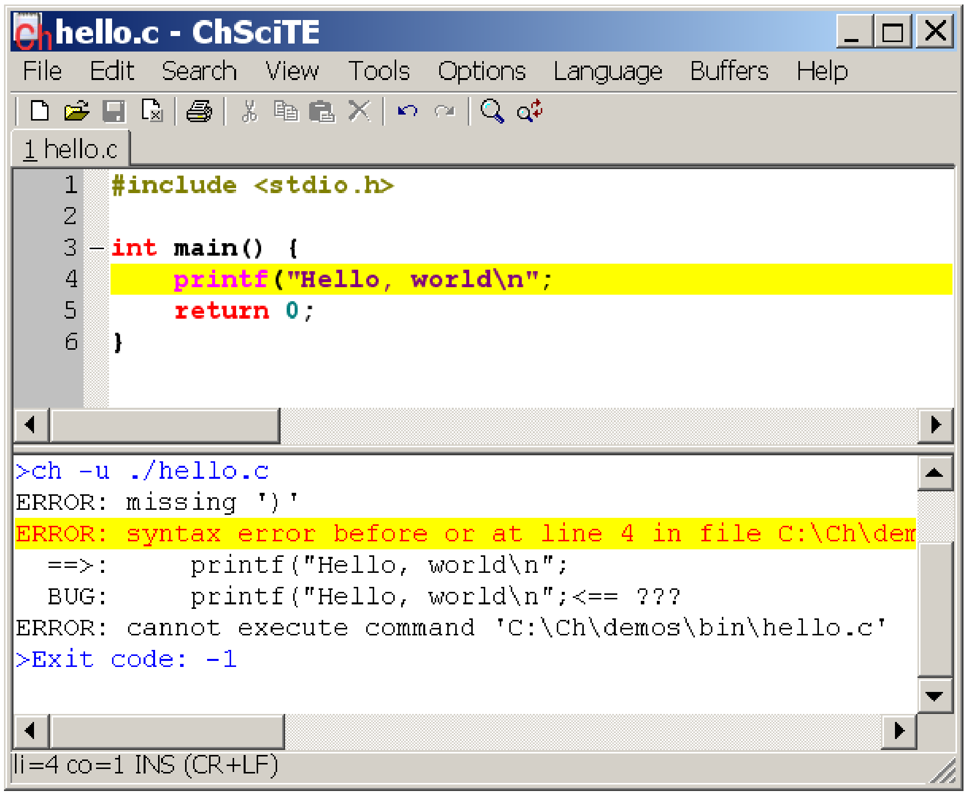
ChSciTE can also execute programs with the user's input through C functions such as `scanf()`. It can also handle command line options. More information about running C and C++ programs in Ch using ChSciTE IDE can be obtained on-line by clicking ChSciTE Help as shown in Figure 8.



```
1  #include <stdio.h>
2
3  int main() {
4      printf("Hello, world\n";
5      return 0;
6  }
```

```
>ch -u ./hello.c
ERROR: missing ')'
ERROR: syntax error before or at line 4 in file C:\Ch\dem
  ==>:     printf("Hello, world\n";
  BUG:     printf("Hello, world\n";<== ???
ERROR: cannot execute command 'C:\Ch\demos\bin\hello.c'
>Exit code: -1
```

Figure 6. The error line in output from executing program hello.c.



```
1  #include <stdio.h>
2
3  int main() {
4      printf("Hello, world\n";
5      return 0;
6  }
```

```
>ch -u ./hello.c
ERROR: missing ')'
ERROR: syntax error before or at line 4 in file C:\Ch\dem
  ==>:     printf("Hello, world\n";
  BUG:     printf("Hello, world\n";<== ???
ERROR: cannot execute command 'C:\Ch\demos\bin\hello.c'
>Exit code: -1
```

Figure 7. Finding the error line in output from executing program hello.c.

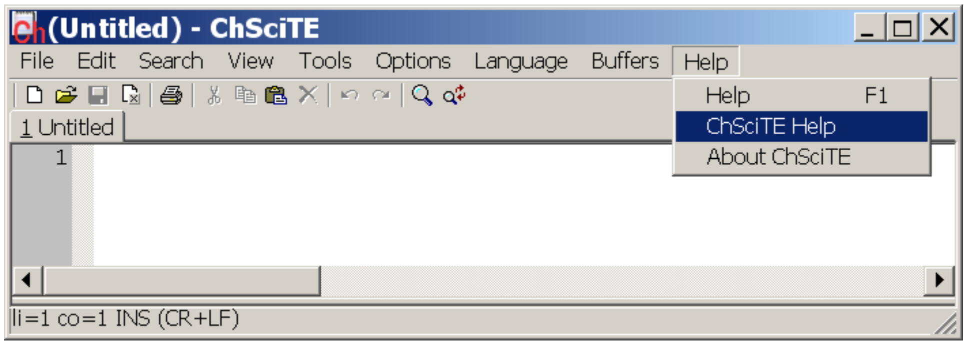


Figure 8. Get on-line help on how to use ChSciTE.

Figure 9. A Ch icon on a desktop in Windows.

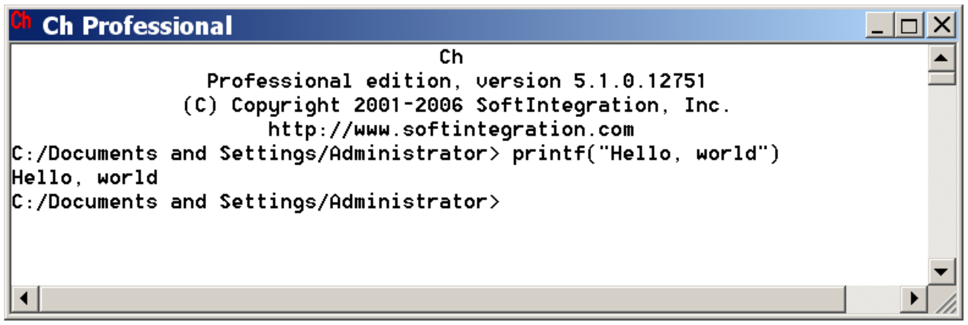
2 Getting Started with Ch Command Shell

Ch is also a command shell in which commands are processed. Like other commonly used shells such as the MS-DOS shell, Bash-shell, or C-shell, commands can be executed in a Ch shell. Unlike these conventional shell, expressions, statements, functions and programs in C and C++ can be readily executed in a Ch shell. Therefore, the Ch command shell is an ideal solution for teaching and learning C/C++. An instructor can use Ch interactively in classroom presentations with a laptop to quickly illustrate programming features, especially when answering students' questions. Learners can also quickly try out different features of C/C++ without tedious compile/link/execute/debug cycles. To assist beginners in learning, Ch has been especially developed with many helpful warning and error messages, as an alternative to crashing with cryptic, arcane messages like *segmentation fault* and *bus error*.

A Ch shell can be launched by running the command `ch`. In Windows, a Ch command shell can also be conveniently launched by clicking its icon shown in Figure 9 on the desktop. Assume the user account is the administrator, after a Ch shell is launched in Windows, by default, the screen prompt becomes

```
C:/Documents and Settings/Administrator>
```

where `C:/Documents and Settings/Administrator` is the user's *home directory* as shown in Figure 10 on the desktop. The displayed directory `C:/Documents and Settings/Administrator` is also called the *current working directory*. If the user account is not the administrator, the account name *Administrator* in the home directory shall be changed accordingly. The prompt indicates that the system is in a Ch shell and is ready to accept the user's terminal keyboard input. The default prompt in a Ch shell can be reconfigured. If the input typed in is syntactically correct, it will be executed successfully. Upon completion of the execution, the system prompt > will appear again. Otherwise, it prints out the corresponding error messages to assist the user in debugging the program.



```
                                Ch
                 Professional edition, version 5.1.0.12751
               (C) Copyright 2001-2006 SoftIntegration, Inc.
                     http://www.softintegration.com
C:/Documents and Settings/Administrator> printf("Hello, world")
Hello, world
C:/Documents and Settings/Administrator>
```

Figure 10. A Ch command shell.

Table 1. Portable commands for handling files.

Command	Usage	Description
cd	cd	change to the home directory
	cd *dir*	change to the directory *dir*
cp	cp *file1 file2*	copy *file1* to *file2*
ls	ls	list contents in the working directory
mkdir	mkdir *dir*	create a new directory *dir*
pwd	pwd	print (display) the name of the working directory
rm	rm *file*	remove *file*
chmod	chmod +x *file*	change the mode of *file* to make it executable

All statements and expressions of C can be executed interactively in a Ch command shell as shown in Figure 10. For example, the output `Hello, world` can be obtained by calling the function **printf**() interactively as shown below.

```
C:/Documents and Settings/Administrator> printf("Hello, world")
Hello, world
```

In comparison with Figure 10, the last prompt `C:/Documents and Settings/Administrator>` is not displayed in the presentation of this book. Note that the semicolon at the end of a statement in a C program is optional when the corresponding statement is executed in command mode. There is no semicolon in calling the function **printf** in the above execution.

2.1 Portable Commands for Handling Files.

At the system prompt >, not only C programs and statements, but also any other commands, such as **pwd** for printing the current working directory, can be executed. In this scenario, Ch is used as a command shell in the same manner as Bash-shell, C-shell, or Korn-shell in Unix or MS-DOS shell in Windows.

There are hundreds of commands along with their online documentation in the system. No one knows all of them. Every computer wizard has a small set of working tools that are used all the time, plus a vague idea of what else is out there.

In this section, we will describe how to use the most commonly used commands for handling files listed in Table 1 by examples. It should be emphasized again that these commands running in the Ch shell are portable across different platforms. Using these commands, a user can effectively manipulate files on the system to run C programs.

Assume that Ch is installed in the directory `C:/Ch` in Windows by default. The current working directory is `C:/Documents and Settings/Administrator`, which is also the user's home directory. The application of portable commands for file handling can be illustrated by interactive execution of commands in a Ch shell below.

```
C:/Documents and Settings/Administrator> mkdir c99
C:/Documents and Settings/Administrator> cd c99
C:/Documents and Settings/Administrator/c99> pwd
C:/Documents and Settings/Administrator/c99
C:/Documents and Settings/Administrator/c99> cp C:/Ch/demos/bin/hello.c hello.c
C:/Documents and Settings/Administrator/c99> ls
hello.c
C:/Documents and Settings/Administrator/c99>
```

As shown in *Usage* in Table 1, the command **mkdir** takes one argument as a directory to be created. We first create a directory called `c99` using the command

```
mkdir c99
```

Then, change to this new directory `C:/Documents and Settings/Administrator/c99` using command

```
cd c99
```

Next, display the current working directory with the command

```
pwd
```

A C program `hello.c` shown in Program 2 in the directory `C:/Ch/demos/bin` is copied to the working directory with the same file name using the command

```
cp C:/Ch/demos/bin/hello.c hello.c
```

Finally, files in the current directory are listed using the command

```
ls
```

At this point, there is only one file `hello.c` in the directory `C:/Documents and Settings/Administrator/c99`. It is recommended that you save all your developed C programs in this directory.

2.2 Interactive Execution of Programs

It is very simple and easy to run C programs interactively without compilation in a Ch shell. For example, assume that `C:/Documents and Settings/Administrator/c99` is the current working directory as presented in the previous section. The program `hello.c` in this directory can be executed in Ch to get the output of `Hello, world` as shown below.

```
C:/Documents and Settings/Administrator/c99> hello.c
Hello, world
C:/Documents and Settings/Administrator/c99> _status
0
```

The exit code from executing a program in a Ch command shell is kept in the system variable **_status**. Because the program `hello.c` has been executed successfully, the exit code is 0 as shown in the above output.

In Unix, in order to readily use the C program `hello.c` as a command, the file has to be executable. The command **chmod** can change the mode of a file. The following command

```
chmod +x hello.c
```

will make the program `hello.c` executable so that it can run in a Ch command shell.

2.3 Setup Paths and Finding Commands in Ch

When a command is typed into a prompt of a command shell for execution, the command shell will search for the command in prespecified directories. In a Ch shell, the system variable **_path** of string type contains the directories to be searched for the command. Each directory is separated by a semicolon inside the string **_path**. When a Ch command shell is launched, the system variable **_path** contains some default search paths. The user can add new directories to the search paths for the command shell by using the string function **stradd()** which adds arguments of string type and returns it as a new string. For example, the directory `C:/Documents and Settings/Administrator/c99` is not in the search paths for a command. If you try to run program `hello.c` in this directory when the current working directory is `C:/Documents and Settings/Administrator`. The Ch shell will not be able to find this program as shown below and give two error messages.

```
C:/Documents and Settings/Administrator> hello.c
ERROR: variable 'hello.c' not defined
ERROR: command 'hello.c' not found
```

When Ch is launched or a Ch program is executed, by default, it will execute the startup file **.chrc** in Unix or **_chrc** in Windows in the user's home directory if it exists. In the remaining presentation, it is assumed that Ch is used in Windows with a startup file **_chrc** in the user's home directory. This startup file typically sets up the search paths for commands, functions, header files, etc. By default, there is no startup file in a user's home directory. The system administrator may add such a startup file in a user's home directory. However, the user can execute Ch with the option -d as follows

```
ch -d
```

to copy a sample startup file from the directory **CHHOME**/config/ to the user's home directory if there is no startup file in the home directory yet. Note that **CHHOME** is not the string **"CHHOME"**, instead it uses the file system path under which Ch is installed. For example, by default, Ch is installed in `C:/Ch` in Windows and `/usr/local/ch` in Unix. In Windows, the command in a Ch shell below

```
C:/Documents and Settings/Administrator> ch -d
```

will create a starup file **_chrc** in the user's home directory `C:/Documents and Settings/Administrator`. This local Ch initialization startup file `_chrc` can be opened for editing the search paths by ChSciTE editor as shown in Figure 11.

To include the directory `C:/Documents and Settings/Administrator/c99` in the search paths for a command, the following statement

```
_path = stradd(_path, "C:/Documents and Settings/Administrator/c99;");
```

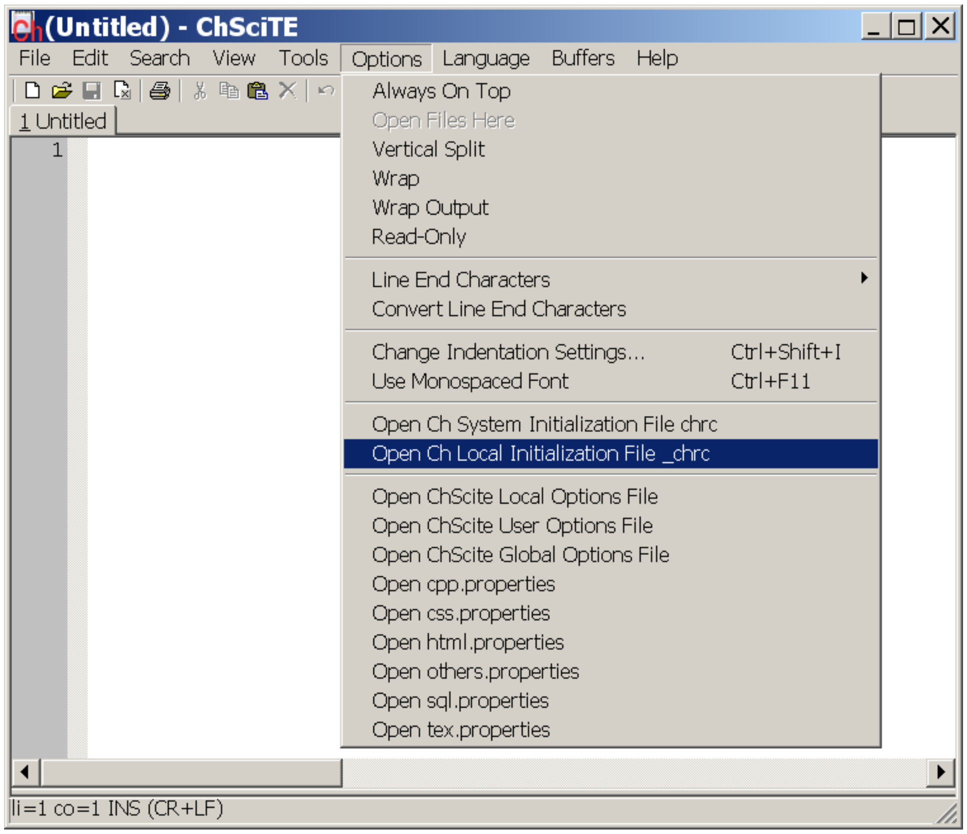


Figure 11. Open the local Ch initialization startup file for editing.

needs to be added to the startup file **_chrc** in the user's home directory so that the command hello.c in this directory can be invoked regardless of what the current working directory is. After the directory `C:/Documents and Settings/Administrator/c99` has been added to the search path, **_path**, you need to restart a Ch command shell. Then, you will be able to execute the program `hello.c` in this directory as shown below.

```
    C:/Documents and Settings/Administrator> hello.c
    Hello, world
```

Similar to **_path** for commands, the header files in Ch are searched in directories specified in the system variable **_ipath**. Each path is also delimited by a semicolon. For example, the statement below

```
   _ipath = stradd(_ipath, "C:/Documents and Setting/Administrator/c99;");
```

adds the directory `C:/Documents and Setting/Administrator/c99` to the search paths for header files. One can also add this directory to the function file search paths by the statement

```
   _fpath = stradd(_fpath, "C:/Documents and Setting/Administrator/c99;");
```

In Unix, the search paths for commands by default do not contain the current working directory. To include the current working directory in the search paths for a command, the following statement

```
   _path = stradd(_path, ".;");
```

in startup file **.chrc** in the user's home directory needs to be added. Function call `stradd(_path, ".;")` adds the current directory represented by '.' to the system search paths **_path**.

3 Interactive Execution of Expressions and Statements

For simplicity, only the prompt `>` in a Ch command shell will be displayed in the remaining presentation. If a C expression is typed in, it will be evaluated by Ch. The result then will be displayed on the screen. For example, if the expression `1+3*2` is typed in, the output will be 7 as shown:

```
> 1+3*2
7
```

Any valid C expression can be evaluated in a Ch shell. Therefore, Ch can be conveniently used as a calculator.

As another example, one can declare a variable at the prompt. Then, use the variable in the subsequent calculations as shown:

```
> int i
> sizeof(int)
4
> i = 30
30
> printf("%x", i)
1e
> printf("%b", i)
11110
> i = 0b11110
30
> i = 0x1E
30
> i = -2
-2
> printf("%b", i)
11111111111111111111111111111110
> printf("%32b", 2)
00000000000000000000000000000010
```

In the above C statements, variable `i` is declared as int type with 4 bytes. Then, the integer value 30 for `i` is then displayed in decimal, hexadecimal, and binary numbers. The integral constants in different number systems can also be assigned to variable `i`. The two's complement representation of the negative number -2 is also displayed. Characteristics for all other data types in C can also be presented interactively. Different format specifiers for the families of input function fscanf() and output function **fprintf**() using file streams opened by function fopen() can also be tried out this way.

By default, a value of float or double type is displayed with two four digits after the decimal point, respectively. For example,

```
> float f = 10
> 2*f
20.00
> double d = 10
> d
10.00000
```

All C operators can be used interactively as shown:

```
> int i=0b100, j = 0b1001
> i << 1
8
> printf("%b",  i|j)
1101
```

The concept of pointers and addresses of variables can be illustrated as shown:

```
> int i=10, *p
> &i
1eddf0
> p = &i
1eddf0
> *p
10
> *p = 20
20
> i
20
```

In this example, the variable `p` of pointer to int points to the variable `i`. The relation of arrays and pointers can be illustrated as follows:

```
> int a[5] = {10,20,30,40,50}, *p;
> a
1eb438
> &a[0]
1eb438
> a[1]
20
> *(a+1)
20
> p = a+1
1eb43c
> *p
20
> p[0]
20
```

Expressions `a[1], *(a+1), *p`, and `p[0]` all refer to the same element. Multi-dimensional arrays can also be handled interactively. The boundary of an array is checked in Ch to detect potential bugs. For example,

```
> int a[5] = {10,20,30,40,50}
> a[-1]
WARNING: subscript value -1 less than lower limit 0
10
> a[5]
WARNING: subscript value 5 greater than upper limit 4
50
> char s[5]
> strcpy(s, "abc")
abc
> s
abc
> strcpy(s, "ABCDE")
ERROR: string length s1 is less than s2 in strcpy(s1,s2)
ABCD
> s
ABCD
```

The allowed indices for array `a` of 5 elments are from 0 to 4. Array `s` can only hold 5 characters including a null character. Ch can catch bugs in existing C code related to the array boundary overrun.

The alignment of a C structure or C++ class can also be examined as shown:

```
> struct tag {int i; double d;} s
> s.i =20
20
> s
.i = 20
.d = 0.0000
> sizeof(s)
16
```

In this example, although the sizes of int and double are 4 and 8, respectively, the size of structure `s` with two fields of int and double types is 16, instead of 12, for the proper alignment.

4 Interactive Execution of Functions

A program can be divided into many separate files. Each file consists of many related functions, at the top level, which are accessible to any part of a program. All functions in the C standard libraries can be executed interactively and can be used inside user defined functions. For example, in the interactive execution:

```
> srand(time(NULL))
> rand()
4497
> rand()
11439
> double add(double a, double b) {double c; return a+b+sin(1.5);}
> double c
> c = add(10.0, 20)
30.9975
```

The random number generator function `rand()` is seeded with a time value in `srand(time(NULL)`. Function `add()` which calls type-generic mathematical function `sin()` is defined at the prompt and then used.

A file that contains more than one function definition is usually suffixed with `.ch` to identify itself as part of a Ch program. One can create a function file in a Ch programming environment. A *function file* in Ch is a file that contains only one function definition. The name of a function file ends in `.chf`, such as `addition.chf`. The names of the function file and function definition inside the function file must be the same. The functions defined using function files are treated as if they were the system built-in functions in Ch.

Similar to **_path** for commands, a function is searched based on the search paths in the system variable **_fpath** for function files. Each path is delimited by a semicolon. By default, the variable **_fpath** contains the paths `lib/libc, lib/libch, lib/libopt,` and `libch/numeric` in the home directory of Ch. If the system variable **_fpath** is modified interactively in a Ch shell, it will be effective only for functions invoked in the current shell interactively. For running scripts, the setup of function search paths in the current shell will not be used and inherited in subshells. In this case, the system variable **_fpath** can be modified in startup file **_chrc** in Windows or **.chrc** in Unix at the user's home directory.

For example, if a file named `addition.chf` contains the program shown in Program 1, the function `addition()` will be treated as a system built-in function, which can be called to compute the sum $a+b$ of two input arguments a and b. Assume that the function file `addition.chf` is located at `C:/Documents and Settings/Administrator/c99/addition.chf`, the directory `C:/Documents and Settings/Administrator/c99` should be added to the function search path in the startup file **.chrc** in Unix or **_fpath** in Windows in the user's home directory with the following statement.

```
_fpath=stradd(_fpath, "C:/Documents and Settings/Administrator/c99;");
```

```
/* File: addition.chf */
int addition(int a, int b) {
    int c;
    c = a + b;
    return c;
}
```

Program 1. Function file `addition.chf`.

```
/* File: program.ch */
int main() {
    int a, b, c;

    a = 2;
    b = 3;
    c = addition(a, b);
    printf("c = %d\n ", c);
    return 0;
}
```

Program 2. Program using function file `addition.chf`.

Function `addition()` then can be used either interactively in command mode as shown below,

```
> int i = 9
> i = addition(3, i)
12
```

or inside programs. In Program 2, the function `addition()` is called without a function prototype in the **main()** function so that the function prototype defined inside the function file `addition.chf` will be invoked. The output of Program 2 is `c = 5`. If the search paths for function files have not been properly setup, a warning message such as

```
WARNING: function 'addition()' not defined
```

will be displayed, when the function `addition()` is called.

When a function is called interactively in a Ch shell, the function file will be loaded. If you modify a function file after the function has been called, the subsequent calls in the command mode will still use the old version of the function definition that had been loaded. To invoke the modified version of the new function file, you can either remove the function definition, say `addition`, in the system using the command

```
> remvar addition
```

or start a new Ch shell.

5 Interactive Execution of C++ Programming Features

Not only C++ programs can be executed in Ch, but also classes and some C++ features are supported in Ch as shown below for interactive execution of C++ code.

```
> int i
> cin >> i
10
> cout << i
10
```

```
> class tagc {private: int m_i; public: void set(int); int get(int &);}
> void tagc::set(int i) {m_i = 2*i;}
> int tagc::get(int &i) {i++; return m_i;}
> tagc c
> c.set(20)
> c.get(i)
40
> i
11
> sizeof(tagc)
4
```

The input and output can be handled using `cin` and `cout` in C++. The public method `tagc::set()` sets the private member field `m_i`, whereas the public method `tagc::get()` gets its value. The argument of method `tagc::get()` is passed by reference. The size of the class `tagc` is 4 bytes which does not include the memory for member functions.

6 High-Level Graphical Plotting

Graphical plotting is important for the visualization and interpretation of numerical results. C does not support graphical plotting by default. C programmers typically generate data in a file and then use other software packages, such as Excel, to plot the data in the file. This is an inconvenient process for the development of algorithms. Ch provides the simplest possible solution for two and three dimensional graphical plottings within the framework of C/C++. Two and three dimensional graphical plots can be easily generated in Ch by plotting functions or member functions in a graphical library. Plots in Ch can be created from data arrays or files, and can be displayed on the screen, saved as an image file in different file formats, or output to the *stdout* stream in a proper image format for display in a Web browser through a Web server.

6.1 Plotting Function plotxy()

Function **plotxy()** is a high-level plotting function defined in header file **chplot.h**. Function **plotxy()** is useful for generating two-dimensional plots. The function **plotxy()** can be taken as if it had the following function prototype.

```
int plotxy(double x[], double y[],
           char *title, char *xlabel, char *ylabel);
```

The array `x` stores data for the x-axis, while array `y` stores data for the y-axis. The title and labels for axes are specified by the arguments *title*, *xlabel*, and *ylabel*, respectively.

Program 3 provides a simple example of using the **plotxy()** function for generating the plot of a sine wave for x from 0 to 2π radians. The variables `x0` and `xf` represent the initial and final values of x, respectively. The macro NUMPOINTS is defined as the number of 37 points. A **for**-loop is used to generate 37 data points for the plot, which is shown in Figure 12. The statement

```
x[i] = x0+i*(xf - x0)/(NUMPOINTS-1);
```

generates the x-coordinates of the plot with the number of points specified in the macro NUMPOINTS for x from `t0` to `tf`. Note that the title of the plot and axes are specified as strings `"function sin(x)"`, `"x"`, and `"sin(x)"`, respectively.

6.2 Plotting Function fplotxy()

Function **plotxy()** in the previous section can be used to plot data stored in arrays. To plot a function with a single variable, the plotting function **fplotxy()** can be used. The function **fplotxy()** can be taken as if it had the following function prototype.

```
/* File: plotxy.cpp */
#include <math.h>
#include <chplot.h>

#define NUMPOINTS 37

int main() {
    double x[NUMPOINTS], y[NUMPOINTS], x0, xf;
    int i;

    x0 = 0;
    xf = 2*M_PI;
    for(i=0; i<NUMPOINTS; i++)  {
        x[i] = x0 + i*(xf - x0)/(NUMPOINTS-1);
        y[i] = sin(x[i]);
    }
    plotxy(x, y, "function sin(x)", "x", "sin(x)");
    return 0;
}
```

Program 3. Plotting data using function plotxy().

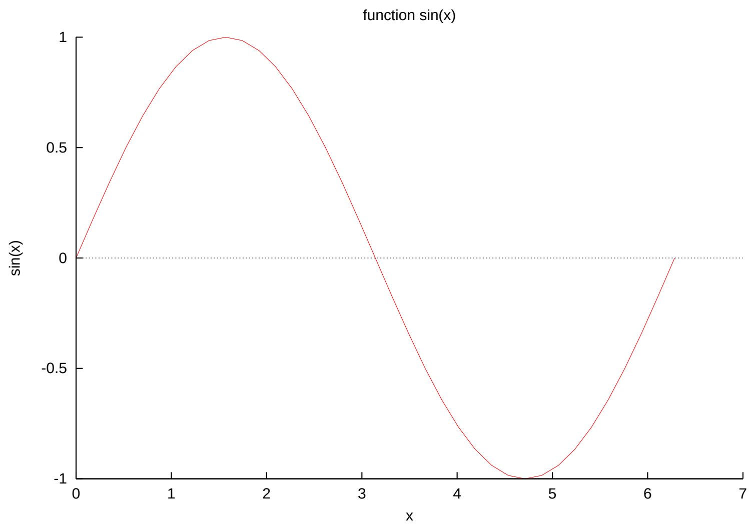


Figure 12. Output plot of Program 3.

```
/* plot a function in Ch */
#include<math.h>
#include<chplot.h>

double func(double x) {
   return sin(x);
}

int main() {
   double x0 = 0, xf = 2*M_PI;
   int num = 37;

   fplotxy(func, x0, xf, num, "function sin(x)", "x", "sin(x)");
}
```

Program 4. Plotting a function using fplotxy().

```
int fplotxy(double (*func)(double), double x0, double xf, int num,
            char *title, char *xlabel, char *ylabel);
```

It plots a function with the variable *x* in the range *x0* $\leq$ *x* $\leq$ *xf*. The function to be plotted, *func*, is specified as a pointer to a function that takes an argument of double type and returns a double type. The arguments *x0* and *xf* are the end-points of the range to be plotted. The argument *num* specifies how many points in the range are to be plotted. The number of points plotted are evenly spaced in the range. The remaining three arguments label the title and axes of a plot.

As an example, Program 4 plots the function $\sin()$ from 0 to 2π shown in Figure 12.

6.3 Plotting Function fplotxyz()

Function **fplotxyz()** can be used to plot a function with two variables. The function **fplotxyz()** is prototyped as follows.

```
int fplotxyz(double (*func)(double, double), double x0, double xf,
             int numx, int numy, char *title, char *xlabel,
             char *ylabel, char *zlabel);
```

It generates a three-dimensional plot for a function with two variables *x* and *y* in the range *x0* $\leq$ *x* $\leq$ *xf* and *y0* $\leq$ *y* $\leq$ *yf*. The function to be plotted, *func*, is specified as a pointer to a function that takes two **double** arguments and returns a **double**. The arguments *x_num* and *y_num* specify how many points in the *x* and *y* ranges to be plotted, respectively. The remaining four arguments label the title and axes of a plot.

As an example, Program 4 uses the plotting function **fplotxy()** to plot the function $\cos(x)\sin(y)$ for *x* and *y* in the range *-3* $\leq$ *x* $\leq$ *3* and *-4* $\leq$ *y* $\leq$ *4*. It uses 80 points for the x coordinates and 100 points for the y coordinates. The generated plot is shown in Figure 13.

6.4 Copying and Printing a Plot

In Windows, the plot displayed in Figure 12 can be copied to the clipboard first. Then, paste it to other programs such as a Word document. Click the plot icon on the upper left corner of the plot, it will bring up several menus. Click `Copy to Clipboard` under the `Options` menu as shown in Figure 14, it will save the plot in the clipboard first. The plot can be printed out by clicking `Print` under the `Options` menu. Click the line in output with red color in Figure 6.

6.5 Using a Plotting Class to Plot Multiple Sets of Data

The plotting capabilities in Ch are implemented in a plotting class **CPlot**. The function call

```
/* plot a function in Ch */
#include<math.h>
#include<chplot.h>

double func(double x) {
   return sin(x);
}

int main() {
   double x0 = 0, xf = 2*M_PI;
   int num = 37;

   fplotxy(func, x0, xf, num, "function sin(x)", "x", "sin(x)");
}
```

Program 5. Plotting a function with two variables using fplotxyz().

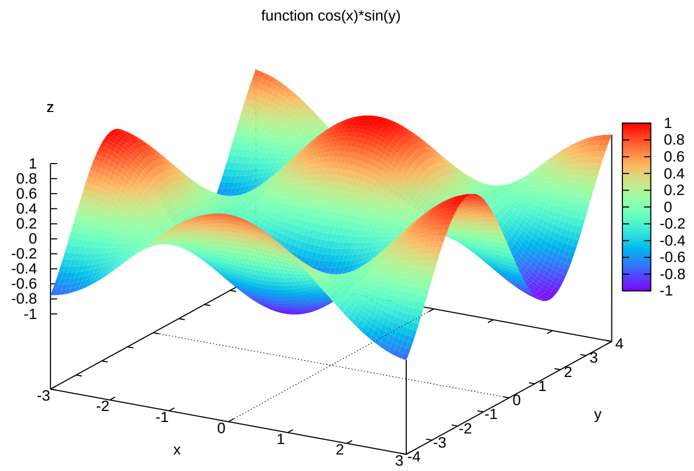


Figure 13. Output plot of Program 5.

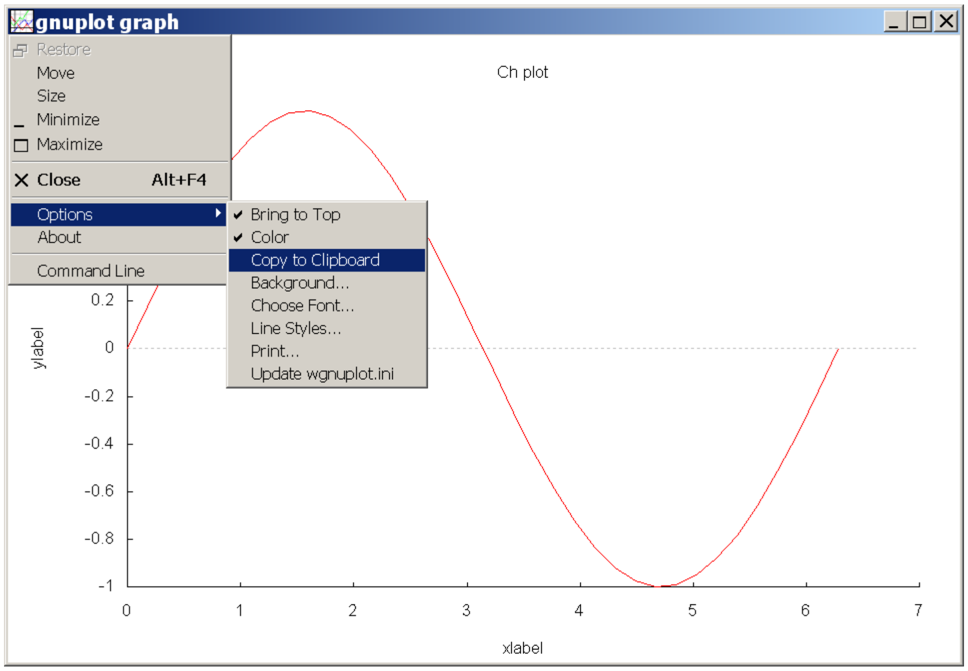


Figure 14. Copy a plot to the clipboard.

```
plotxy(x, y, title, xlabel, ylabel);
```

is equivalent to

```
class CPlot plot;
plot.data2DCurve(x, y, n);
plot.title(title);
plot.label(PLOT_AXIS_X, xlabel);
plot.label(PLOT_AXIS_Y, ylabel);
plot.plotting();
```

as shown in Program 6, which generates the same plot displayed in Figure 12.

The data for plotting of 2D curve are added to an instance of **CPlot** class by the member function

```
int CPlot::data2DCurve(double x[], double y[], int n);
```

Both one-dimensional arrays `x` and `y` have the same number of elements of size `n`.

The title and labels on axes are annotated using corresponding member functions

```
void CPlot::title(char *title);
```

and

```
void CPlot::label(int axis, char *label);
```

respectively. The argument *axis* of member function **CPlot**::**label**() is the axis to be set. The valid macros for *axis* are listed in Table 2 and defined in header file `chplot.h`. At the point where member function **CPlot**::**plotting**() is called, a plot is generated.

Many other member functions of the plotting class can be used to specify the desired features for the generated plot and create different output.

As an example for plotting with multiple data sets, a plot with two sets of data for sine and cosine functions from 0 to 360 degrees with legends shown in Figure 15 is generated by Program 7. In Program 7, array `x` contains the

```
/* File: plotclass.cpp */
#include <math.h>
#include <chplot.h>

#define NUMPOINTS 36

int main() {
    double x[NUMPOINTS], y[NUMPOINTS], x0, xf;
    char title[] = "function sin(x)",
         xlabel[] = "x",
         ylabel[] = "sin(x)";
    int i;
    CPlot plot;

    x0 = 0;
    xf = 2*M_PI;
    for(i=0; i<NUMPOINTS; i++)  {
        x[i] = x0 + i*(xf - x0)/(NUMPOINTS-1);
        y[i] = sin(x[i]);
    }
    plot.data2DCurve(x, y, NUMPOINTS);
    plot.title(title);
    plot.label(PLOT_AXIS_X, xlabel);
    plot.label(PLOT_AXIS_Y, ylabel);
    plot.plotting();
    return 0;
}
```

Program 6. Plotting data usin the plotting class CPlot.

Table 2. The macros for axes.

PLOT_AXIS_X	Select the x axis only.
PLOT_AXIS_Y	Select the y axis only.
PLOT_AXIS_Z	Select the z axis only.
PLOT_AXIS_XY	Select the x and y axes.
PLOT_AXIS_XYZ	Select the x, y, and z axes.

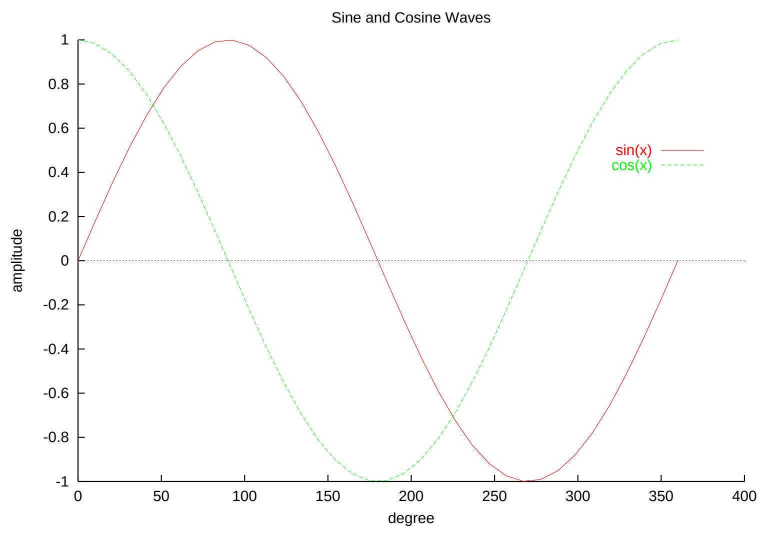


Figure 15. A plot with two sets of data and legends.

data for the x-coordinates. arrays `y1` and `y2` each contain a data set for the y-coordinates. The member function **CPlot**::**data2DCurve**() is called twice to add two sets of data to the plot. A string of *legend* can be added to the plot by member function

```
void CPlot::legend(char *legend, int num);
```

The argument `legend` is a string of characters. The number of data set to which the legend is added is indicated by the second argument *num* of int type. Numbering of the data sets starts with zero. New legends will replace previously specified legends. This member function shall be called after plotting data have been added by the member function **CPlot**::**data2DCurve**().

The member function

```
void CPlot::legendLocation(double x, double y, ... /* [double z] */);
```

specifies the position of the plot legend using plot coordinates *(x, y, y)* of double type. For a two-dimensional plot, only the coordinates *(x, y)* are needed. The position specified is the location of the top part of the space separating the markers and labels of the legend as shown in Figure 16. By default, the location of the legend is near the upper-right corner of the plot.

The range an axis can be set by member function **CPlot**::**axisRange**(),

```
void CPlot::axisRange(int axis, double minimum, double maximum,
                      ... /* [double incr] */);
```

The valid macros for *axis* are listed in Table 2. The minimum and maximum values for an axis are given in second and third arguments, respectively. The increment between tick marks is given in *incr*. By default, this value is calculated internally. For example, function call `plot.axisRange(PLOT_AXIS_X, 0, 360, 30)` sets the range of the `x`-axis from 0 to 360 degrees with tick marks at every 30 degrees.

```
/* File: legend.cpp */
#include<math.h>
#include<chplot.h>

#define NUMPOINTS 36

int main() {
    int i;
    double x0, xf, x[NUMPOINTS], y1[NUMPOINTS], y2[NUMPOINTS];
    char *title="Sine and Cosine Waves",
         *xlabel="degree", *ylabel="amplitude";
    class CPlot plot;

    x0 = 0;
    xf = 360;
    for(i = 0; i < NUMPOINTS; i++) {
        x[i]  = x0 + i*(xf - x0)/(NUMPOINTS-1);
        y1[i] = sin(x[i]*M_PI/180);
        y2[i] = cos(x[i]*M_PI/180);
    }
    plot.data2DCurve(x, y1, NUMPOINTS);
    plot.data2DCurve(x, y2, NUMPOINTS);
    plot.legend("sin(x)", 0);
    plot.legend("cos(x)", 1);
    plot.legendLocation(350, 0.5);
    plot.title(title);
    plot.label(PLOT_AXIS_X, xlabel);
    plot.label(PLOT_AXIS_Y, ylabel);
    plot.plotting();
    return 0;
}
```

Program 7. A program for plotting two sets of data with legends.

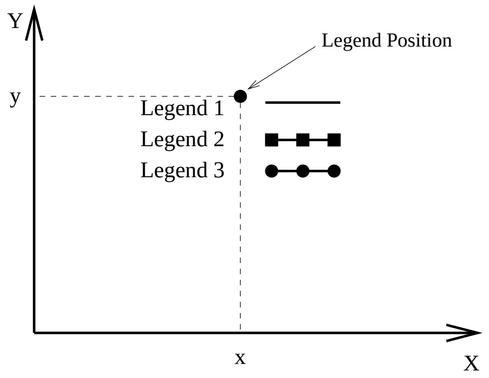


Figure 16. The position for legend.

Exercises

1. Describe four methods that Ch can be launched in Windows after installation.
2. In a Ch command shell, type

   ```
   ch -d
   ```

   The startup configration file _chrc in Windows or .chrc in Unix will be copied from CHHOME/config directory to your home directory, where CHHOME is the home directory for Ch, such as `C:/Ch` for Windows and `/usr/local/ch` for Unix. What's your home directory?
3. In a Ch command shell, create a directory (folder) called `C99` in your home directory using command

   ```
   mkdir c99
   ```

4. Assume `YourHomeDir` is your home directory, the startup file is _chrc for Windows and .chrc for Unix in your home directory. Modify your startup file using ChSciTE by clicking `Open Ch Local Initialization File` under `Option` menu or any other text editor.

   (a) Add `YourHomeDir/c99` in the command search path for Ch. This can be done by adding

   ```
   _path = stradd(_path, "YourHomeDir/c99;");
   ```

   in your startup file.

   (b) If you use Mac, Linux, or Unix, add the current directory to the command search path for Ch by adding

   ```
   _path = stradd(_path, ".;");
   ```

   in your startup file.

   (c) Add the directory `YourHomeDir/c99` to function file search pathes for Ch programs by adding

   ```
   _fpath = stradd(_fpath, "YourHomeDir/c99;");
   ```

   in your startup file.

   (d) Add `YourHomeDir/c99` in the header file search path for Ch shell. for Ch programs by adding

   ```
   _ipath = stradd(_ipath, "YourHomeDir/c99;");
   ```

   in your startup file.

5. Setup an alias `c99` in Ch shell that will change directory to `YourHomeDir/c99` by adding

   ```
   alias("c99", "cd YourHomeDir/c99");
   ```

   to your startup file.
6. What is the first statement that is typically included in a Ch shell program so that other shell programs, such as bash, can recognize it as a Ch script.
7. Write a program `plotting.c` to plot function $f(x) = 2x^2 * \sin(x)$ versus x when x varies within the range of $-8.5 \text{ radian} \leq x \leq 8.5 \text{ radian}$ using plotting function `plotxy()`.
8. Write a program to plot functions $y1 = \exp(x) + \sqrt{x}$ and $y2 = \sin(x)$ from $x = 0$ to π without labels in both x and y axes. The plot shall have legends of `y1` and `y2` for the curves of these two functions with 50 points for each curve.

Index