Harry H. Cheng

Ch: A C/C++ Interpreter for Script Computing

Interactive computing in C

For some tasks, C and its compile/link/execute/debug process are not productive. As computer hardware becomes cheaper and faster, to be productive and cost effective, script computing in C/C++ can be an appealing solution. To this end, we have developed Ch, an embeddable C/C++ interpreter for cross-platform scripting, shell programming, 2D/3D plotting, numerical computing, and embedded scripting [1].

As a complete C interpreter, Ch supports all language features and standard libraries of the ISO C90 Standard. It also supports an increasing number of C/C++ libraries including POSIX, TCP/IP socket, Winsock, Win32, X11/Motif, GTK+, OpenGL, ODBC, SQLite, CGI, LAPACK, LDAP, PCRE, Gnome Libxml2, Oracle XDK for XML, NAG statistics library, Intel OpenCV for computer vision, ImageMagick for image processing, SigLib for signal processing, National Instruments' NI-DAQ, and NI-Motion.

Ch supports most new features added in the ISO C99, such as complex numbers, variable length arrays (VLA), IEEE floating-point arithmetic, and type-generic mathematical functions. C programmers are encouraged to use these new features because they significantly simplify many programming tasks. Ch also supports classes in C++ for object-based programming.

Harry H. Cheng is a Professor and Director of the Integration Engineering Laboratory at the University of California, Davis. He is the Chief Architect of Ch, an embeddable C/C++ interpreter for script computing. He can be reached at hhcheng@ucdavis.edu.

Ch has borrowed features and ideas from many other languages and software packages. The following is a short list of other languages and software packages that in one way or another have influenced the development of Ch.

- Like the C shell, Ch can be used as a login shell and for shell programming. But, as a superset of C, Ch is a genuine C shell.
- Like Basic, Ch is designed for, and has been used by, beginners who have no prior programming experience.
- Like Perl, Ch has the built-in string and other features for text handling, and can be used for common gateway interface (CGI) applications in a web-server environment.
- Like Java, Ch can be used for Internet computing. Safe Ch uses a sandbox model for security control. A Ch applet can be executed across a network on different computer platforms on the fly.
- Like Tcl/TK, Ch can be embedded as a scripting engine in other applications and supports GTK+, Win32, and X/Motif for GUI development.
- Like Fortran 90, Ch can be used for scientific computing.
- Like MATLAB/Mathematica, Ch has computational arrays and can be used for numerical computing, graphical plotting, and rapid prototyping.

The relationship of Ch to some of these languages and software packages is shown in Figure 1.

Unlike other languages and software packages, Ch bridges the gap between low-level languages and very high-level languages. As a superset of C, Ch retains low-level features of C such as accessing memory for hardware interface. However, unlike C, Ch can be considered a very high-level language (VHLL) environment. It makes hard things easy and easy things easier. Ch extends C with many high-level features such as string type and computational arrays as first-class objects, as in Fortran 90 and MATLAB for linear algebra and matrix computations. Ch supports shell programming with a built-in string type. Some problems, which might take thousands of lines of C code, can be easily solved with only a few lines of Ch code.

Furthermore, Ch is designed to be platform independent. It can run in a heterogeneous computing environment with different computer hardware and operating systems including Windows, Mac OS X, Linux in both Intel and PPC architectures, UNIX, FreeBSD, and QNX. A program developed in one platform can run in any other platforms.

In this article, I will present some unique features offered by this C/C++ interpreter for script computing and illustrate how they are used in applications. Examples presented are portable across different platforms. One can readily try these examples by downloading Ch from http://www.softintegration.com/.

Interactive Execution Of C/C++ Statements And Expressions

I regularly teach engineering students on introductory computer programming in C. I use Ch interactively in classroom presentations using a laptop. Ch allows me to illustrate programming features quickly, especially in answering students' questions. Learners can also quickly try out different features of C without tedious compile/link/execute/debug cycles. To help beginners learn, Ch has been specially developed with many helpful warning and error messages, as an alternative to crashing with cryptic, arcane messages like "segmentation fault" and "bus error."

All C statements and expressions can be executed interactively in a Ch command shell as shown in Figure 2. The output from the system resulting from executing an expression, statement, or command in Ch is displayed on the screen. For example, the output "Hello, world" can be obtained by calling function

(continued from page 6)

`printf()` interactively as shown in Figure 2. 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 function `printf` in the previous execution. The default prompt in a Ch shell can be reconfigured. For simplicity, I'll only show the prompt `>` in a Ch command shell for the rest of this article.

If a C expression is typed in, it will be evaluated by Ch. The result will then 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:

```
> 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 these 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 2's complement representation of the negative number –2 is displayed as well. 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. All C operators can be used interactively:

```
> 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
```

Figure 1: ***Relationship of Ch to some other languages and software packages.***

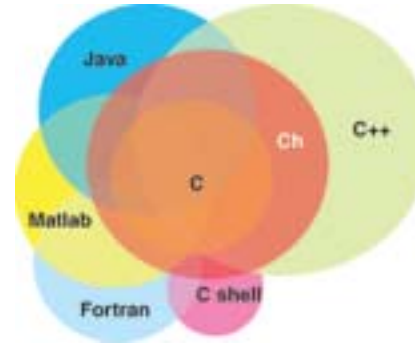


Figure 2: ***The user interface in a Ch command shell.***

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. Multidimensional arrays can also be handled interactively. The boundary of an array is checked in Ch to detect potential bugs; see Example 1.

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 of 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
```

(continued from page 8)

```
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.

Interactive and Interpretive Execution Of C/C++ Functions and Programs

Not only C statements and expressions, but also C functions and programs can be interactively executed in Ch. All functions in the C standard libraries can be executed interactively and can be used inside user-defined functions. For example, in Example 2, 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 function can also be invoked from a function file. A function file in Ch is a file with the extension .chf that contains only one function definition. The names of the function file and function definition inside the function file must be the same. A function is searched based on the search paths in the system variable `_fpath` for function files. The system variables can be setup in the startup configuration file .chrc for UNIX and chrc in Windows in the user's home directory. For example, Listing 1 is the function file addition.chf for function `addition()`. If file addition.chf is located in a directory specified in `fpath`, function `addition()` can be used inside a program or command line:

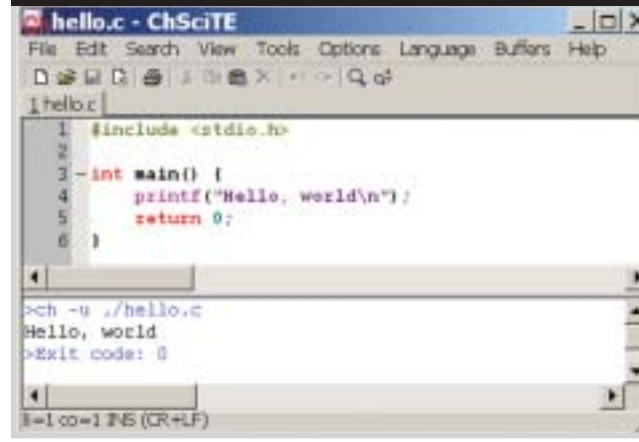
```
> int i = 10
> i = addition(10, i)
20
```

All functions, including function `main()`, in C are at the same level; functions cannot be defined inside other functions. In other words, there are no internal procedures in C. Ch extends C with nested functions. A function can define other functions inside itself. For example, the nested function `nestedfunc()` defined in Listing 2 can be executed interactively as follows:

Example 1: ***Array-boundary checking in Ch.***

```
> 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
```

Figure 3: ***Editing and running a program using ChSciTE IDE.***




```
> nestedfunc(10, 20)
60
```

where function `innerfunc()` is defined inside function `nestedfunc()`. With nested functions, details of one functional module can be hidden from the other modules that do not need to know about them. Each module can be studied independent of others. Software maintenance is the major cost of a program. People who were not involved in the original design often do the most program maintenance. Nested functions modularize a program, thus clarifying the whole program and easing the pain of making changes to modules written by others. Nested functions are useful for information hiding and modular programming. They are complementary to data encapsulation in object-oriented programming in C++. In some applications, using nested functions is simpler than using member functions of classes.

Classes and some other C++ features are also supported in Ch for interactive execution of C++ code, as shown in Example 3. 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.

C/C++ programs can also be executed interactively without compilation. For example, to run the program hello.c in Listing 3, one can type the command hello.c in a Ch command shell to get the output of "Hello, world":

```
> hello.c
Hello, world
```

Example 2: ***Using C Standard Library functions from the command line.***

```
> 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
```

Listing 1

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

Listing 2

```
/* File: nestedfunc.chf */
int nestedfunc(int a, int b) {
int innerfunc(int i, int j) {
int k;
k = 2*(i+j);
return k;
}
int c;
c = innerfunc(a, b);
return c;
}
```

Listing 3

```
/* File: hello.c */
#include <stdio.h>
int main() {
printf("Hello, world\n");
return 0;
}
```

Ch finds executable commands in directories specified in the system variable `_path`. From other command shells, program hello.c can be executed in Ch without compilation, as follows:

```
% ch hello.c
```

C/C++ programs can also run in Ch from an Integrated Development Environment (IDE) with a graphical user interface (GUI). Many IDEs, including the open-source ChSciTE IDE shown in Figure 3, support Ch by default. ChSciTE has a native graphical user interface in more than 30 localized languages.

To speed up interpretive execution of a C program, one should try to avoid loops in the program. For scientific numerical computations, one can use computational arrays described in the next section for fast execution of algorithms with arrays. Alternatively, the code that is computationally intensive in a script program can be compiled as a binary module. In general, you may find it is necessary to interface binary modules in the following situations:

- The source code of a C/C++ library is not available.
- To run Ch scripts fast by making the frequently called underlying C/C++ functions a binary library.
- To protect the intellectual property and source code from being revealed.

Ch Software Development Kit (SDK) included in the distribution of Ch allows C/C++ scripts (Ch scripts) to interface C/C++ binary libraries without recompilation. C functions `dlopen()`, `dlsym()`, `dlerror()`, `dlclose()` and Ch extension function `dlrunfun()` are used to dynamically load binary modules. They allow Ch scripts to access global variables or call functions in the compiled C/C++ libraries such as static library, shared library, or Dynamically Linked Library (DLL). Ch scripts can callback Ch functions from C/C++ libraries. There is no distinction between the interpreted and compiled code.

The Ch SDK provides a utility to generate wrappers for Ch to interface C/C++ libraries automatically. Ch SDK has been effectively used to interface with standard C libraries such as POSIX, X11/Motif, Win32, GTK+, OpenGL, and ODBC.

The interpretive execution of C functions and programs is suitable for applications that do not take very much CPU time. Unlike other scripting languages, as a superset of C, Ch is particularly suitable for applications that need to interface to hardware, such as automated hardware testing and diagnosis.

The interpretive execution of C functions and programs is ideal for rapid prototyping. Programs can be first developed and tested in Ch. Later, the same code can be compiled in C or C++ compiler for final production. Applications with tens of thousands of lines using functions in libraries such as POSIX, X11/Motif, Win32, GTK+, OpenGL, and ODBC can also effectively run in Ch.

The program compilation presents a serious problem for real-time manipulation of mechatronic systems. For real-time mechatronic systems, the external environment may be different at each execution, so the testing scenario may not be repeatable. During debugging and testing, it is impractical to restart a program from the very beginning every time a change is made or a problem is diagnosed. Ch is time deterministic. It can be used for control of mechatronic systems such as robotics. In a typical real-time application in RTLinux, a hard real-time module runs in the Linux kernel, Ch runs in the user space and communicates with the real-time module through interprocess communication such as FIFO.

Example 3: ***Some C++ features are supported in Ch 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 interpretive execution of C programs is especially suitable for interactive presentations using a laptop in a classroom with a quick response. With an interactive computing environment, instructors can relieve themselves from tedious compile/link/execute/debug cycles, and focus on teaching the knowledge and problem-solving skills. All sample code in a textbook for teaching computer programming in C can readily run in Ch without modification.

Ch is especially suitable for developing web-based interactive content in engineering and science for general applications and distance learning. For example, web-based design and analysis of control systems and mechanisms have been developed and they are available online at http://www.softintegration.com/webservices. The shopping cart on this site was also developed in Ch.

Conclusion

Ch is a complete C interpreter. It supports most new features added in C99 and classes in C++. Ch is embeddable in other applications as a C/C++ scripting engine. It is ideal for cross-platform scripting, shell programming, 2D/3D plotting, numerical computing, and embedded scripting. C/Ch/C++ allow users to use one language, anywhere and everywhere, for any programming tasks. Ch lowers the barrier for system programmers to do scripting. Whether you are a novice computer user or experienced C/C++ programmer, I hope that Ch will make your programming tasks more enjoyable.

Acknowledgment

I would like to thank Tom MacDonald for his suggestions and comments on this article.

Reference

[1] Ch: An Embeddable C/C++ Interpreter, http://www.softintegration.com/. ❑