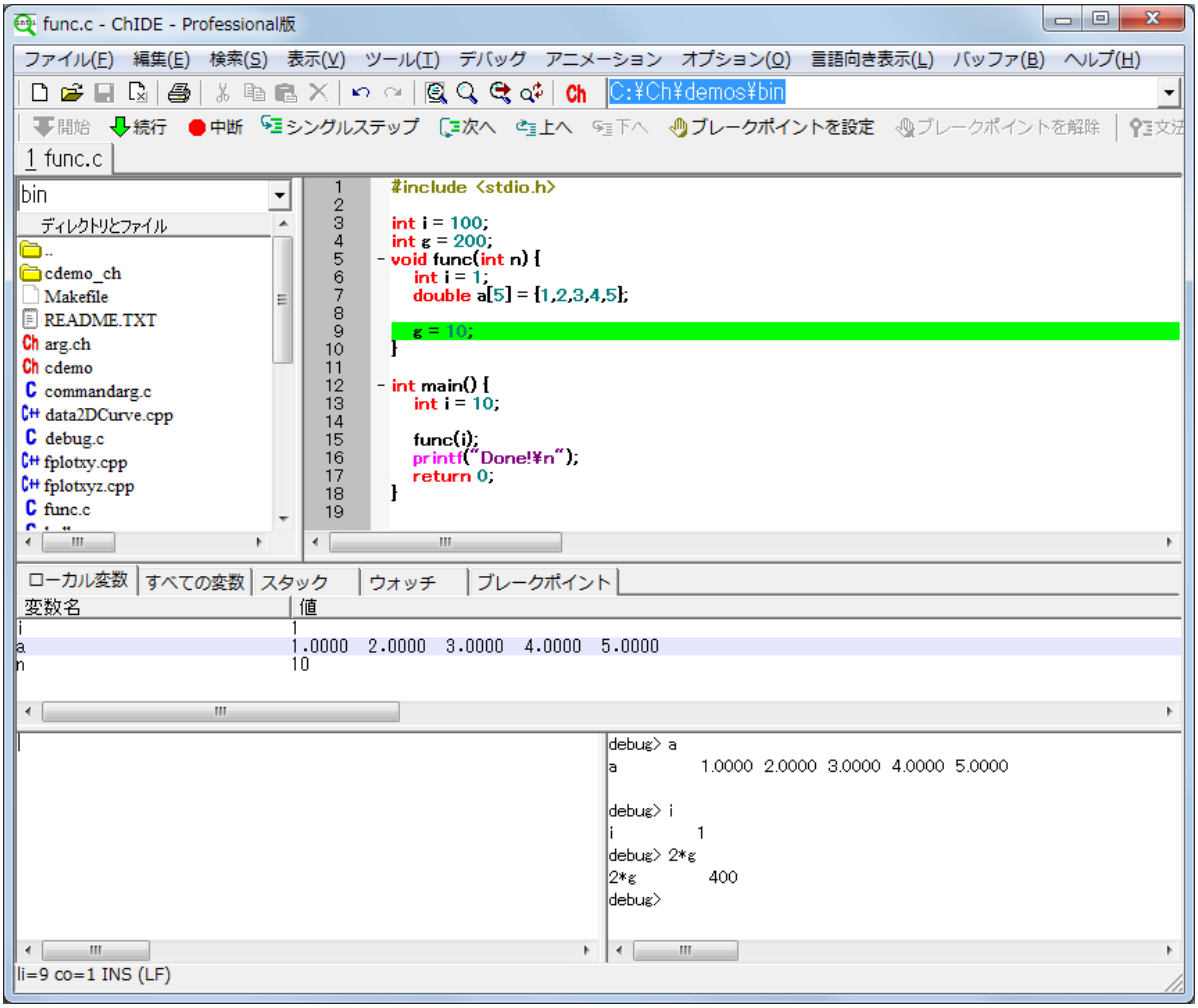
ChIDE および
Ch コマンドシェル入門

Ch バージョン 7.0



2012 年 9 月 日本語版 7.0

目次

1 はじめに

Ch は、組み込み可能なクロスプラットフォームの C/C++インタープリタです。Ch は、C++におけるクラスをサポートしたスーパーセットです。Ch は他のユーザーフレンドリーなハイレベルな拡張機能とともに C99 と呼ばれる最新の C 標準のほとんどの機能をサポートしています。Ch は、2D/3D プロット、数値計算、組み込みスクリプティングおよびクイックアニメーションに使用することができます。高度な数値機能により、Ch は工学および科学技術における様々なアプリケーションに使用することが可能です。しかしながら、Ch は、C/C++を学ぶ学生を教えるクラスのインタラクティブなプレゼンテーションに特に適しています。

統合化開発環境（IDE）は C および C++プログラムの開発に使用することができます。特にこれは、自動書式ハイライト機能を用いてプログラムを編集し、これを IDE 内で実行する目的で使用されます。ChIDE はクロスプラットフォームの IDE であり、コンパイルすることなしに Ch のインタープリタで C/Ch/C++ プログラムを編集、デバッグ、および実行します。ユーザーは、プログラム実行中に、ブレークポイントの設定、プログラムのシングルステップ実行、変数値のウォッチと変更などを行うことができます。ChIDE は Embedded Ch を用いて開発されました。C および C++でコンピュータプログラミングを学習する初心者にとって、これはもっともユーザーフレンドリな IDE です。ChIDE はまた、Windows 上の Microsoft Visual Studio .NET、Linux および Mac OS X 上の GNU gcc/g++などのようなお使いの C/C++コンパイラを用いて編集された C/C++プログラムをコンパイル、リンクに使用することも可能です。

Ch はインタープリタですので、C/C++の式、ステートメント、関数、およびプログラムは、コンパイルせずにただちに実行することができます。それゆえ、Ch は C/C++の教授と学習に対する理想的なソリューションです。インストラクターは学級のプレゼンテーションにおいてノート PC で Ch をインタラクティブに用い、プログラミング授業、特に学生の質問に迅速に答えるような形式の授業に使用できます。学習者はまた、コンパイル、リンク、実行、デバッグの退屈なサイクル抜きに、C/C++の異なる機能をすばやく試すこともできます。初心者の学習支援のため、Ch はエラー発生時に、**セグメンテーションフォールト**や**バスエラー**、またはクラッシュなどの不可解でよく知られていないメッセージの代わりに多くのわかりやすい警告やエラーメッセージを使って開発されています。

この簡潔な ドキュメント により、ユーザーは ChIDE と Ch コマンドシェルを使ってコンピュータプログラミングを学習し、C/C++でプログラム開発を迅速に開始できるようになります。

2 ChIDE 上での C/Ch/C++プログラムの実行

2.1 ChIDE の起動

ChIDE はプラットフォームの違いに関わらず、同じプログラム **chide** の実行により起動できます。Windows 上では、ChIDE は、デスクトップ上で図 1に示されているアイコンをダブルクリックして、簡単に起動することも可能です。

Mac OS X x86 上では、ChIDE は、ダッシュボード上または Application フォルダ内の図 1に示されているアイコンをクリックして起動することも可能です。

図 1: Windows、Mac OS X、および Linux 上の ChIDE アイコン

Linux 上では、ChIDE はスタートアップメニュー内のエントリ Programming Tools から起動することも可能です。コマンド

```
ch -d
```

を使用すると、Ch のアイコンがデスクトップ上に作成されます。Ch が ChIDE とともにインストールされると、ChIDE のアイコンもまたデスクトップ上に作成されます。

コマンド **chide** に対するコマンドラインオプションは以下のようになります。

```
chide [-d directory] [filenames]
```

コマンドラインオプションがない場合、コマンド **chide** は ChIDE が前回終了した際に保存された情報を用いて ChIDE を起動します。オプション `filenames` を指定すると、1 つまたは複数のファイルが開かれます。オプション `-d directory` を指定すると、ChIDE のファイルブラウザウィンドウにより、ディレクトリ `directory` 内のファイルの一覧が表示されます。たとえば、コマンド

```
chide -d /Users/home/harry file1.ch file2.ch
```

は、カレントディレクトリ内のファイル `file1.ch` と `file2.ch` を開き、ファイルブラウザウィンドウでディレクトリ `/Users/home/harry` 内のファイルの一覧が表示します。

2.2　C/Ch/C++プログラムの編集と実行

2.2.1　C/Ch/C++プログラムのソースコードの編集

ChIDE におけるテキスト編集作業は、[メモ帳] に自動書式スタイリング機能を追加したような、ほとんどの Windows または Macintosh エディタでの作業と類似しています。

ChIDE は、セクション 3.8 に記載されているように、最大 20 個のファイルまでをメモリに保持可能です。ChIDE の一般的なレイアウトは、図 2で示されています。, この図はまた、このドキュメント内での ChIDE の説明に使用されている様々な用語も示しています。

1 つの例として、コマンド［ファイル］→［新規作成］またはツールバー上の最初のアイコンをクリックして新規のドキュメントを開き、編集ウィンドウ内に図 3 のように、以下のコードを入力します。

```
/* File: hello.c
   Print 'Hello, world' on the screen */
#include <stdio.h>

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

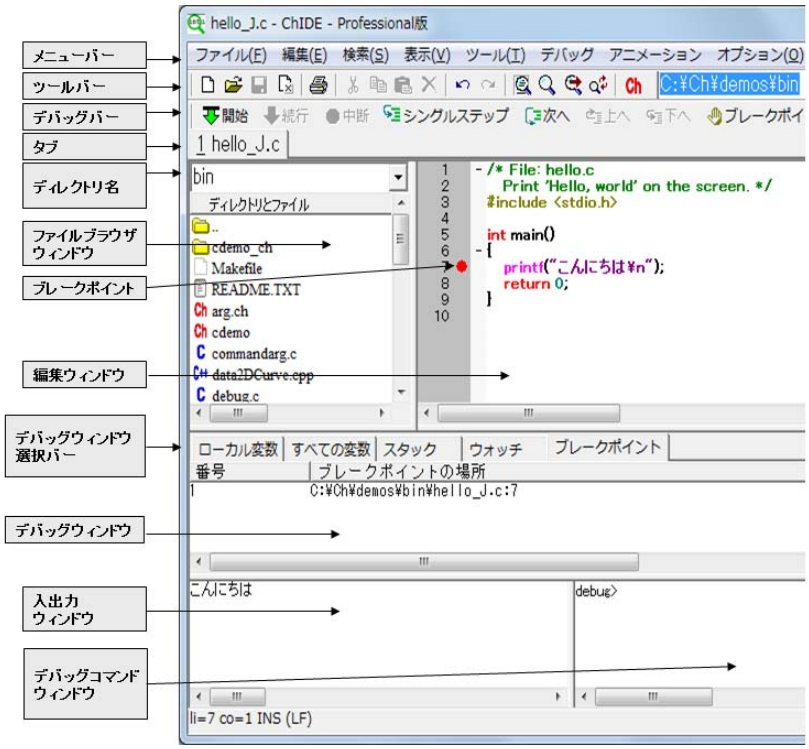


図 2: ChIDE のレイアウトに関連した用語

プログラムは、書式ハイライト機能により、色付きで表示されます。

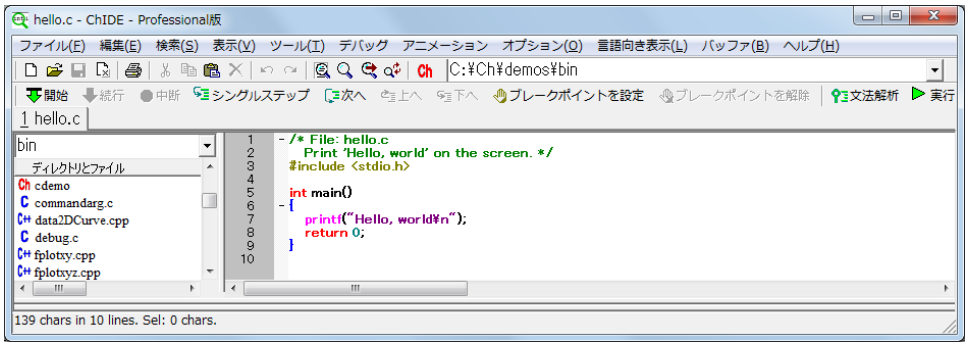


図 3: ChIDE の編集ウィンドウ内で編集されたプログラム

ChIDE の初回起動時に、`CHHOME/demos/chdemos` 内のプログラムが `HOME/chdemos` にコピーされます。ここで、`CHHOME` は Ch のホームディレクトリ（フォルダ）であり、Windows では `C:/Ch`、Mac では `/usr/local/ch` であり、そして `HOME` はユーザーのホームディレクトリであり、たと

えば、C:/Documents and Settings/Administrator のようになります。. Windows 上では、HOME/chdemos/hello.c にある同じプログラム hello.c、たとえば、C:/Documents and Settings/Administrator/chdemos/hello.c もまた［ファイル］→［開く］コマンドを使用してロードすることができます。既定では、このプログラムは ChIDE 起動時にロードされます。Windows 上では、Windows エクスプローラに表示されているプログラムは ChIDE にドラッグアンドドロップすることもでき、これにより、そのプログラムを編集ウィンドウ上で開くことができます。

編集ウィンドウの左側の行番号、余白、および折りたたみ表示用余白は、コマンド［表示］→［行番号］、［余白］、［折りたたみ表示用の余白］をそれぞれクリックすることにより、図 4 のように、オフにすることができます。これらのコマンドを再度クリックすると、それぞれのマークは元に戻ります。折りたたみ表示用余白上の折りたたみ点のマーカー '-' と '+' をクリックして、ブロックコードをそれぞれ折りたたんだり、展開したりすることができます。

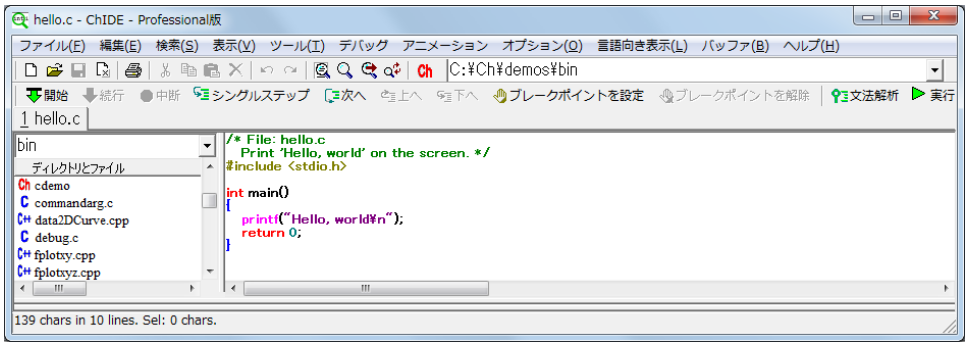


図 4: ChIDE 内に、行番号、余白、折りたたみ表示用余白、ファイルブラウザウィンドウなしに表示されたプログラム

図 4では、ファイルブラウザウィンドウは、コマンド［表示］→［ファイルブラウザウィンドウ］をクリックすることにより、閉じられています。

このドキュメントをコマンド［ファイル］→［名前を付けて保存］により、図 5のように、ファイル名 hello.c として保存します。

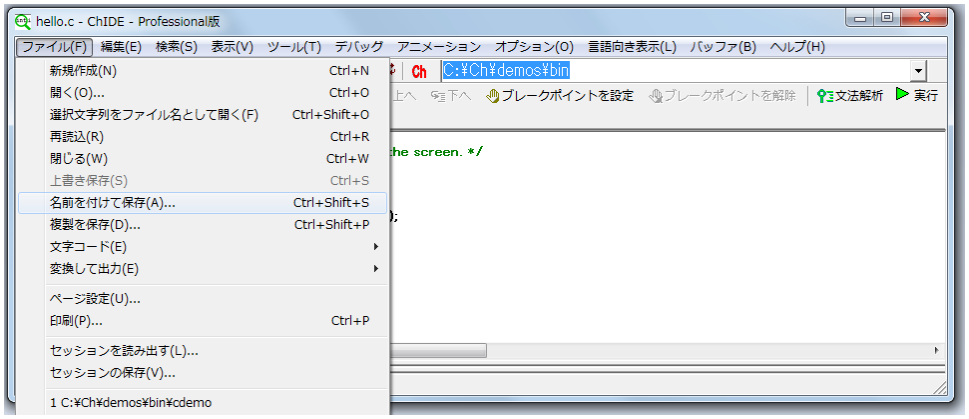


図 5: ChIDE で編集したプログラムをコマンド［ファイル］→［名前を付けて保存］により保存

デバッグバーの下にあるそのファイル名のタブを右クリックして、図 6 のように、コマンド［名前を付けて保存］を選択し、そのプログラムを保存することもできます。

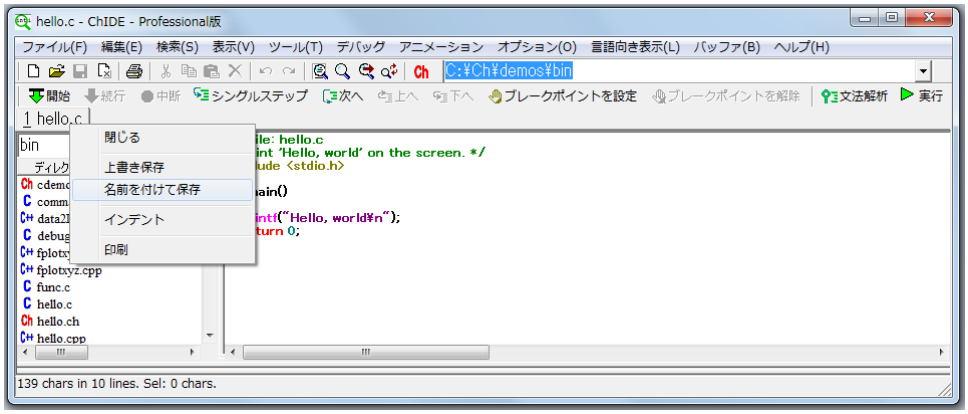


図 6: ファイル名を右クリックして ChIDE で編集したプログラムを保存

ChIDE には、図 2に示されているように、編集ウィンドウ、ファイルブラウザウィンドウ、デバッグウィンドウ、デバッグコマンドウィンドウ、および入出力ウィンドウの 5 つのウィンドウがあります。ファイルブラウザウィンドウは、編集ウィンドウの左側に配置されています。デバッグウィンドウは、編集ウィンドウの下、または次の段落で説明されているように、ChIDE が「左右に分割」モードで表示されている場合は右に配置されています。このウィンドウのサイズは起動時にはゼロになっていますが、編集ウィンドウとの境界をドラッグすることにより、拡大することができます。デバッグコマンドウィンドウはデバッグウィンドウの下または右に配置されています。デバッグおよびデバッグコマンドウィンドウの詳細については、セクション 4. を参照してください。同様に、入出力ウィンドウもデバッグウィンドウの下または右に配置されています。入出力ウィンドウはデバッグコマンドウィンドウの左に配置されています。このウィンドウのサイズは起動時にはゼロになっていますが、デバッグウィンドウとの境界をドラッグすることにより、拡大することができます。既定では、プログラムからの出力は、入出力ウィンドウ内に指定されています。

［表示］ → ［左右に分割］コマンドは、ファイルブラウザウィンドウを左側、そのとなりに編集ウィンドウ、デバッグウィンドウ、さらに入出力ウィンドウとデバッグコマンドウィンドウをその右側に配置する、「左右に分割」モードに ChIDE のレイアウトを変更することに使用されます。現在のセッションの ChIDE の位置とサイズ、ファイルブラウザウィンドウ、編集ウィンドウ、デバッグウインドウ、および入出力ウィンドウは ChIDE 終了時に保存されます。ChIDE の次回の起動時には、前回のセッションのこれらの保存された値は、新規のセッションで使用されます。コマンド［表示］ → ［既定のレイアウト］は、ChIDE のグローバルおよびユーザーオプションファイル内の値を用いて、既定のレイアウトを使用します。

2.2.2　C/Ch/C++プログラムの実行とその停止

ファイル拡張子 .c、.ch、.cpp、.cc、および .cxx をもつか、またはファイル拡張子をもたない C/Ch/C++ プログラムは、ChIDE ですぐに実行可能です。

プログラム hello.c を実行するには、図 7のように、デバッグバー上の［実行］ボタンをクリックするか、または［ツール］ → ［実行］を選択します。

［実行］または［ツール］ → ［実行］コマンドを選択する代わりに、ファンクションキー F2 を押しても、プログラムを実行することが可能です。

キーボードコマンドの詳細については、セクション 3.6を参照してください。

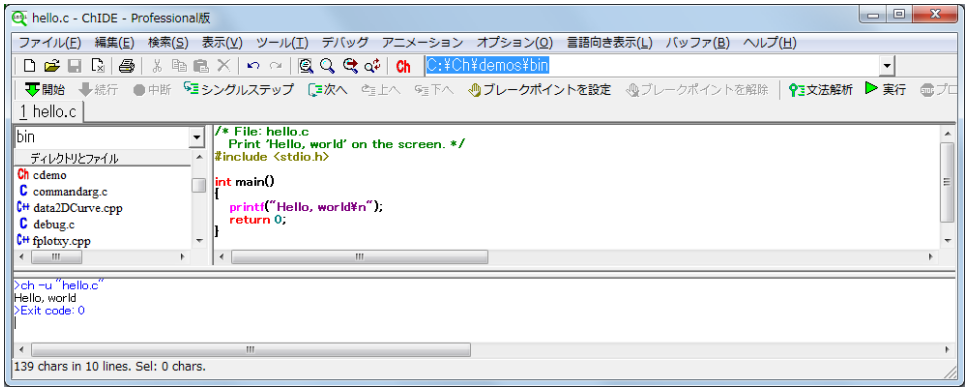


図 7: ChIDE 上の［実行］コマンドを使用したプログラムの実行とその出力

デバッグバー上の［文法解析］または［ツール］ → ［文法解析］を使用して、プログラムを実行せずにその構文エラーのみをチェックすることができます。

［表示］ → ［左右に分割］コマンドは、ChIDE の水平方向のレイアウト変更に用いられます。ここでは、編集ウィンドウを左に、デバッグウィンドウを真ん中に、そして入出力およびデバッグコマンドウィンドウを右側に配置されます。現在のセッションにおける、ChIDE の位置とサイズ、編集ウィンドウ、デバッグウィンドウ、および入出力ウィンドウのサイズは、ChIDE が閉じられる際に保存されます。ChIDE が次回起動される際には、前回のセッションでこれらの保存された値が、新しいセッションに対して使用されます。コマンド［表示］ → ［既定のレイアウト］は、ChIDE のグローバルおよびユーザーオプションファイル内の値を使用し、ChIDE がこの既定値を使うようにリセットします。

コマンドの実行が失敗し、終了までに非常に長い時間がかかる場合、デバッグバー上の［プログラムの強制終了］または［ツール］ → ［実行の中断］コマンドを用いてプログラムを停止することができます。

2.2.3　プログラムの実行による出力

プログラム `hello.c` が実行されると、入出力ウィンドウがまだ表示されていな場合はこれが表示され、以下の実行結果

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

が、図 7のように表示されます。

ChIDE から出力された 1 行目の青字の行

```
>ch -u "hello.c"
```

は、これにより、**Ch** がプログラム `hello.c` を実行したことを示しています。次の黒字の行は、プログラム `hello.c` の実行結果です。ChIDE から出力された最後の青字の行は、プログラムが終了したことを示しています。

終了コードが 0 であることは、プログラム中のステートメント

```
return 0;
```

または

```
exit (0);
```

により、実行が正常に終了したことを意味しています。プログラムの実行中に問題が発生した場合、あるいは

```
return -1;
```

または

```
exit(-1);
```

のように、プログラムが return あるいは exit ステートメントで非ゼロの値で終了した場合、終了コードは-1 になります。

2.2.4　プログラムの構文エラーの検出

ChIDE は、Ch が発生したエラーメッセージを正しく受け取ることができます。このことを確かめるには、プログラムに意図的に誤りを追加します。たとえば、

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

を

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

のように変更します。修正したプログラムに対して［実行］または［ツール］ → ［実行］を選択すると、その結果は以下のようになります。

```
 ERROR: missing ')' before ';'
 ERROR: syntax error before or at line 7 in file 'C:\ch\demos\bin\hello.c'
  ==>:    printf("Hello, world\n";
  BUG:    printf("Hello, world\n"; <== ???
ERROR: cannot execute command 'C:\ch\demos\bin\hello.c'
```

実際の実行画面は、図 8のようになります。

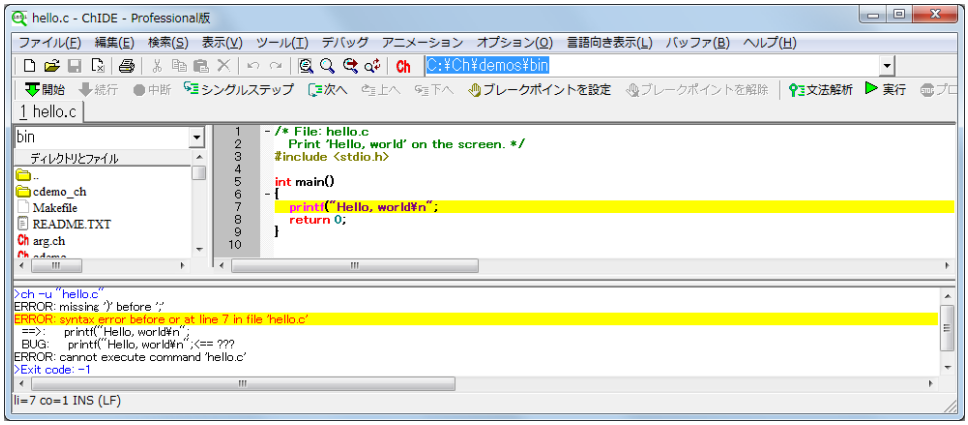


図 8: プログラム hello.c の実行結果の出力におけるエラー行

このプログラムの実行が失敗したため、終了コード-1 が

```
>Exit code: -1
```

のように、入出力ウィンドウの最後に表示されます。
図 8のような入出力ウィンドウ内の赤字のエラーメッセージをマウスの左ボタンでダブルクリックすると、編集ウィンドウ内の不正な書式とエラーメッセージを含んだ行が図 9のように黄色の背景で強調表示されます。

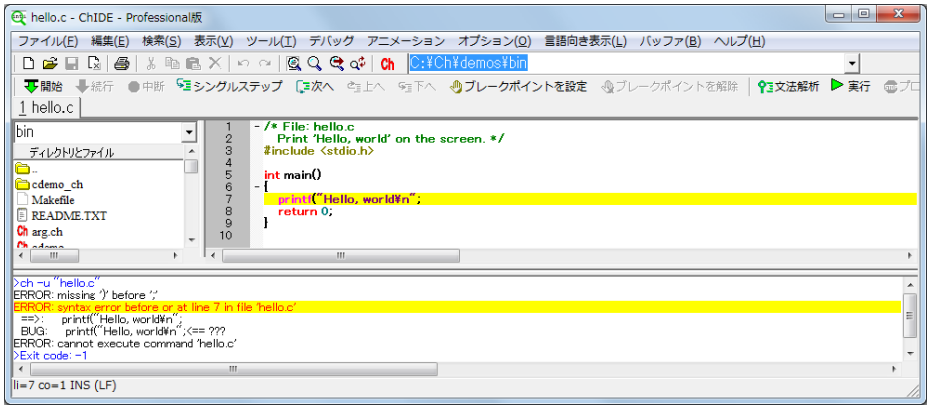


図 9: プログラム hello.c 実行出力からのエラー行の検索

文字カーソルがこの行に移動して、必要な場合には、このウィンドウが自動的にスクロールして、この行が表示されます。

ChIDE は、エラーが発生したファイルが（ヘッダファイルのように）別のファイルの場合、そのファイルを開けるように、エラーメッセージのファイル名と行番号の部分を理解します。

この単純な例では問題がどこにあるかを見つけるのは簡単ですが、大きなファイルの場合、［ツール］ → ［次のエラーメッセージ］コマンドまたはファンクションキー F4 を使用して、レポートされたエラー個所を表示することができます。［ツール］ → ［次のエラーメッセージ］を実行すると、入出力ウィンドウ内の最初のエラーメッセージと、編集ウィンドウ内の該当する行が、黄色い背景で強調表示されます。

コマンド［ツール］ → ［前のエラーメッセージ］、またはファンクションキー Shift+F4 は、直前のエラーメッセージの表示に使用されます。

コマンド［表示］ → ［入出力ウィンドウ］を選択すると、入出力ウィンドウを開く／閉じることができます。既定では、プログラムの実行前には入出力ウィンドウはクリアされます。入出力ウィンドウの内容を維持するには、コマンド［表示］ → ［実行時に入出力ウィンドウを消去］をクリックします。入出力ウィンドウの内容は、図 10のように、コマンド［表示］ → ［入出力ウィンドウを消去］またはファンクションキー F9 によりクリアされます。

2.3　ユーザー入力をともなう C/Ch/C++プログラムの実行

ChIDE は、**scanf()** のような C 関数からのユーザー入力を要求するプログラムも実行できます。たとえば、図　11のようなプログラム　C:/Ch/demos/bin/scanf.c（Windows）または /usr/local/ch/demos/bin/scanf.c（Linux または Mac OS X）をロードします。

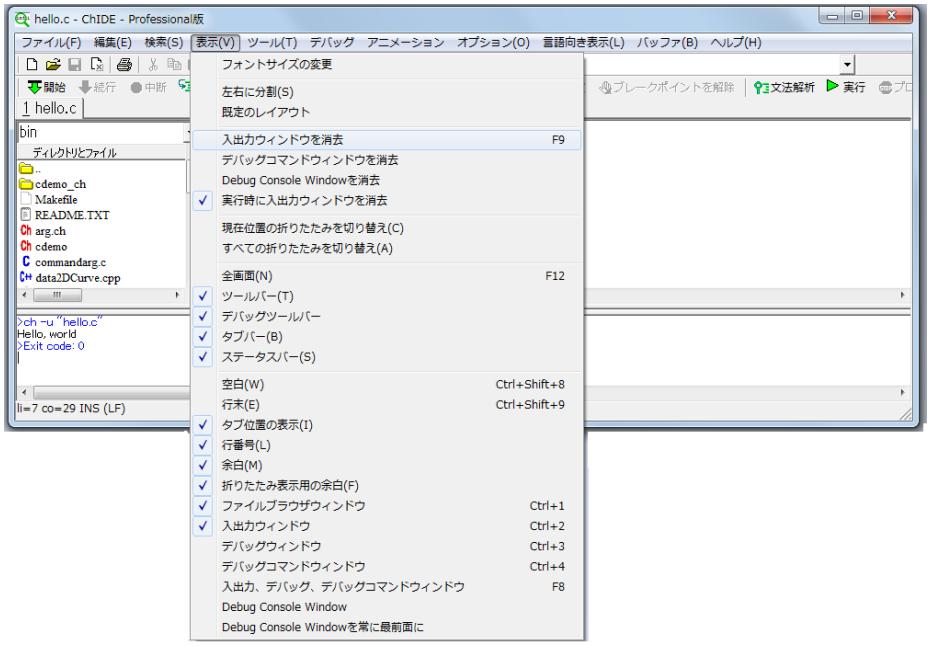


図 10: 入出力ウィンドウの内容の消去

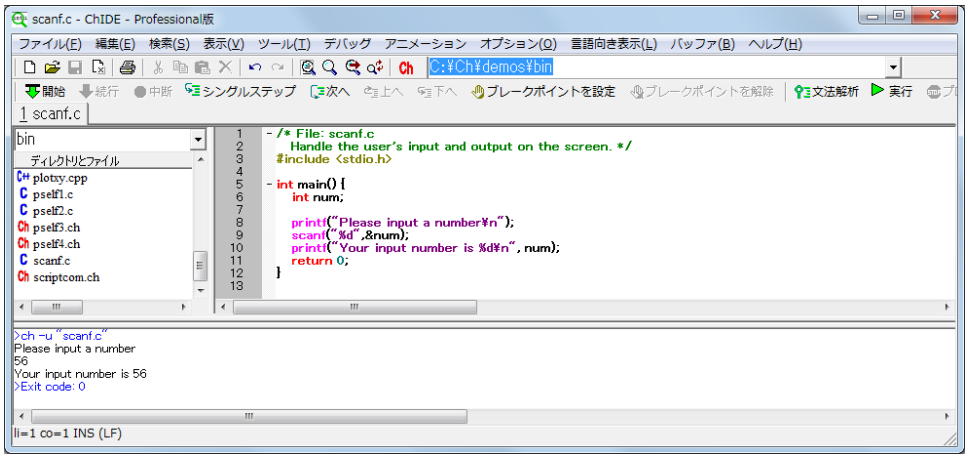


図 11: 入出力をともなうプログラムの実行

このプログラムが実行されると、ユーザーは図 11のように数値の入力を促されます。ユーザーは数値を入出力ウィンドウに入力する必要があります。入力値 56 と出力値は、図 11のようになります。

2.4　プロットをともなう C/Ch/C++プログラムの実行

グラフィックプロットをともなう C/Ch/C++プログラムの実行は、他のプログラムと同様です。このことを 1 つの例によってデモします。

図 12のようなコードを入力します。同じプログラムは `C:/Ch/demos/bin/fplotxy.cpp` にあります。このプログラムが実行されると、図 13のようなプロット結果が出力されます。プロット関数 **fplotxy**() は Ch または SoftIntegration C++ Graphical Library (SIGL) で提供されています。このプログラムは、プロット関数 **fplotxy**() を使用して、0 から 360 の範囲における 37 個の x の値で関数 func() をプロットします。

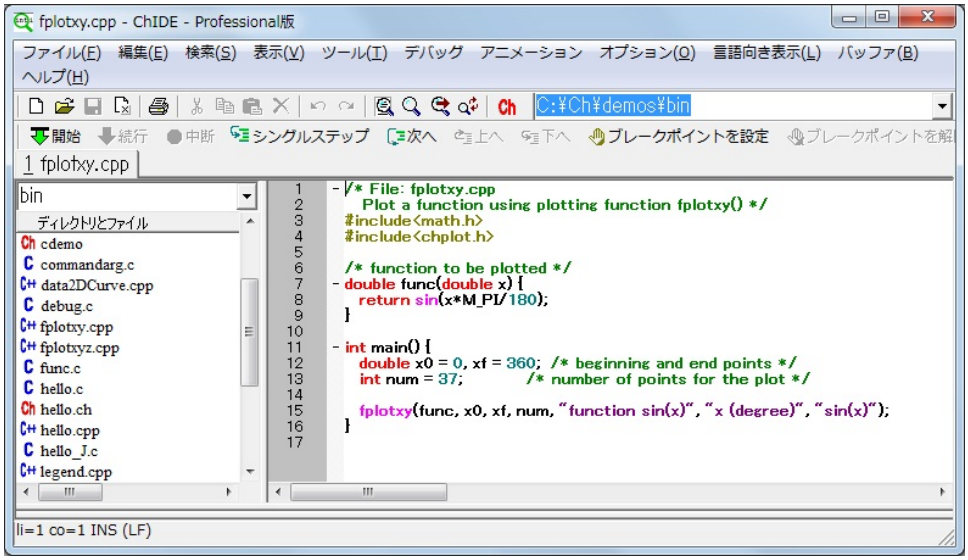


図 12: プロット関数 **fplotxy**() を使用したプログラム

ヘッダファイル **chplot.h** をもつプロット機能を使用したプログラムをコンパイルするには、これを SIGL　C++プロットライブラリとリンクするためにプログラムを拡張子 .cpp をもった C++プログラムとして扱う必要があります。C++プログラムを C++コンパイラを用いてコンパイルする方法は、セクション 8で記述されています。

CHHOME/demos/bin および CHHOME/demos/lib/libch/plot ディレクトリには、Ch のプロット機能の可能性と使用法をデモした多くのサンプルが提供されています。たとえば、プログラム `C:/Ch/demos/bin/plotxy.cpp` は、配列に格納された、プロット関数 **plotxy**() のプロットデータを使用します。これを実行すると、図　13のようなプロット結果が得られます。プログラム `C:/Ch/demos/bin/fplotxyz.cpp`　はプロット関数　fplotxyz()　を使用して、関数 $\cos(x)\sin(y)$ を、－3から3までの範囲の x の値と－4から4までの範囲の y の値に対する独立した2つの値でプロットします。このプロットは、x と y 座標の両方で 80 個の点を使用します。プログラム `C:/Ch/demos/bin/legend.cpp` は、1つのプロットに複数の曲線に対する凡例を追加する方法を示しています。

2.5　コマンドライン引数をともなった C/Ch/C++プログラムの実行

ChIDE は変更可能なコマンドライン引数をもったプログラムを実行することができます。コマンドラインを設定するには、［ツール］ → ［コマンドライン引数］コマンドを使用して現在のコマンドライン引数を表示し、新規の値の設定を許可する、モードレスのコマンドライン引数ダイアログを表示します。メインウィンドウのアクセラレータキーは、異なる引数で 1 つのコマンドを複数回すばやく実行することに使用できるように、このダイアログが表示されている間はアクティブのままになっています。代わりに、セクション 6で記述されているように、入出力ウィンドウ内で実行されるコマンドは、`'*'` で始まるコマンドにより実行される際に、モーダルのコマンドライン引数ダイアログを表示して実行できます。そうでない場合、以下に示すように、これらのコマンドは無視されます。

```
* C:/Ch/demos/bin/commandarg.c
* "C:/Ch/demos/bin/commandarg.c"
```

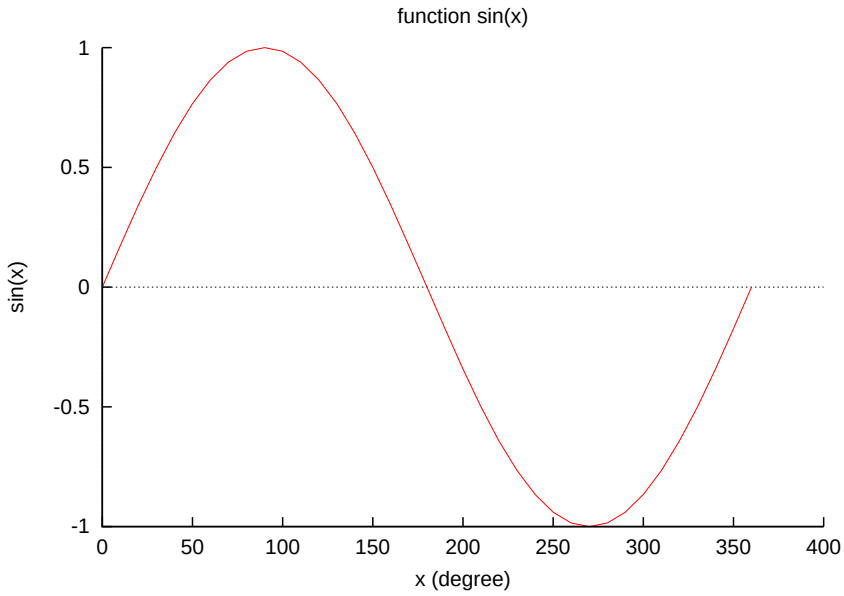


図 13: 図 12のプロットプログラムの出力

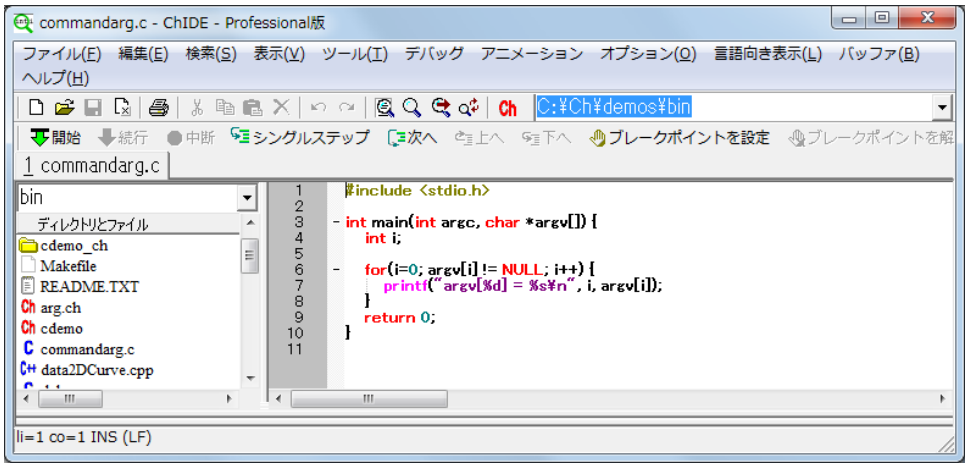


図 14: コマンドライン引数を扱うプログラム

モードレスのコマンドライン引数ダイアログはすでに表示されていて、'*' は無視されます。

図 14 内のプログラムはコマンドライン引数を受け入れ、これを表示します。図 15に示されているコマンド［ツール］→［コマンドライン引数］はモードレスのコマンドライン引数ダイアログを起動します。図 16 は、コマンドライン引数のセットアップ方法を示しています。コマンドライン引数をもつこのプログラムの実行結果は、図 17内に表示されます。

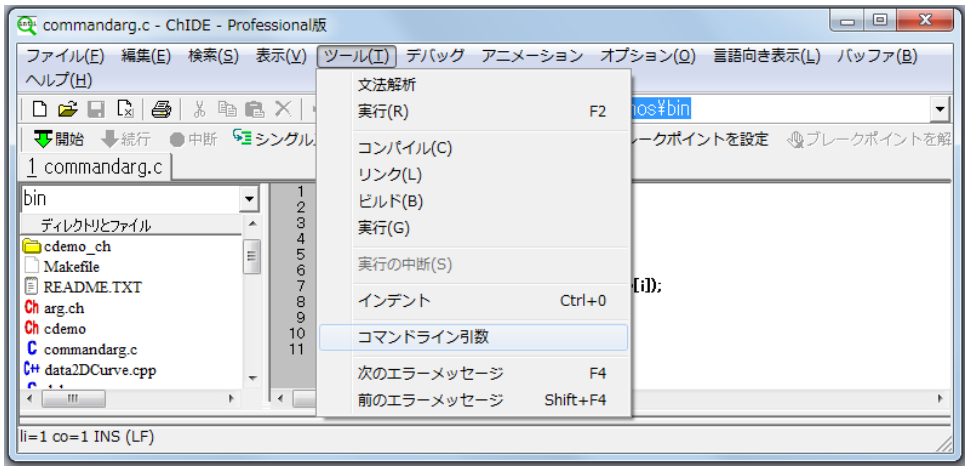


図 15: モーダルのコマンドライン引数ダイアログの起動

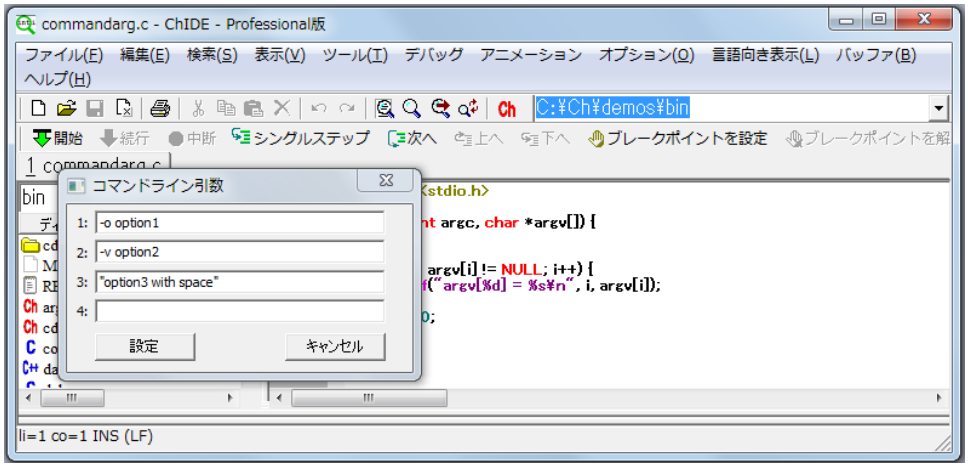


図 16: コマンドライン引数の設定

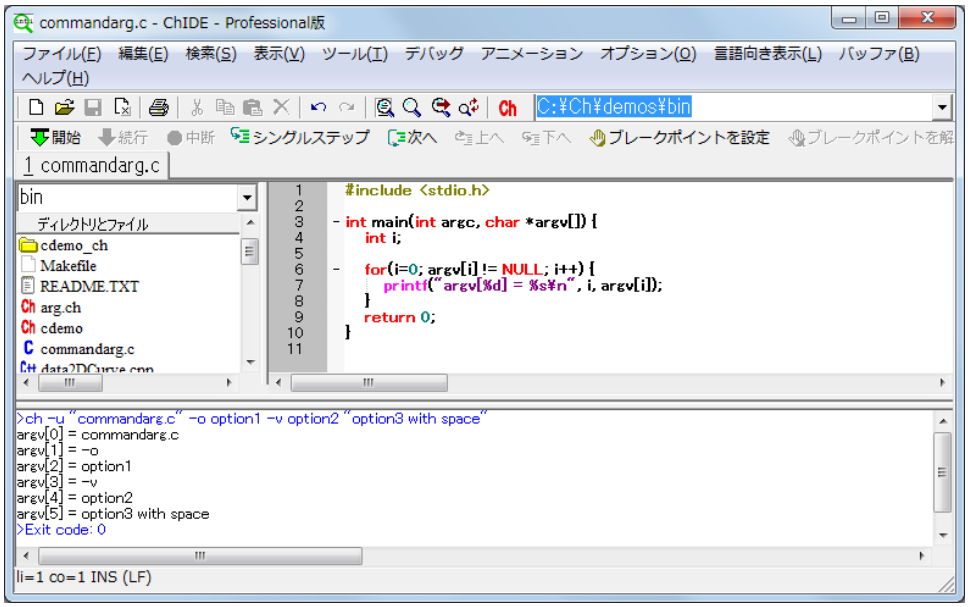


図 17: コマンドライン引数をともなうプログラムの実行

2.6　C/Ch/C++プログラムのインデント設定

読みやすさとソフトウェアのメンテナンスのために、プログラムの各行に適切にインデントが設定されます。多くのネスティングされたループと選択ステートメントをもつプログラムを読みやすくするためには、このことは特に重要です。

メニューバー上のコマンド［ツール］→［インデント］は、編集ウィンドウ内のプログラムに適当にインデントを設定します。デバッグバーの下にあるファイル名のタブを右クリックし、コマンド［インデント］を選択して、プログラムにインデントを設定します。図 6 は、タブバー上のファイル名 `hello.c` が右クリックされたときのコマンド［インデント］を示しています。

3　ChIDE での編集

Microsoft Word またはメモ帳のようなワードプロセッサにおけるほとんどのテキスト編集機能は、ChIDE で使用可能です。ツールバー上のメニューとメニューバー上のコマンド［編集］コマンドの下のメニューは、編集ウィンドウ内でプログラムを編集する際に使用できます。ChIDE における C/Ch/C++ プログラムの編集用のいくつかの独自機能については、このセクションで説明します。

3.1　ファイルの参照

カレントの作業ディレクトリ内のプログラムは、図 2のように、ファイルブラウザウィンドウ内に表示されます。カレントの作業ディレクトリはツールバーの端に表示され、さらにファイルブラウザウィンドウの最上部にも表示されます。

ファイルブラウザウィンドウ内の［ディレクトリとファイル］の下の最初のエントリは、カレントの作業ディレクトリの親ディレクトリです。ファイルブラウザウィンドウ内のカレントの作業ディレクトリの最後の矢印をクリックすると、図 18のように、カレントの作業ディレクトリのすべての親ディレクトリが表示されます。親ディレクトリの 1 つをクリックすると、その内容がファイルブラウザウィンドウ内に表示されます。

カレントの作業ディレクトリの履歴は、図 19で示されているように、ツールバーの最後の矢印をクリックすることにより表示可能です。カレントの作業ディレクトリの履歴にあるディレクトリの 1 つを選択すると、そのディレクトリがカレントの作業ディレクトリになり、その内容がファイルブラウザウィンドウ内に表示されます。

3.2　編集

Windows 上で編集ウィンドウを右クリックすると、共通で使用れている編集コマンドも図 20のように表示されます。

ユーザーが編集ウィンドウ内にテキストを入力すると、その入力された文字列が編集中のファイル内の単語に一致した場合にその一致した単語が表示されます。ユーザーは `Enter` キーを押し、その入力内容を一致した単語に自動的に完成することができます。`Enter` を入力すると、一致したすべての単語を表示させ、矢印キーを用いて単語を選択し、キーを押してその単語入力を完成することができます。

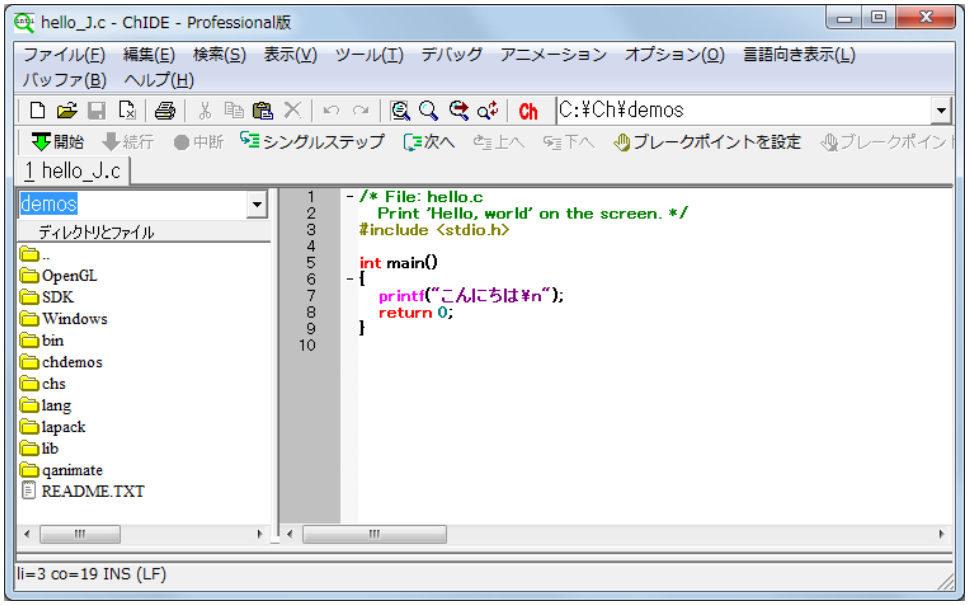


図 18: カレントの作業ディレクトリの親ディレクトリの表示

テキスト上をマウスでドラッグしている間に Windows 上で Alt キーを、または Linux および Mac OS X 上で　Ctrl キーを押し続けることにより、テキストの矩形領域を ChIDE 内で選択することができます。

キーコマンドと省略形を使用することにより、編集作業を迅速化することができます。セクション 3.6の表 Table 2には、編集作業を迅速化するための多くのキーコマンドの一覧が提供されています。省略形はセクション 3.7で説明されています。

3.3　検索と置換

ChIDE では、単語、正規表現、大文字と小文字の区別、逆方向、ワードラップ、文書の最後までなどの単語検索が可能です。C スタイルのバックスラッシュエスケープは、制御文字の検索と置換に使用されます。置換は、現在の選択範囲またはファイル全体にわたり、個別的に行われます。正規表現が使用される際は、タグ形式の副式が置換テキスト内で使用可能です。正規表現は行末を越えて合致しません。便宜上、ファンクションキー F3 が次の単語の検索に使用されます。

3.4　フォントサイズの変更

学級のプレゼンテーション用に、コマンド［表示］ → ［フォントサイズの変更］をクリックし、変更のサイズを設定して、表示されたプログラムのフォントサイズを拡大することができます。さらに、キーボードコマンド `Ctrl+Keypad+`、`Ctrl+Keypad-`、および `Ctrl+Keypad/` をプレゼンテーション中に適切に使用して、フォントサイズを拡大、縮小、通常に戻すなどの設定が可能です。例として、セクション 3.6の表 2をご覧ください。テンプレートキーボードがないノート PC の場合、キーボードコマンドを使用するには、最初に `Shift+NumLk` キーを押して、“Num Lock” をオンにする必要があります。

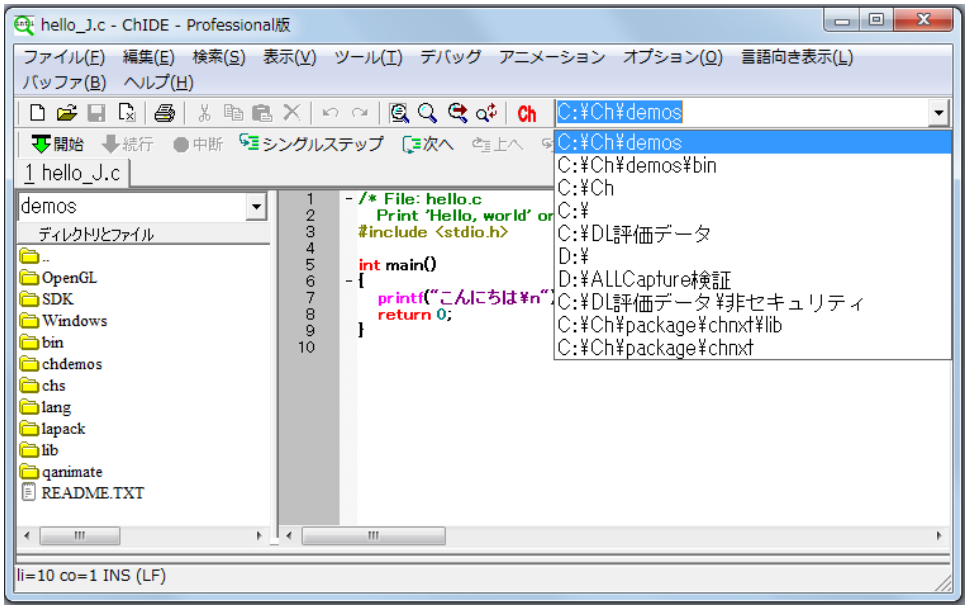


図 19: カレントの作業ディレクトリの履歴の表示

3.5　折りたたみ

ChIDE は、セクション 9で示されているように、C/Ch/C++および他の言語に対する折りたたみ表示をサポートしています。折りたたみポイントは、C/Ch/C++ではインデント設定に、他の言語では括弧のカウントに基づいています。折りたたみポイントマーカーは、セクション 2.2の図 3および図 4に示されているように、折りたたみの展開または折りたたみのためにクリックできるようになっています。折りたたみマージン内のキーボードコマンド `Ctrl+Shift+`クリックは、すべてのトップレベルの折りたたみを展開または折りたたみます。折りたたみポイント上のコマンド `Ctrl+`クリックは、起動すると 1 回ごとにオンとオフが切り替わり、すべての下の階層に対して同じ操作を行います。折りたたみポイント上のコマンド `Shift+Click` はすべての下の階層を表示します。

3.6　キーボードコマンド

ChIDE におけるキーボードコマンドは、ほとんど共通の Windows と GTK+規則にしたがいます。すべての移動キー（矢印、page up/down、home および end）は、Shift キーを押した状態ではストリーム選択、Shit と Alt キーを押した状態では矩形領域選択の拡張または縮小を可能にします。いくつかのキーはある国のキーボードでは使用できません。なぜならば、これらのキーはウィンドウマネージャまたは GTK+のようなシステムにより予約されているからです。メニューコマンドと同等のキーボードはメニュー内にリストされています。

表 1は、もっとも共通に使用されるコマンドとそれに対応するキーボードコマンドの一覧を示しています。

表 2はメニューコマンドにはなく、あまり共通に使用されないコマンドの一覧を示しています。

既定では、Mac OS X のファンクションキー F9、F10、F11、および F12 は、ある機能に前もってバインドされています。表 1のようにこれらのファンクションキーを ChIDE で使用するには、これらのバインドされている機能を以下のステップで無効にします。

- 左上隅の Apple シンボルをクリック。
- [システム環境設定] をクリック。

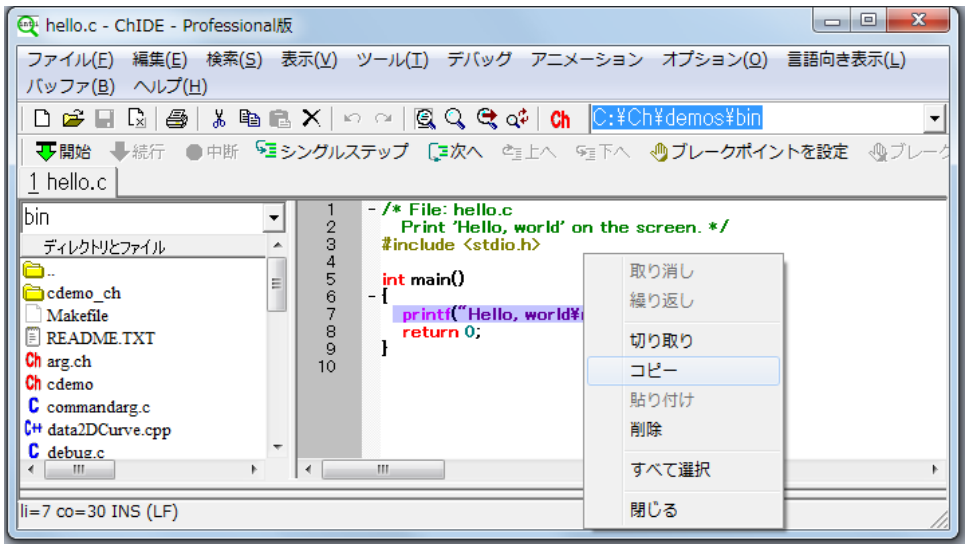


図 20: 編集ウィンドウ上での右クリックによる編集コマンドの使用

表 1: ChIDE で共通に使用されるコマンドとそれに対応するキーボードコマンド

コマンド	**キーボードコマンド**
ヘルプ	F1
Ch で C/Ch/C++プログラムを実行	F2
次を検索	F3
前を検索	Shift+F3
次のエラーメッセージ	F4
前のエラーメッセージ	Shift+F4
開始 (プログラムのデバッグ)	F5
ステップ (シングルステップ)	F6
次へ (次のステートメントをステップオーバー)	F7
入出力、デバッグ、デバッグコマンドウィンドウを開くまたは閉じる	F8
入出力ウィンドウを消去	F9
QuickAnimation ファイルをアニメーション	F10
QuickAnimation に出力を送信	F11
全画面	F12
ファイルブラウザウィンドウを開くまたは閉じる	Ctrl+1
入出力ウィンドウを開くまたは閉じる	Ctrl+2
デバッグウィンドウ開くまたは閉じる	Ctrl+3
デバッグコマンドウィンドウ開くまたは閉じる	Ctrl+3

表 2: ChIDE であまり共通に使用されないコマンドとそれに対応するキーボードコマンド

説明	キーボードコマンド
フォントサイズの拡大	Ctrl+キーパッドの+
フォントサイズの縮小	Ctrl+キーパッドの-
フォントサイズを通常に戻す	Ctrl+キーパッドの/
バッファ上にオープンされているファイルを順に選択	Ctrl+Tab
ブロックのインデント	Tab
ブロックの逆インデント	Shift+Tab
単語の始めまで削除	Ctrl+BackSpace
単語の終わりまで削除	Ctrl+Delete
行の先頭まで削除	Ctrl+Shift+BackSpace
行の終わりまで削除	Ctrl+Shift+Delete
文書の先頭に移動	Ctrl+Home
文書の先頭まで選択	Ctrl+Shift+Home
表示行の先頭に移動	Alt+Home
表示行の先頭まで選択	Alt+Shift+Home
文書の最後に移動	Ctrl+End
文書の最後まで選択	Ctrl+Shift+End
表示行の最後に移動	Alt+End
表示行の最後まで選択	Alt+Shift+End
フォールドポイントの拡張または縮小	Ctrl+キーパッドの*
ブックマークの作成または削除	Ctrl+F2
次のブックマークまでを選択	Alt+F2
スクロールアップ	Ctrl+Up
スクロールダウン	Ctrl+Down
行の切り取り	Ctrl+L
行のコピー	Ctrl+Shift+T
行の削除	Ctrl+Shift+L
前の行と入れ替える	Ctrl+T
行の複製	Ctrl+D
合致する条件付きコンパイルディレクティブを検索（ネストをスキップ）	Ctrl+K
合致する条件付きコンパイルディレクティブまで選択	Ctrl+Shift+K
合致する条件付きコンパイルディレクティブを逆方向に検索（ネストをスキップ）	Ctrl+J
合致する条件付きコンパイルディレクティブまでを逆方向に選択	Ctrl+Shift+J
直前の段落。Shift も押すとその位置まで選択する	Ctrl+[
次の段落。Shift も押すとその位置まで選択する	Ctrl+]
直前の単語。Shift も押すとその位置まで選択する	Ctrl+Left
次の単語。Shift も押すとその位置まで選択する	Ctrl+Right
直前の単語パート。Shift も押すとその位置まで選択する	Ctrl+/
次の単語パート。Shift も押すとその位置まで選択する	Ctrl+\

- [キーボード & マウス] をクリック。
- [キーボードショートカット] をクリック。
- F9、F10、F11、および F12 にバインドされている機能のチェックをはずす。

3.7 省略表現

ChIDE における省略表現により、定義済みのテキストを短い名前に置換し、テキストまたはプログラムを迅速に編集することが可能になります。省略表現を使用するには、これを入力し、[編集] → [省略表現の展開] または Ctrl+B キーを使用して展開された文字列を入力します。省略表現は、省略表現ファイル内でグローバルまたはユーザー指定で定義されている展開文字列と置き換えられます。C/Ch/C++記述用のグローバル省略表現は、次のコマンドによりオープンされます。

[オプション] → [ChIDE グローバル省略表現ファイルを開く]

グローバル省略表現はユーザー省略表現により上書き可能です。ユーザー省略表現ファイルは、次のコマンドによりオープンされます。

[オプション] → [ChIDE ユーザー省略表現ファイルを開く]

省略表現ファイルは、以下の形式のエントリの一覧を含んでいます。

省略表現=展開形

省略表現の名前には、(CR または LF のような制御文字を除き) アクセント文字および中国語のようなアジア言語用の複数バイト文字を含む、任意の文字を含むことができます。省略表現の名前には、プロパティファイルの制限があります。この名前はシャープ (#) または空白またはタブで始まることはできまず (しかし中に空白を含むことはできます)、さらに文字'=' を中に含むことはできません。省略表現の名前は 32 文字以内に制限されていますが、**省略表現**の性格上これで十分です。展開形は'\n' で示される改行文字を含んでいても構いません。展開形内の文字'|' は、展開後にキャレットが存在する位置をマークしています。展開形内で文字表現'|' を入れるには、'||' を使用します。

展開時には、その名前を前のテキストと区切る必要はありません。すなわち、ë を'&eacute' のように定義した場合、これを単語の中で展開することができます。

ある名前が他の名前の終わりである場合、より短い名前のみが展開されます。すなわち、'ring' と'gathering' を定義した場合、後者は展開された'ring' 部分のみを見ます。

グローバルプログラミング省略表現は、プログラムのタイピングとインデント化をスピードアップします。表 3は、C/Ch/C++プログラム用に定義済みのグローバル省略表現の一覧を示しています。

サンプルの省略表現 hw が、配布されている既定のユーザー省略表現ファイル内に含まれています。省略表現 hw を入力し、Ctrl+B を押すと、図 21のような宿題の課題用のヘッダ内容が編集ウィンドウ内に追加されます。ユーザー省略表現ファイルは以下のコマンド

[オプション] → [ChIDE ユーザー省略表現ファイルを開く]

で編集でき、省略表現 hw をユーザー名と関連するクラスまたはプロジェクトの情報とともに構築することができます。

表 3: 既定のグローバル省略表現とその展開形（続き）

省略表現	展開形
com	/* | */
inc	#include < | >
myinc	#include " | "
def	#define |
main	main() 関数
mainarg	引数をもつ main() 関数
if	if ステートメント
elseif	else if ステートメント
else	else ステートメント
for	for ループ
while	while ループ
do	do-while ループ
switch	switch ステートメント
foreach	foreach ループ
a	[|] 配列インデックス
c	' | ' 文字
s	" | " 文字列
p	(|) 括弧
pi	M_PI |
epsilon	FLT_EPSILON |
cond	| ? : 条件演算子
sizeof	sizeof(|)
struct	struct 構造
union	union 構造
enum	enum 構造
class	class 構造
stdlib.h	include stdlib.h
time.h	include time.h
assert.h	include assert.h
complex.h	include complex.h
ctype.h	include ctype.h
errno.h	include errno.h
fenv.h	include fenv.h
float.h	include float.h
inttypes.h	include inttypes.h
iso646.h	include iso646.h
limits.h	include limits.h
locale.h	include locale.h
math.h	include math.h

表 3: (続き)

省略表現	展開形
setjmp.h	include setjmp.h
signal.h	include stdarg.h
stdarg.h	include stdarg.h
stdbool.h	include stdbool.h
stddef.h	include stddef.h
stdint.h	include stdint.h
stdio.h	include stdio.h
stdlib.h	include stdlib.h
string.h	include string.h
tgmath.h	include tgmath.h
time.h	include time.h
wchar.h	include wchar.h
wctype.h	include wctype.h
chdl.h	include chdl.h
chplot.h	include chplot.h
chshell.h	include chshell.h
numeric.h	include numeric.h
func	関数定義
prot	|(); 関数プロトタイプ
call	|(); 関数の呼び出し
printf	printf("|\n");
scanf	scanf("|", &);
sin	sin(|)
標準 *C* 関数名	標準 C 関数を呼び出す

3.8　バッファ

ChIDE には既定で 20 個のバッファがあり、各バッファには 1 個のファイルが含まれています。既定のバッファ数は ChIDE のユーザーオプションファイル内で変更可能です。バッファメニューは、バッファ間の切り替えに使用され、ファイル名または［バッファ］ → ［次］および［バッファ］ → ［前］コマンドのいずれかで選択されます。

キーボードコマンド `Ctrl+Tab` は、セクション 3.6 の表 1で示されているように、バッファ内で開かれているファイルを順に選択します。

すべてのバッファがファイルを含んでいる場合、新たなファイルを開くと、バッファの再利用が必要になり、そこに含まれていたファイルの保存が要求されることがあります。この場合には警告が表示され、ユーザーがそのファイルを保存するかどうか確認します。

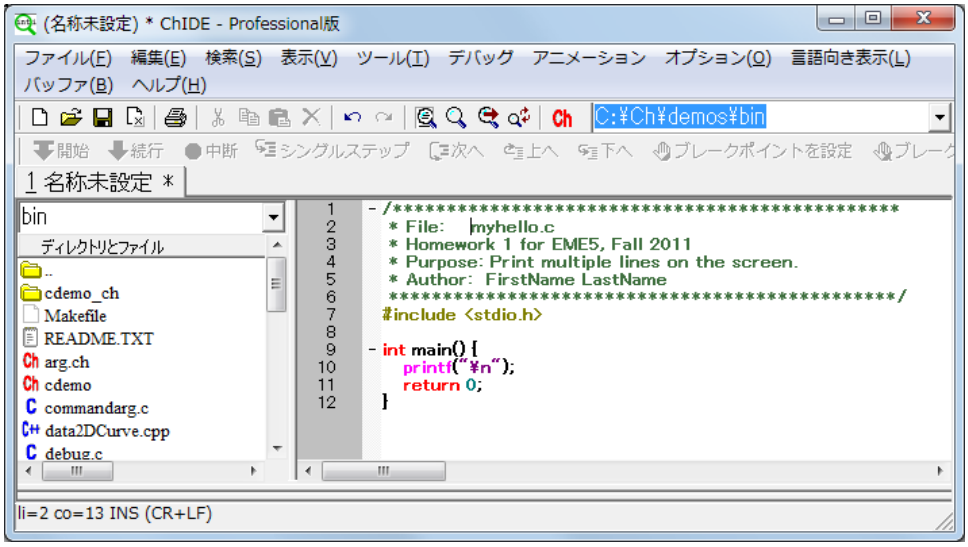


図 21: 省略表現 hw を使用して宿題の課題用ヘッダを作成

3.9　セッション

セッションとは、ファイル名と ChIDE に対するいくつかのオプションの一覧です。現在開いているバッファをセッションとして、将来の迅速なバッチ読み込み用に完全に保存することができます。セッションは拡張子 ".session" をもったプレーンテキストファイルとして保存されます。

セッションを読み込みまたは保存するには、コマンド［ファイル］→［セッションを読み出す］および［ファイル］→［セッションの保存］を使用します。

ChIDE が閉じられると、開かれているバッファがセッション内に保存されます。ChIDE が次回起動されると、直前に保存されたセッションが新たなセッションで自動的に読み出されます。

4　ChIDE における C/Ch/C++プログラムのデバッグ

ChIDE は、バイナリ C プログラム用の典型的なデバッガで利用可能なすべての機能を備えています。「開始」と「シングルステップ」のようなデバッグインターフェースコマンドは、図 22のように、メニューバー上の［デバッグ］から使用可能です。

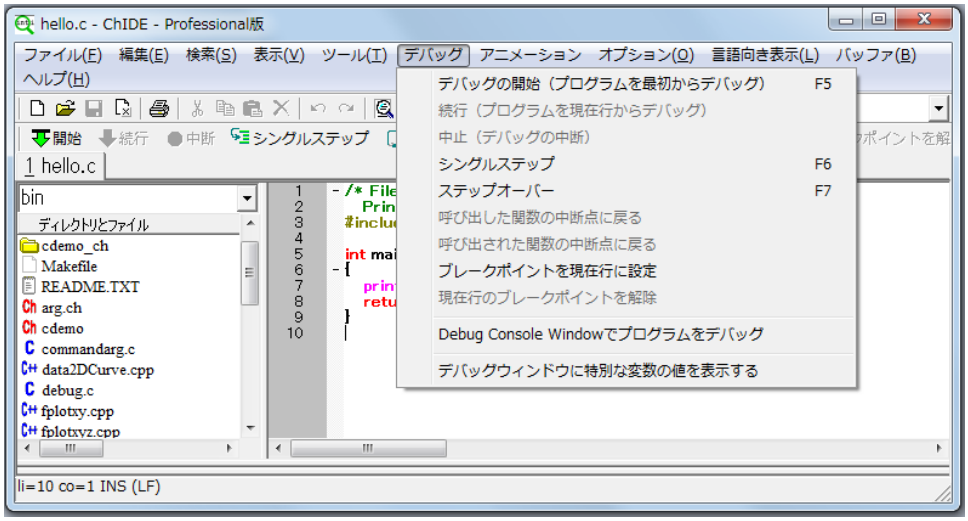


図 22: [デバッグ] メニュー

これらはまた、デバッグバー上で直接に使用可能です。デバッグ中の任意の時点で使用可能なコマンドは、デバッグバー上でクリック可能になっています。クリックできないコマンドはグレーアウトされています。

4.1　デバッグモードでのプログラムの実行

デバッグモードでは、［開始］コマンドまたはファンクションキー F5 により、編集ウィンドウ内のプログラムを実行することができます。プログラムの実行は、ブレークポイントにヒットすると停止します。プログラムを 1 行ずつ実行するには、［シングルステップ］コマンドまたはファンクションキー F6 を使用します。一方、［ステップオーバー］コマンドまたはファンクションキー F7 は関数をステップオーバーして、次の行を実行対象にします。デバッグ中に［続行］コマンドをクリックすると、プログラムは、これが終了するか、あるいはブレークポイントにヒットするまで実行されます。この動作は後述のセクション 4.2で説明されます。

プログラムの実行が失敗し、終了までに時間がかかりすぎる場合は、コマンド［中止］を用いてこのプログラムを停止することができます。しかしながら、プログラムが入力関数 **scanf()** によるような、ユーザーからの入力アクセス中である場合は、この入力関数が終了後にのみ停止させることができます。

4.2　ブレークポイントの設定と消去

プログラムの実行前またはデバッグ中に、新規のブレークポイントを追加して、これにヒットしたときにプログラムの実行を停止させるようにすることができます。ある行に対するブレークポイントは、図 2のように、その行の左余白（[表示] → [余白] をクリックして表示させる）をクリックして設定します。このブレークポイントを解除するには、この行の左マージンにある赤い丸印をクリックします。デバッガのブレークポイントは、図 2のように、デバッグウィンドウの上のデバッグウィンドウ選択バーの［ブレークポイント］タブをクリックして、調べることができます。

デバッグウィンドウは、各ブレークポイントに対するブレークポイント番号とその場所を表示します。現在の行に対するブレークポイントは、デバッグバー上の［ブレークポイントを現在行に設定］コマンドをクリックしても追加可能です。これは、デバッグバー上の［現在行のブレークポイントを解除］コマンドをクリックしても解除することが可能です。

ブレークポイントが何も設定されていない場合、コマンド［現在行のブレークポイントを解除］はクリック不可になっています。しかしながら、ブレークポイントは以下のような初期化をともなう宣言文には設定できません。

```
int i = 10;
```

実行中およびデバッグ中のプログラムを編集すべきではありませんが、あえてそうしようとすると、警告メッセージ

> 警告：デバッグ中にこのファイルに加えられた変更は現在のデバッグセッションには反映されません。

が表示されます。そのプログラムの実行が終了すると、編集可能になります。

コードの追加／削除によりプログラムが編集されると、そのプログラムに対するブレークポイント設定が自動的に更新されます。

後述のセクション 4.5で説明するように、デバッグコマンドウィンドウ内部のデバッグコマンドを使用すると、ブレークポイントを関数に対しておよび、変数の制御用に設定することもできます。

4.3　デバッグウィンドウ内のローカル変数とその値の監視

デバッグバー上またはメニューバー上の［デバッグ］の下の［シングルステップ］コマンドは、ある関数の中へのステップインに使用されます。その関数がすでにバッファ内にロードされているファイルの中にない場合、その関数を含むファイルがロードされます。プログラムの実行の最後に、デバッグ中にロードされたそのファイルがバッファから削除されます。しかしながら、ブレークポイントがそのロードされたファイルに設定されている場合、プログラムの実行が終了しても、そのファイルはバッファ内にとどまります。

［シングルステップ］または［ステップオーバー］コマンドにより、プログラムが 1 行ずつ実行された場合、現在のスタック内の変数名とその値は、デバッグウィンドウ上で、デバッグウィンドウ選択バーの［ローカル変数］メニューをクリックすることにより、調べることができます。プログラムの実行制御が関数内にあるときに、`[locals]` コマンドを入力すると、その関数のローカル変数と引数の値が表示されます。プログラムの実行制御がスクリプトの関数内にないときに `[locals]` コマンドを入力すると、そのプログラムのグローバル変数の値が表示されます。図 23のように、`CHHOME/demos/bin` ディレクトリにあるプログラム func.c が 9 行目を実行対象にし、緑色でハイライトされているとき、ローカルの整数 `i` と `n` は 1 と 10 に、一方 double 型の配列 `a` は、デバッグウィンドウ内で表示されるように、1、2、3、4、5 を含んでいます。

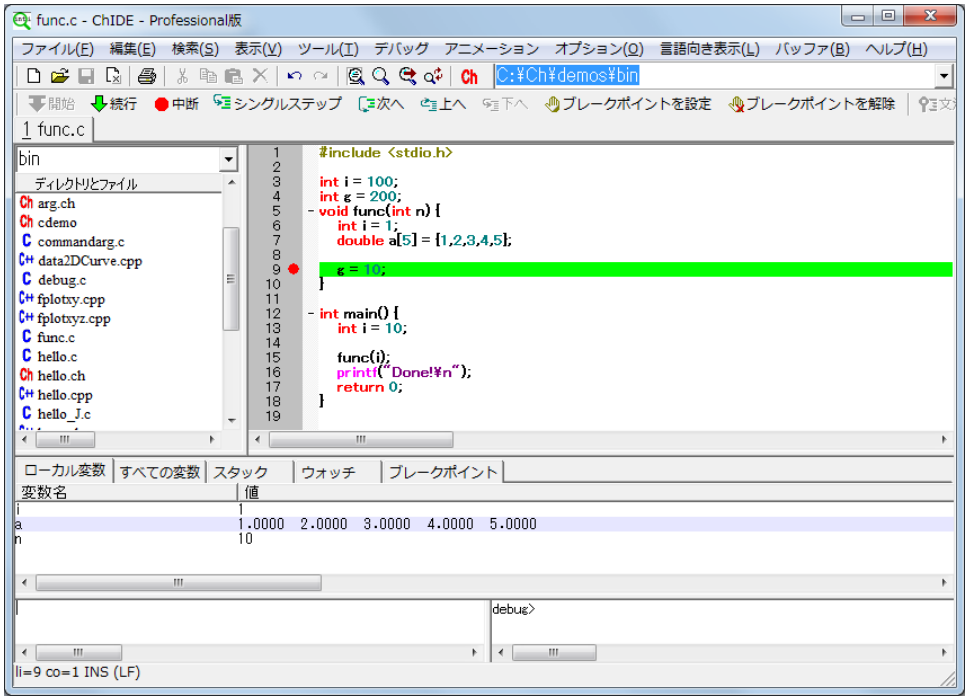


図 23: 現在呼び出されている関数内のローカル変数名と値の表示

4.4　異なるスタック内の変数と、デバッグウィンドウ内のその値の監視

デバッグ中に関数スタックを変更することができます。このスタックは、そのスコープ内の変数がデバッグウィンドウ内で表示され、またはデバッグコマンドウィンドウ内でアクセスされるように、呼び出し関数の［上へ］移動するか、または呼び出された関数の［下へ］移動することができます。現在の行と呼び出し関数の実行行のハイライト用に異なる色が使用されます。たとえば、図 23で［上へ］コマンドをクリックすると、プログラムの制御フローは、図 24で青色にハイライトされているように、呼び出し関数 **main()** の 15 行目に移動します。

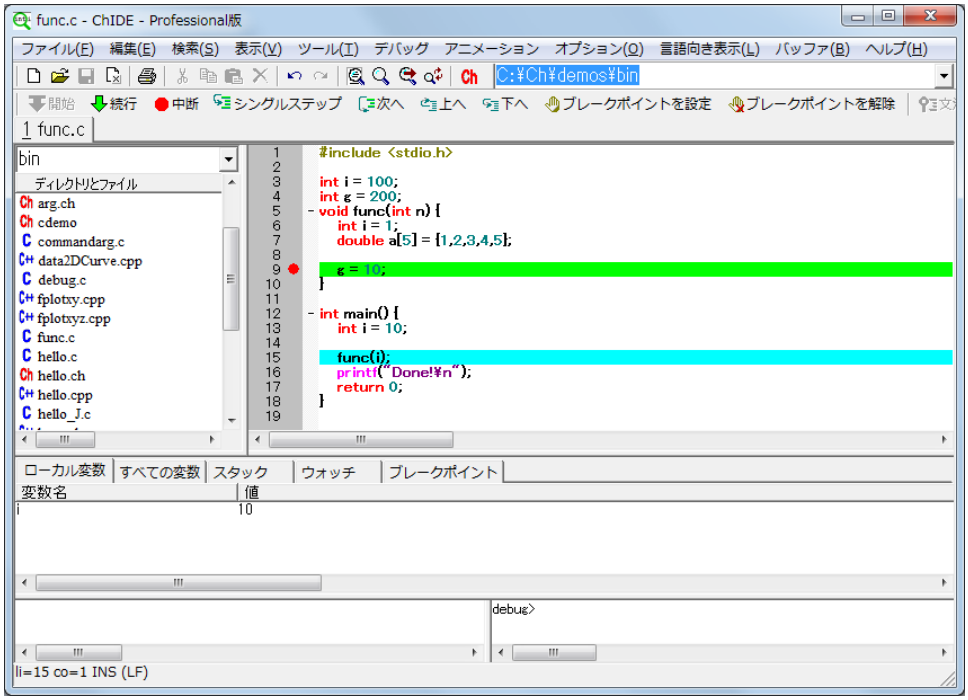


図 24: 呼び出し関数内のローカル変数名と値の表示

図 23では、［下へ］メニューはクリック不可になっていますが、このメニューは、現在のスタックが上に移動させられた際、図 24ではクリック可能になっています。

この時点でのデバッグウィンドウは、唯一の定まった変数である、呼び出し関数 **main()** 内の変数 `i` の名前と値を表示します。

デバッグウィンドウ上部の［スタック］タブは、各スタック内の、関数、メンバ関数、またはプログラム名および対応するスタックレベルを表示します。現在実行中の関数のスタックレベルは 0 であり、一方レベル n+1 は、スタックレベル n の関数から呼び出されている関数です。

たとえば図 25に示されているように、関数 `func()` は、プログラム `func.c` により起動された関数 **main()** により、呼び出されます。

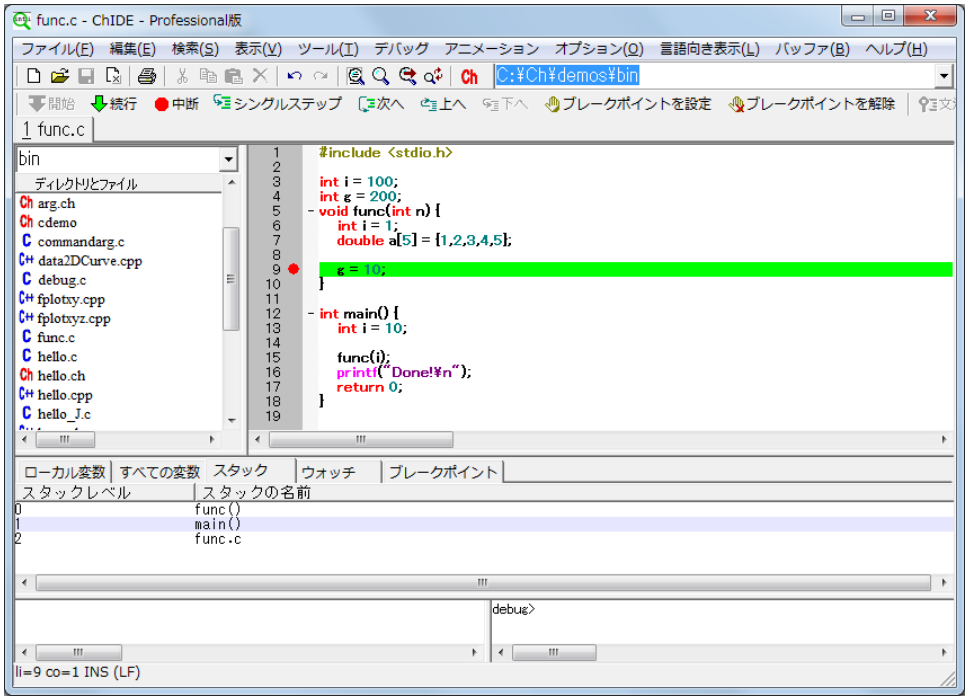


図 25: 実行点に対する異なるスタックの表示

全スタック内の変数名の名前とその値は、デバッグウィンドウ選択バーの［変数］タブで、図 26のように表示されます。

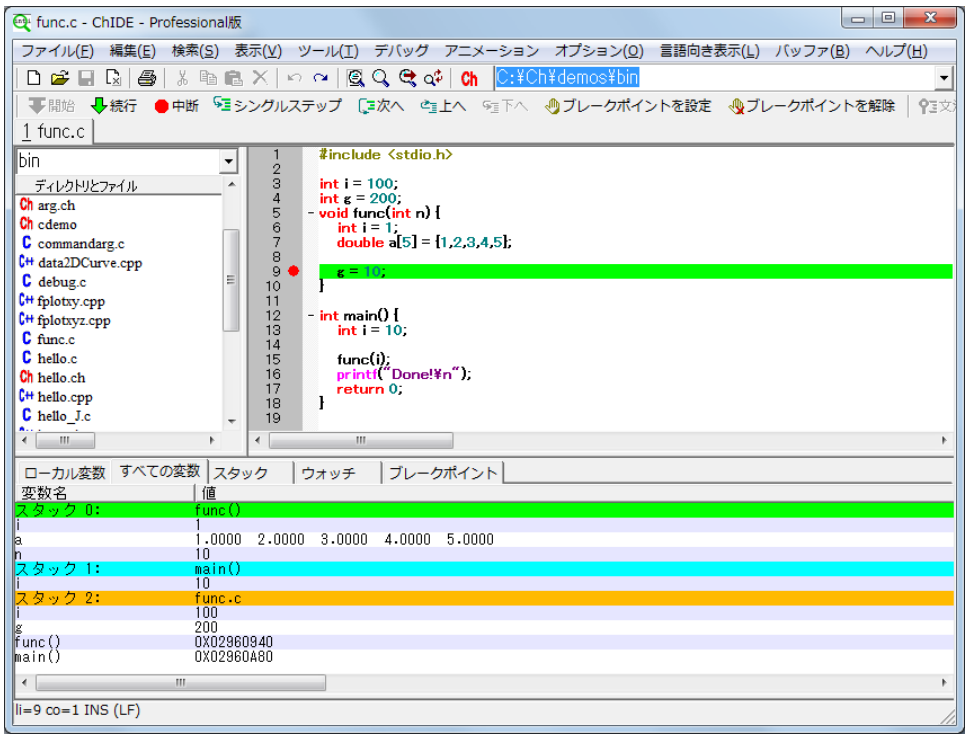


図 26: 全スタック内のすべての変数の名前と値の表示

スタックレベルは、図 24 で示されているように、編集ウィンドウ内の現在の行と呼び出し関数の実行行に対応した色でハイライトされます。図 26 では、プログラムは 9 行目で停止します。

関数 `func()` と **main()** 内のグローバル変数はもちろん、ローカル変数の名前と値もデバッグウィンドウ内に表示されます。おわかりのように、9 行目の実行前のグローバル変数 `g` の値は 200 になっています。

図 22で表示されている、デバッグメニュー内のコマンド

[デバッグウィンドウに特別な変数の値を表示する]

をクリックすると、`__func__`のような特殊な変数の名前と値がデバッグウィンドウの [ローカル変数] と [変数] タブ内に表示されます。

4.5　デバッグコマンドウィンドウ内での [debug] コマンドの使用

プログラムのデバッグ中には、多くのデバッグコマンドがデバッグコマンドウィンドウ内で使用可能です。プロンプト

```
debug>
```

がデバッグコマンドウィンドウ内に表示され、デバッガがデバッグコマンドを受け付け可能になったことを示します。`[help]` コマンドを入力すると、図 27のように、使用可能なすべてのコマンドが表示されます。

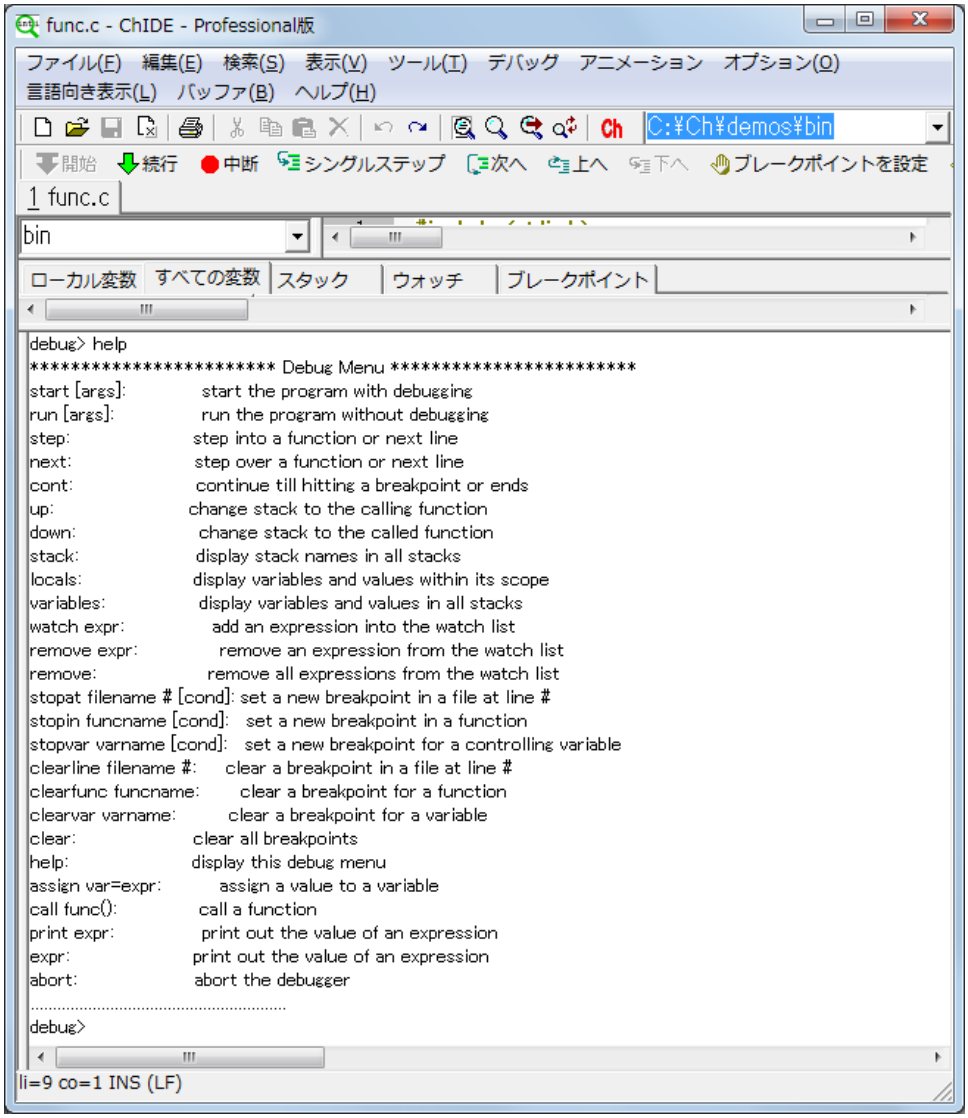


図 27: デバッグコマンドウィンドウ上のデバッグコマンド

コロン（:）の左側のメニューはコマンドを示し、その右側はそのコマンドの動作について説明しています。デバッグバー上の全コマンドには、対応するコマンドがこのデバッグコマンドウィンドウ内にあります。しかしながら、デバッグコマンドウィンドウ内でしか使用できない機能がいくつかあります。

変数、式、および関数は、`[assign]`、`[call]`、および `[print]` コマンドで扱われます。`[assign]` コマンドは値を変数に割り当て、`[call]` は関数を起動し、`[print]` は変数または関数を含む、式の値を出力します。void 型の式を出力することは、void 型を返す関数の場合も含めて、無効な操作です。式をタイプするだけで、その式の値が表示されます。その式が void の型を返す関数である場合、その関数のみが呼び出されます。たとえば、コマンド

```
debug> assign i=2*10
debug> call func()
debug> print i
```

```
20
debug> 2*i
40
debug>
```

は、変数 i に値 2*10 を割り当て、関数 func() を呼び出し、変数 i がそのスコープ内で有効な場合には、式 2*i の値を出力します。他の例として、図 28のように、プログラム func.c が実行され、9 行目で停止した場合、式 2*g とともに変数 a と i の値が、デバッグコマンドウィンドウ上で対応するコマンドを入力することにより、表示されます。

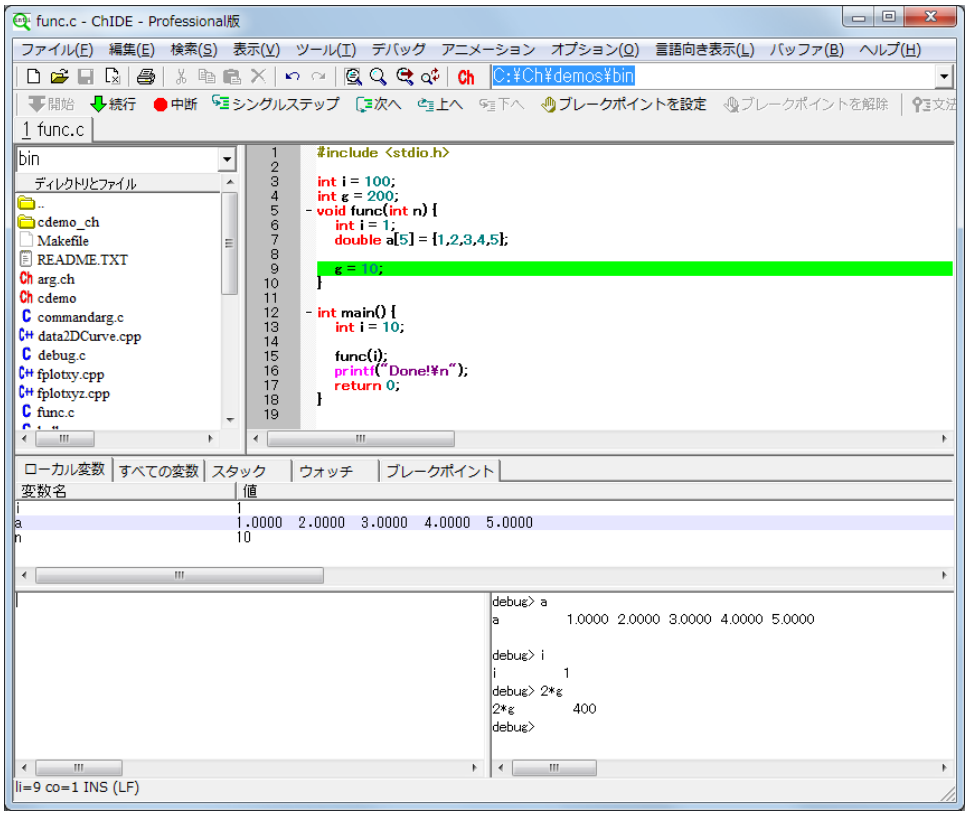


図 28: デバッグコマンドウィンドウ内でのデバッグコマンドの使用

[start] コマンドは、プログラムのデバッグを開始します。[start] と [run] コマンドのオプションのコマンドライン引数は処理されて、**main()** 関数に対する引数に渡されます。たとえば、図 17のプログラム C:\Ch\demos\bin\commandarg.c を実行するため、デバッグコマンド

```
debug> start -o option1 -v option2 "option3 with space"
```

は、文字列 "C:\Ch\demos\bin\commandarg.c"、"-o"、"option1"、"-v"、"option2"、および"option3 with space" を、エレメント argv[0]、argv[1]、argv[2]、argv[3]、argv[4]、および argv[5]、すなわち、Ch スクリプト commandarg.c の main 関数

```
int main(int argc, char *argv[])
```

の引数 argv にそれぞれ割り当てます。

[Debug Console Window] の出力は、図 17の入出力ウィンドウに表示された内容と同様です。空白文字を含んだコマンドライン引数は、上記の文字列 "option3 with space" で示されるように、二重引用符で囲む必要があります。

プログラムはブレークポイントにヒットすると停止します。[run] コマンドを入力すると、プログラムは、ブレークポイントを無視し、デバッグされることなしに実行されます。[step] または [next] コマンドにより、デバッグバー上のコマンドと同様に、プログラムを 1 行ずつ実行することができます。[step] コマンドは、関数にステップインし、一方 [next] コマンドは、関数をステップオーバーして次の行を実行対象にします。デバッグ中に [cont] コマンドを起動して、プログラムをブレークポイントにヒットするか、または最後まで実行させることができます。デバッグ中に関数スタックを変更することが可能です。[up] および [down] コマンドにより、関数スタックのスコープ内の変数にデバッグコマンドウィンドウ内でアクセスできるように、それぞれ関数スタックを、その呼び出し関数の上に移動させるか、または呼び出された関数の下に移動させることができます。すべてのスタック内の関数またはプログラム名を表示させるには、[stack] コマンドを使用します。現在のスタック内の変数の名前とその値を表示させるには、[locals] コマンドを使用します。[variables] コマンドを入力すると、各スタック内のスコープにあるすべての変数の名前と値が表示されます。

[watch] コマンドは、シングル変数を含む式をウォッチ式のリストに追加します。ウォッチ式は、プログラムの実行前にも実行中にも追加することが可能です。[remove expr] コマンドを使用すると、ウォッチ式のリストから式を削除することができます。また、[remove] コマンドは、ウォッチ式リスト内のすべての式を削除します。たとえば、デバッグコマンドウィンドウ内で以下のコマンド

```
debug> watch 2*g
debug> watch i
```

を実行すると、図 29のように、式 2*g と変数 i がウォッチ式のリストに追加されます。 プログラムがブレークポイントで停止するか、または次のステートメントにステップインすると、デバッグウィンドウ選択バーの [ウォッチ] タブをクリックすることにより、図 29のように、これらのウォッチ式の値がデバッグウィンドウ内に表示されます。

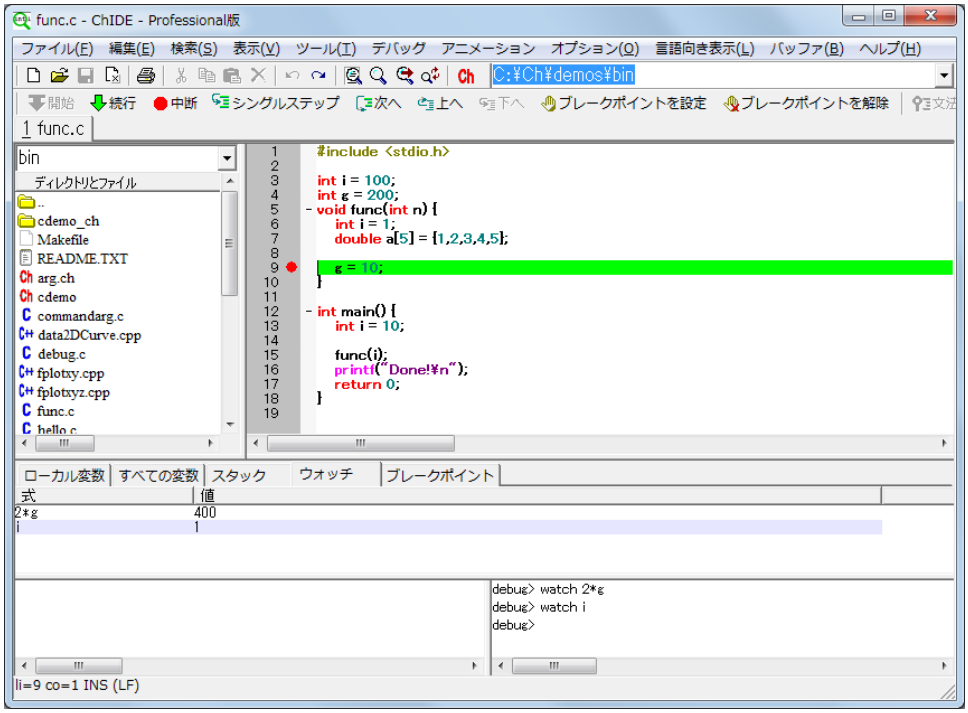


図 29: デバッグコマンドウィンドウ内でウォッチ式と変数を設定し、その値をデバッグウィンドウ内に表示

プログラム実行前またはデバッグ中に、新規のブレークポイントを追加して、プログラムの実行を停止させることができます。ブレークポイントは、次の 3 項目の指定、すなわち、ファイル名と行番号、関数、および制御変数、により設定されます。ブレークポイントが関数内に設定されると、プログラムは関数の最初の実行可能な行で停止します。ブレークポイントが変数に対して設定されると、プログラムは、その変数値が変更されたときに停止します。各ブレークポイントには、オプションの条件式があります。ブレークポイントの場所に到達すると、条件式があればその条件式が評価されます。ブレークポイントは、式（ブレークポイント設定時に指定する必要があります）が真かまたは変更された場合のみ、ヒットします。 既定では、ブレークポイントは、式が真の場合のみヒットします。`[stopat]` コマンドは、引数としてファイル名と行番号を指定することにより、新規のブレークポイントを設定します。プログラムがこの場所に到達すると、実行が停止されます。`[stopin]` コマンドは、関数に対して新規のブレークポイントを設定します。プログラムがこの関数の最初の実行可能行に到達すると、実行が停止されます。`[stopvar]` コマンドは、制御変数に対して新規のブレークポイントを設定します。この変数は、プログラム実行中に評価されます。プログラムは、この変数の値が変更されると、実行を停止します。これらのコマンドのいずれかが起動されると、ブレークポイントがブレークポイントリストに追加されます。各ブレークポイントに対するオプションの条件式とトリガ手段は、これらのコマンドの最後の 2 つの引数として渡されます。たとえば、ブレークポイントを完全なパスと行番号を用いて、ファイル内に設定する場合は、以下のようになります。

```
debug> stopat filename #
debug> stopat filename # condexpr
debug> stopat filename # condexpr condtrue
```

シンボル # を行番号で置き換える必要があります。ブレークポイントの場所に到達すると、オプション式 [condexpr] が評価されます。引数 [condtrue] が真であるか、またはない場合、ブレークポイントは、この式の値が真である場合にヒットします。そうでない場合、ブレークポイントは、この式の値が変更された場合にヒットします。たとえば、コマンド

```
debug> stopat C:/Ch/demos/bin/func.c 6
```

は、C:/Ch/demos/bin ディレクトリにあるファイル func.c 内の 6 行目にブレークポイントを設定します。コマンド

```
debug> stopat C:/Ch/demos/bin/func.c 6 i+j 1
```

も、ファイル func.c の 6 行目にブレークポイントを設定しますが、次のような違いがあります。すなわち、ファイル func.c の 6 行目の位置のブレークポイントに到達すると、式 i+j が評価され、この式の値が真のときに、ブレークポイントがヒットします。上記の式は、以下の式と同等です。

```
debug> stopat C:/Ch/demos/bin/func.c 6 i+j
```

コマンド

```
debug> stopat C:/Ch/demos/bin/func.c 6 i+j 0
```

もやはりファイル func.c の 6 行目にブレークポイントを設定します。この場合、ファイル func.c の 6 行目の位置のブレークポイントに到達すると、式 i+j が評価され、この式の値が変更されたときに、ブレークポイントがヒットします。逆に、適切な引数をともなって、コマンド [clearline]、[clearfunc]、および [clearvar] を実行すると、リスト内のそれぞれ、行、関数、および変数型が削除されます。[clear] コマンドは、デバッガ内のすべてのブレークポイントを削除します。

プログラムの実行が失敗し、終了に時間がかかり過ぎる場合は、[abort] コマンドを用いて、プログラムを停止させることができます。デバッグコマンドウィンドウは、図 10で表示されているように、[表示] → [デバッグコマンドウィンドウを消去] をクリックすることにより、消去できます。

4.6 入出力用に Debug Console Window を使用する

通常、プログラムの標準入出力は入出力ウィンドウで扱われます。関数 hitkb() を使用する Windows コンソールアプリケーションのようないくつかのアプリケーションに対して、ユーザーはデバッグモードにおけるコンソールウィンドウを通してプログラムと相互作用を行うことができます。Debug Console Window は、コマンド [表示] → [Debug Console Window] により、開いたり閉じたりすることができます。プログラムがデバッグモードで実行されると、標準入力、出力、およびエラーストリームは、図 30で示される、切り離された Debug Console Window にリダイレクトされます。

既定では、このコンソールウィンドウは常に他のウィンドウの上に表示されます。この既定の動作は、[表示] → [Debug Console Window を常に最前面に表示] コマンドにより、オン／オフの切り換えを行うことができます。このコンソールウィンドウの内容をクリアするには、図 10のように、[表示] → [Debug Console Window を消去] コマンドを使用します。この「コンソールウィンドウ」のウィンドウおよびフォントサイズはもちろん、背景と文字の色は、このウィンドウの左上隅の ChIDE アイコンを右クリックし、[プロパティ] メニューを選択して変更することができます。なお Windows Vista 上でこのような変更を行う場合、ChIDE を管理者権限で実行する必要があります。

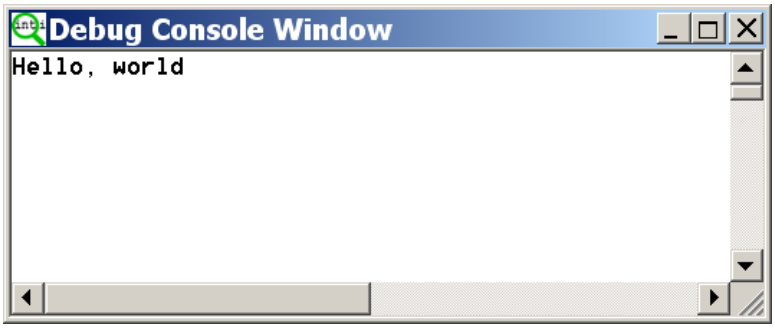


図 30: デバッグ中における入出力用の Debug Console Window

5　Ch コマンドシェル入門

Ch は、コマンドで操作を行うコマンドシェルとして使用することも可能です。MS-DOS シェル、Bash-shell、または C-shell などの他の一般的なシェルのように、Ch シェル内でコマンドが実行可能です。しかしこれらの一般的シェルとは異なり、Ch シェルでは、C および C++における式、ステートメント、関数、プログラムが、容易に実行、参照可能です。したがって、Ch コマンドシェルは、C/C++学習に対する理想的なソリューションです。教官は、Ch をノート PC とともに授業で対話的に用いて、特に学生の質問に答える際に、迅速にプログラミングの要点を説明することができます。学習者は、コンパイル／リンク／実行／デバッグの退屈なサイクルを抜きにして、C/C++の様々な特徴をすばやく学ぶことも可能です。初心者の学習を援助するため、Ch はエラー発生時に、**セグメンテーションフォールト**および**バスエラー**または「クラッシュ」などの不可解でわかりにくいメッセージの代わりに、数多くのわかりやすい警告とエラーメッセージ[1]を発生するようになっています。

Ch シェルは、`ch` コマンドを実行して起動します。Windows および Mac OS X x86 上では、Ch コマンドシェルは、デスクトップまたは ChIDE のツールバー上にある、図 31 のような、赤色の **Ch** アイコンをクリックして簡単に起動することもできます。

図 31: Windows、Linux および Mac OS X デスクトップ上の Ch アイコン

Ch シェルが Windows 上で起動されたあと、既定でユーザーアカウントが管理者であると仮定すると、シェルウィンドウの画面プロンプト

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

が表示されます。ここで、`C:/Documents and Settings/Administrator` は、図 32のようなデスクトップ上のユーザーの**ホームディレクトリ**です。シェルウィンドウのウィンドウとフォントサイズはもちろん、そのテキストと背景の色も、ウィンドウの左上隅の Ch アイコンを右クリックし、[プロパティ] メニューを選択して変更することができます。なお、Windows Vista 上で ChIDE にこのような変更を行うには、管理者権限が必要となります。表示されるディレクトリ `C:/Documents and Settings/Administrator` は、**カレント（現在の）作業ディレクトリ**とも呼ばれます。ユー

[1]ただし現行バージョンでは、これらのメッセージはすべて英語となっています。

ザーアカウントが管理者でない場合、アカウント名 *Administrator* が、適切なユーザーアカウント名に変更される必要があります。

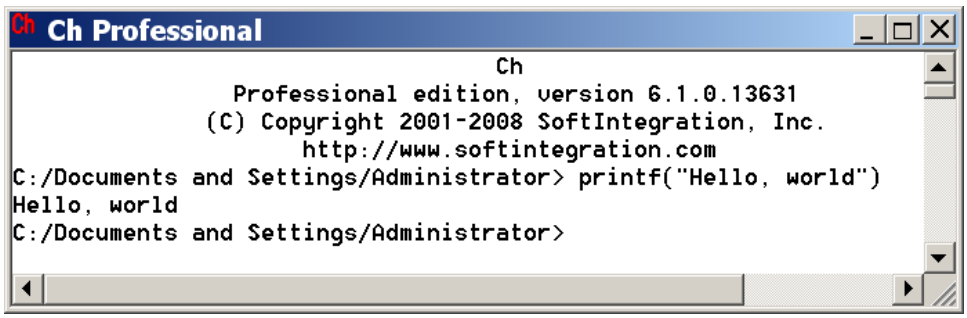


図 32: Ch コマンドシェル

プロンプトは、システムが Ch シェルであり、ユーザーのターミナルキーボード入力の受け入れが可能であることを示しています。Ch シェルの既定のプロンプトは、以下のように構築されています。入力内容が文法的に正しい場合、正常に実行されます。実行終了後、システムプロンプト> が再度表示されます。プログラムまたは式の実行中にエラーが発生した場合、Ch シェルは該当するエラーメッセージを出力し、ユーザーによるプログラムのデバッグを援助します。

C のすべてのステートメントと式は、Ch コマンドシェル内で対話的に実行されます。たとえば、出力 `Hello, world` は、関数 **printf**() を呼び出すことによって以下のように対話的に得られ、

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

図 32のようになります。図 32と比較すると、本書の説明のスペースを節約するために、最後のプロンプト `C:/Documents and Settings/Administrator>`が省略されています。なお、コマンドモード上で C プログラムの該当するステートメントを実行する際には、ステートメントの最後のセミコロンは省略可能です。上の実行例では、関数 **printf**() の呼び出し時のセミコロンはありません。

5.1　ファイルハンドリング用のポータブルコマンド

システムプロンプト>において、C プログラムとステートメントだけでなく、（カレントの作業ディレクトリを表示する **pwd** のような）他の任意のコマンドも実行可能です。このシナリオでは、Ch は Windows 上の MS-DOS シェルと同様に、コマンドシェルとして使用されます。

コマンドは、Ch コマンドシェルまたは Ch プログラム内で実行可能です。数百ものコマンドとそのオンラインドキュメントがシステムに含まれています。これらのすべてを知っている人はいません。各コンピュータウィザードは、常に使用できる作業ツールを備えていて、そこに何があるかという漠然としたアイデアを提供しています。このセクションでは、表 4に掲載されているファイルハンドリング用のもっともありふれたコマンドの使用法について、例を挙げて説明します。

表 4: ファイルハンドリング用のポータブルコマンド

コマンド	用法	説明
cd	cd	ホームディレクトリを変更
	cd *dir*	ディレクトリ *dir* に変更
cp	cp *file1 file2*	*file1* を *file2* にコピー
ls	ls	作業ディレクトリ中の内容の一覧
mkdir	mkdir *dir*	新規のディレクトリ *dir* を作成
pwd	pwd	作業ディレクトリの名前を出力（表示）
rm	rm *file*	*file* を削除
chmod	chmod +x *file*	*file* のモードを変更して、実行可能にする
chide	chide *file.c*	*file.c* の編集と実行のために、ChIDE を起動する

Ch シェル上で実行されるこれらのコマンドは、Windows、Linux または Mac OS X のような異なるプラットフォームに渡ってポータブルであることを再度強調するしておく必要があります。これらのコマンドを使用すると、ユーザーは C プログラムを実行するために、システム上のファイルを効果的に操作することができます。

Ch が Windows 上で、既定のディレクトリ `C:/Ch` にインストールされていると仮定します。カレントの作業ディレクトリは、`C:/Documents and Settings/Administrator` であり、これはユーザーのホームディレクトリでもあります。ファイルハンドリング用ポータブルコマンドのアプリケーションは、以下のように、Ch シェルでのコマンドの対話的実行を例として説明されます。

```
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> chide hello.c
```

表 4の「**用法**」に示されているように、**mkdir** コマンドは、作成するディレクトリとして 1 個の引数を取ります。最初にコマンド

```
mkdir c99
```

を用いて、`c99` というディレクトリを作成します。次に、コマンド

```
cd c99
```

を用いて、この新規に作成したディレクトリ `C:/Documents and Settings/Administrator/c99` に変更します。さらに、コマンド

```
pwd
```

でカレントの作業ディレクトリを表示します。図 3のような、ディレクトリ `C:/Ch/demos/bin` にある C プログラム `hello.c` は、コマンド

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

を用いて、この作業ディレクトリに同じ名前でコピーされます。コマンド

```
ls
```

により、カレントディレクトリにあるファイルの一覧が表示されます。この時点では、ディレクトリ C:/Documents and Settings/Administrator/c99 にあるファイルは、hello.c のみです。すべてのプログラムをあとで容易に探すことができるように、作成した全 C プログラムをこのディレクトリに保存することをお勧めします。

最後に、コマンド

```
chide hello.c
```

により、プログラム hello.c が起動され、図 3に表示されているように、ChIDE で編集と実行が可能になります。学級でのプレゼンテーションに対して、複数のソースファイルを以下のように、1 つのコマンドで開くことは、ときには便利です。

```
> chide file1.c file2.c header.h
```

空白文字を含むパスを扱うコマンドを使用するには、そのパスを二重引用符で囲む必要があります。たとえば、ファイル hello.c を削除するには、以下のように指定します。

```
> rm "C:/Documents and Settings/Administrator/c99/hello.c"
```

5.2 Ch におけるコマンド、ヘッダファイル、および関数ファイルへの検索パス設定

コマンドを実行するために、これをコマンドシェルのプロンプトに入力すると、コマンドシェルはそのコマンドを指定済みのディレクトリ内で検索します。Ch シェルでは、文字列型のシステム変数 **_path** が、検索されるコマンドのディレクトリを含んでいます。各ディレクトリは、文字列 **_path** 内で、セミコロンにより区切られています。Ch コマンドシェルが起動される際、システム変数 **_path** はいくつかの既定の検索パスを含んでいます。たとえば、Windows 上での既定の検索パスは以下のとおりです。

```
C:/Ch/bin;C:/Ch/sbin;C:/Ch/toolkit/bin;C:/Ch/toolkit/sbin;C:/WINDOWS;C:/WINDOWS/SYSTEM32;
```

ユーザーは、スタートアップファイル（これについては、あとで説明されます）内の文字列関数 **stradd()** を使用して、コマンドシェル用の検索パスへの新規のディレクトリを追加することができます。この関数は、文字列型の引数を追加して、これを新規の文字列として返します。たとえば、ディレクトリ C:/Documents and Settings/Administrator/c99 はコマンドの検索パスの中にはありません。カレントの作業ディレクトリが
C:/Documents and Settings/Administrator であるときに、このディレクトリ内のプログラム hello.c を起動しようとすると、Ch シェルはこのプログラムを見つけることができず、以下のようなエラーメッセージを出力します。

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

Ch が起動されるか、または Ch プログラムが実行されると、既定では、ユーザーのホームディレクトリ上にスタートアップファイル**.chrc**（Linux または Mac OS X のような Unix 環境）または**_chrc**（Windows 環境）が存在する場合には、それが実行されます。本書ではこれ以降、Ch が Windows 環境で使用され、スタートアップファイル**_chrc** がユーザーのホームディレクトリ上にあると仮定します。このスタートアップファイルは多くの場合、コマンド、ヘッダファイル、関数ファイルなどの検索パスを設定します。Windows 上では、既定の設定のスタートアップファイル**_chrc** は、Ch のインストール時に、ユーザーのホームディレクトリ上に作成されます。しかしながら、Unix 上では、既定でユーザーのホームディレクトリ上には、スタートアップファイルはありません。システム管理者は、このようなスタートアップファイルをユーザーのホームディレクトリに追加することができます。スタートアップファイルがまだホームディレクトリ上にない場合、以下のように、Ch を `-d` オプション付きで起動し、

```
ch -d
```

サンプルのスタートアップファイルを、ディレクトリ `CHHOME/config/` からユーザーのホームディレクトリにコピーすることができます。ここで、`CHHOME` は文字列`"CHHOME"`ではなくて、Ch がインストールされたシステムファイルパスであることに注意してください。たとえば、既定で Ch が `C:/Ch`（Windows）または`/usr/local/ch`（Unix）にインストールされた場合、Windows 上では、以下の Ch シェルのコマンド

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

はスタートアップファイル**_chrc** をユーザーのホームディレクトリ
`C:/Documents and Settings/Administrator` に作成します。このローカルの Ch 初期化スタートアップファイル`_chrc` は、メニューバー上の以下のコマンド

`[オプション]` → `[Ch のローカル初期化ファイルを開く]`

で開くことができ、図 33のように、ChIDE における検索パスを編集することができます。

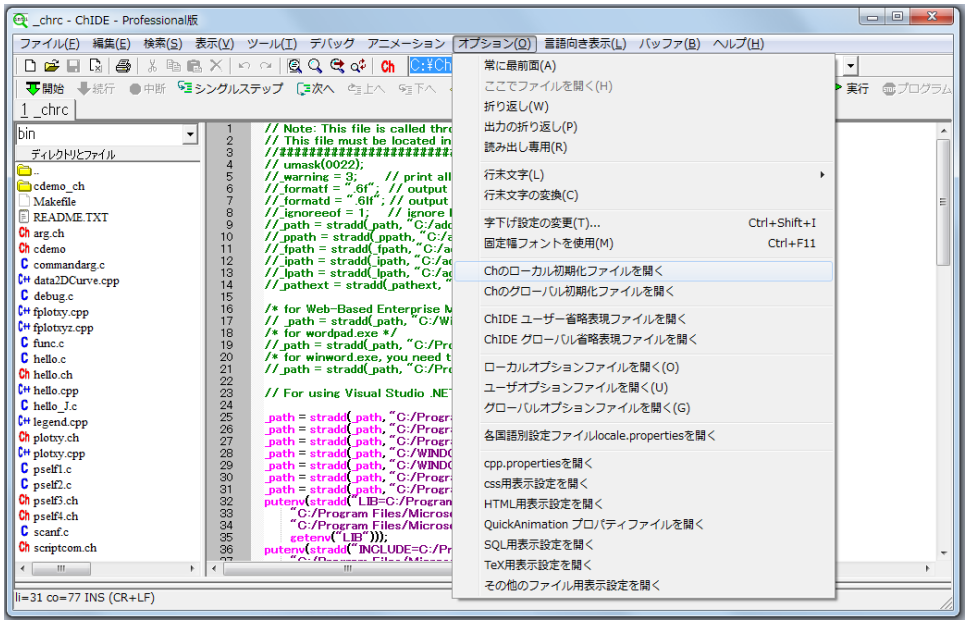


図 33: ローカルの Ch 初期化スタートアップファイルを編集用に開く

Linux では、上記のコマンド `ch -d` は、デスクトップ上に Ch アイコンも作成します。Ch が ChIDE とともにインストールされる場合、ChIDE 用のアイコンもデスクトップ上に作成されます。

ディレクトリ `C:/Documents and Settings/Administrator/c99` をコマンド検索パスに含ませるには、以下のステートメント

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

をユーザーホームディレクトリ上のスタートアップファイル **_chrc** に追加して、このディレクトリ上のコマンド `hello.c` が、カレントの作業ディレクトリの設定に関係なく、起動できるようにする必要があります。ディレクトリ `C:/Documents and Settings/Administrator/c99` が検索パス **_path** に追加されたあとでは、Ch コマンドシェルを再起動する必要があります。このようにして、プログラム `hello.c` をこのディレクトリ上で、以下のように実行することができるようになります。

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

Linux や Mac OS X のような Unix 環境では、コマンドに対する検索パスは既定でカレントの作業ディレクトリを含みません。カレントの作業ディレクトリをコマンドの検索パスに含めるには、以下のステートメント

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

をユーザーホームディレクトリ内のスタートアップファイル **.chrc** に追加する必要があります。以下の関数 `stradd(_path, ".;")` を呼び出すと、'.' で表されるカレントディレクトリがシステム検索パス **_path** に追加されます。

コマンドに対する **_path** と同様に、Ch のヘッダファイルは、システム変数 **_ipath** で指定されるディレクトリ上で検索されます。各パスもまた、セミコロンで区切られます。たとえば、ステートメント

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

は、ディレクトリ C:/Documents and Setting/Administrator/c99 を、

```
#include <headerfile.h>
```

のようなプリプロセッサディレクティブ**#include** によりインクルードされているヘッダファイル用の検索パスに追加します。また、このディレクトリをステートメント

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

により、関数ファイル用の検索パス**_fpath** に追加することもできます。
関数ファイルには、関数定義が含まれています。詳細については、後述のセクション 5.5で説明されています。

5.3 C/Ch/C++プログラムの対話的実行

Ch シェル内で C プログラムを、コンパイルせずに対話的に実行することは非常に簡単です。たとえば、C:/Documents and Settings/Administrator/c99 が、セクション 5.1にあるようなカレントの作業ディレクトリであると仮定します。このディレクトリのプログラム hello.c は Ch で実行可能であり、以下のように、Hello, world の出力を得ます。

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

Ch コマンドシェル上のプログラムの実行による終了コードはシステム変数**_status** 内に保存されます。プログラム hello.c は正常に実行されましたので、コマンドラインで**_status** を入力すると、上記の出力のように、終了コードが 0 になります。
Linux および Mac OS X のような Unix 環境では、C プログラム hello.c をコマンドとしてすぐに使用するために、このファイルを実行可能にする必要があります。コマンド **chmod** はファイルのモードを変更することができます。以下のコマンド

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

はプログラム hello.c を実行可能にし、Ch コマンドシェル内で実行できるようにします。

5.4 C/Ch/C++ 式とステートメントの対話的実行

話を簡単にするため、本書ではこれ以降、Ch コマンドシェル内で表示されるプロンプトは、>のみであるとします。C 式がコマンドシェル内で入力されると、これは Ch で評価され、その結果が画面上に表示されます。たとえば、

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

のように、式 1+3*2 が入力されると、出力が 7 となります。
有効な C 式は、すべて Ch シェル内で評価されます。したがって、Ch は計算機として使用すると便利です。
別の例として、以下のように、変数を宣言してこれをこのあとに続く計算で使用することができます。

```
> 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
```

上記の C ステートメントでは、変数 i が 4 バイトの int 型として、宣言されています。次に、i に対する整数値 30 が、十進、16 進、2 進数で表示されています。異なる数値表記の整数定数も、上記のように、変数 i に割り当てることが可能です。最後に、負数 −2 に対する 2 の補数表記も表示されています。C における他のすべてのデータ型の特性もまた、対話的に表現可能です。

既定では、float または double 型の値は、それぞれ小数点以下 2 桁または 4 桁で表示されます。たとえば、以下のとおりです。

```
> float f = 10
> 2*f
20.00
> double d = 10
> d
10.0000
```

以下のように、すべての C 演算子が対話的に使用可能です。

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

ポインタと変数のアドレスは、以下のように表されます。

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

この例では、int へのポインタの変数 p は、変数 i をポイントします。ポインタへのポインタの作業原理も、同じやり方で対話的に示されます。次の例では、以下のように配列とポインタの関係が示されています。

```
> 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
```

式 a[1]、*(a+1)、*p、および p[0] は、すべて同じ要素を参照しています。多次元配列もまた、対話的に扱うことができます。潜在的なバグ検出のため、Ch は配列の境界をチェックします。たとえば、以下のとおりです。

```
> 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
```

5 要素の配列 a で許されるインデックスは、0 から 4 までです。配列 s はヌル文字を含む 5 文字のみを保持することができます。Ch はこのように、既存の C コード内の、配列境界オーバーランに関係したバグを検知することができます。

C の構造体または C++のクラスのアラインメントも、以下のようにチェック可能です。

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

この例では、int と double のサイズはそれぞれ 4 および 8 であるにも関わらず、int と double 型の 2 つのフィールドをもつ構造体 s のサイズは、アラインメントを適正化するために、12 ではなくて 16 になっています。

5.5 C/Ch/C++ 関数の対話的実行

プログラムを多くの個別のファイルに分割することができます。各ファイルは、プログラムの任意の場所からアクセス可能な、多くの関連した関数群から成ります。C 標準ライブラリのすべての関数が対話的に実行可能であり、ユーザー定義関数内で使用可能です。たとえば、以下のとおりです。

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

乱数発生関数 **rand()** は、**srand(time(NULL)** における時間値を用いて起動の準備がされています。型が特定されない数学関数 **sin()** を呼び出している関数 `add()` がプロンプト上に定義され、使用されています。

複数の関数定義を含んでいるファイルは、通常それ自身を Ch プログラムの一部として識別するために、`.ch` のサフィックスをもっています。Ch プログラミング環境内に関数ファイルを作成することができます。Ch における**関数ファイル** とは、1 つの関数定義のみを含んだファイルのことです。関数ファイル名は、`addition.chf` のように、`.chf` で終わります。関数ファイル内の関数ファイル名と関数定義は同一でなければなりません。関数ファイルを用いて定義された関数は、あたかも Ch におけるシステム内蔵関数として扱われます。

コマンドに対する **_path** と同様に、関数は、関数ファイル用のシステム変数 **_fpath** における検索パスに基づいて検索されます。各パスは、セミコロンで区切られています。既定では、変数 **_fpath** はパス `lib/libc`、`lib/libch`、`lib/libopt`、および `libch/numeric` を Ch のホームディレクトリ内に含んでいます。システム変数 **_fpath** が Ch シェル内で対話的に変更されると、これは、現在のシェル内で対話的に起動された関数に対してのみ、影響を及ぼします。動作中のスクリプトに対しては、現在のシェル内の関数検索パスの設定は、サブシェル内では使用されず、そして継承されません。この場合、システム変数 **_fpath** は、ユーザーのホームディレクトリにあるスタートアップファイル **_chrc**（Windows）または **.chrc**（Unix）内で修正することができます。

たとえば、`addition.chf` という名前のファイルがプログラム 1 のようなプログラムを含んでいる場合、関数 `addition()` は、入力された 2 つの引数 a と b の和 $a+b$ を計算するために呼び出されるシステム内蔵関数として扱われます。

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

プログラム 1: 関数ファイル `addition.chf`

関数ファイル `addition.chf` が
`C:/Documents and Settings/Administrator/c99/addition.chf` にあり、ディレクトリ
`C:/Documents and Settings/Administrator/c99` が、ステートメント

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

をもった、ユーザーのホームディレクトリにあるスタートアップファイル**.chrc**（Unix）または**_fpath**（Windows）内の関数検索パスに追加される必要があると仮定します。関数 `addition()` は、

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

のように、コマンドモードで対話的に使用されるか、またはプログラム内部で使用されるかのどちらかです。プログラム 2において、関数 `addition()` は、関数ファイル `addition.chf` の内部で定義されたプロトタイプ関数が起動されるように、**main()** 関数内で、関数プロトタイプなしに呼び出されます。

```
/* File: program.c
   Program uses function addition() in function file addition.chf */
#include <stdio.h>

/* This function prototype is optional when function addition() in
   file addition.chf is used in Ch */
int addition(int a, int b);

int main() {
    int a = 3, b = 4, sum;

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

プログラム 2: 関数ファイル `addition.chf` を使用したプログラム

関数ファイルの検索パスが適切に設定されていない場合、関数 `addition()` が呼び出された際に、警告メッセージ

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

が表示されます。

関数が Ch シェル内で対話的に呼び出されると、その関数ファイルがロードされます。関数が呼び出されたあとにその関数ファイルを変更すると、コマンドモードにおけるそれ以降の呼び出しは、依然として、古いバージョンのロード済みの関数定義を使用します。修正したバージョンの新たな関数ファイルを起動するには、コマンド **remvar** に続いて関数名を指定して、システム内の関数定義を削除するか、またはプロンプト上で **ch** を入力して、新たな Ch シェルを起動するかのいずれかを行います。たとえば、コマンド

```
    > remvar addition
```

は、関数 `addition()` に対する定義を削除します。コマンド `remvar` は、宣言された変数の削除にも使用されます。

5.6 C++ 機能の対話的実行

Ch では、C プログラムが実行可能であるだけでなく、クラスといくつかの C++機能も、C++コードの対話的実行例として以下に示されているように、Ch でサポートされています。

```
> 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
```

入力と出力は、C++における **cin** および **cout** を用いてハンドリングされています。public メソッド `tagc::set()` は private メンバ `m_i` をセットし、一方、public メソッド `tagc::get()` はその値を取得しています。メソッド `tagc::get()` の引数は、参照渡しされています。クラス `tagc` のサイズは、メンバ関数用のメモリを除いて、4 バイトになっています。

6　入出力ウィンドウ内でのコマンドの対話的実行

バイナリコマンドまたは C/C++プログラムは、図 34に示されるように、入出力ウィンドウ内部でも対話的に実行することができます。

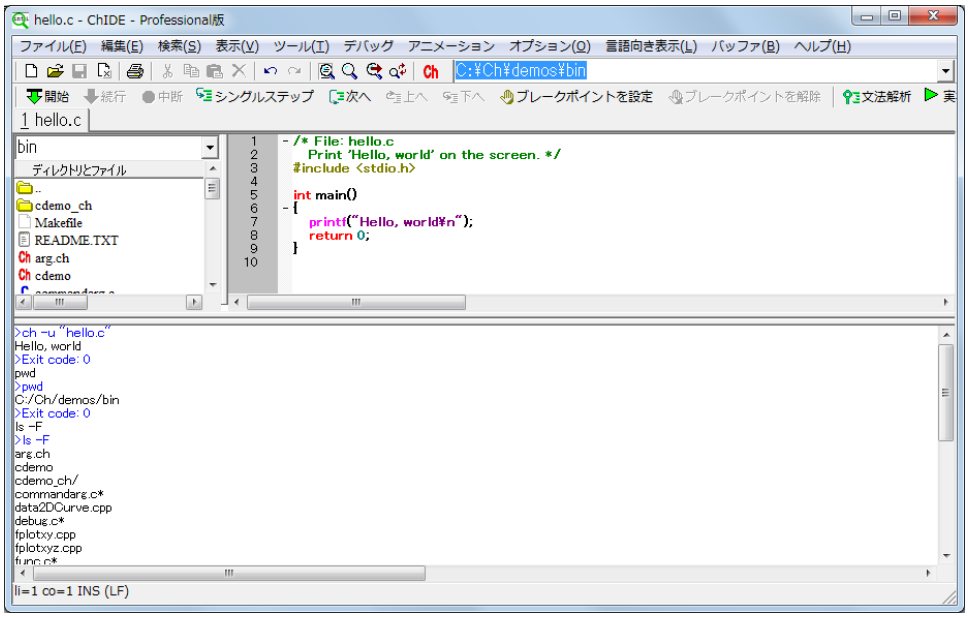


図 34: 入出力ウィンドウ内でのコマンドの実行

図 34において、プログラム `hello.c` が最初に入出力ウィンドウ内で実行され、次にコマンド **pwd** がカレントの作業ディレクトリを表示します。コマンド **ls** は、カレントの作業ディレクトリ内のファイルとディレクトリの一覧を表示します。

コマンドのオプションも使用可能です。たとえば、コマンド **ls** は

```
ls -F
```

の書式で実行可能であり、ディレクトリを最後にスラッシュ付きで一覧表示します。
空白文字を含む完全なパスをともなうコマンドを使用するには、以下のように、そのパスを二重引用符で囲む必要があります。

```
> "C:/Documents and Settings/Administrator/c99/hello.c"
```

C/Ch/C++プログラムをコマンドライン引数付きで実行する方法については、セクション 2.5で説明されています。

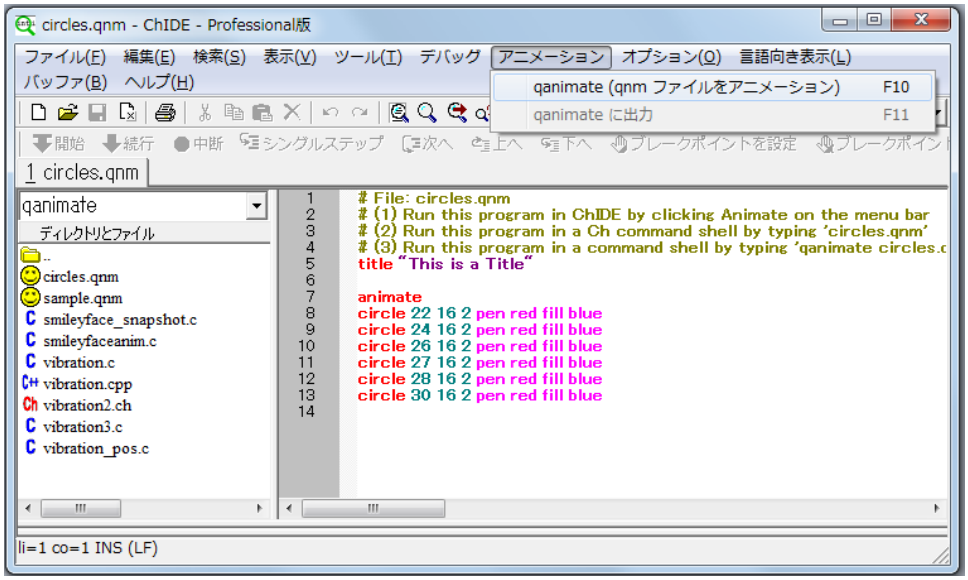


図 35: QuickAnimation ファイル *circles.qnm* の実行

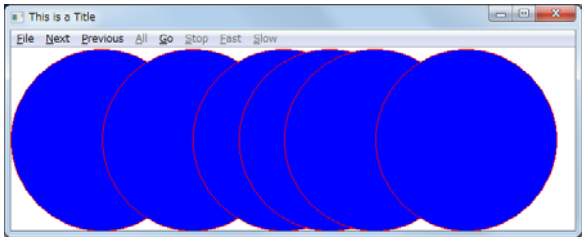


図 36: 図 35の QuickAnimation ファイル *circles.qnm* の実行による出力

7 Quick Animation

Ch は、QuickAnimation™ を用いてクイックアニメーション用に使用することができます。Ch はまた、指定されている x-y 座標データに基づいた様々なオブジェクトの表示に使用することもできます。QuickAnimation™ は、二次元の力学系のアニメーション用途に特に適しています。QuickAnimation™ についての詳細情報は、`CHHOME/docs/qanimate.pdf` にある *QuickAnimation* **ユーザーズガイド**で参照可能です。

拡張子`.qnm` をもつ QuickAnimation ファイルは、図 35 で QuickAnimation ファイル `circles.qnm` に対して示されているように、ChIDE において書式ハイライト機能を用いて編集することができます。

`circles.qnm` に対するアニメーションは、図 35に示されているように、`[アニメーション] → [qanimate (qnm ファイルをアニメーション)]` コマンドまたはファンクションキー F10 をクリックすることにより、生成されます。図 36は、このアニメーションの全フレームを示しています。

printf() のような C 標準関数を用いて、QuickAnimation フォーマットで標準出力を生成する Ch プログラムを記述することができます。図 37でプログラム `CHHOME/demos/qanimate/smileyfaceanim.c` の実行に対して示されているように、この標準出力は、`[アニメーション] → [qanimate に出力]` コマンドまたはファンクションキー F11 をクリックすることにより、QuickAnimation プログラム **qanimate** に直接送ることができます。図 38は、生成されたアニメーションの画面コピーを示しています。

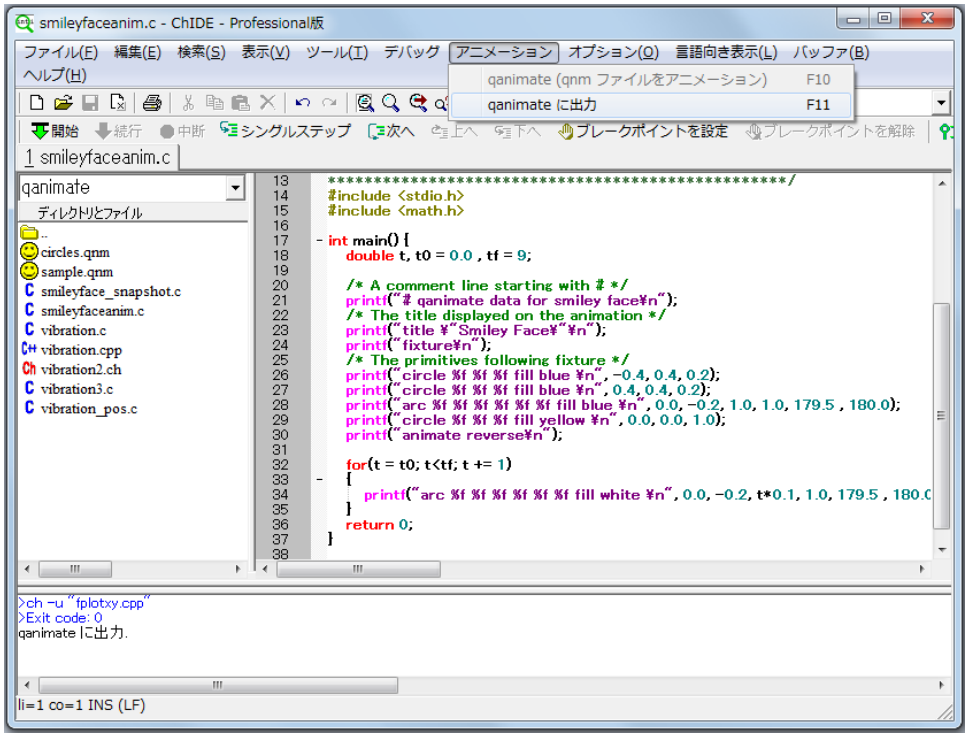


図 37: QuickAnimation に送られる標準出力 (stdout) をもったプログラムの実行

図 38: 図 37のプログラム *smileyfaceanim.c* 実行による出力

8　ChIDE における C/C++プログラムのコンパイルとリンク

ChIDE は、編集ウィンドウ内の編集済みの C/C++プログラムを、C および C++コンパイラを用いてコンパイル、リンクし、生成されたバイナリ実行プログラムを実行することもできます。

既定では、ChIDE は、C および C++プログラムのコンパイル用に Windows 上にインストールされている最新の Microsoft Visual Studio .NET を使用するように、インストール時に構成されます。Visual Studio コンパイラ用の環境変数とコマンドは、ユーザのホームディレクトリにあり、図 33のメニューで開いて編集できる、別個のスタートアップ構築ファイル **_chrc** 内で修正可能です。Linux および Mac OS X x86 上では、ChIDE は、C および C++プログラムのコンパイルに、それぞれ GNU gcc および g++コンパイラを使用します。既定のコンパイラは、［オプション］ → ［cpp.properties を開く］コマンドで開くことができる C/Ch/C++プロパティファイル `cpp.properties` を編集することにより変更可能です。

図 39で表示されている、［ツール］ → ［コンパイル］コマンドは、プログラムのコンパイルに使用することができます。

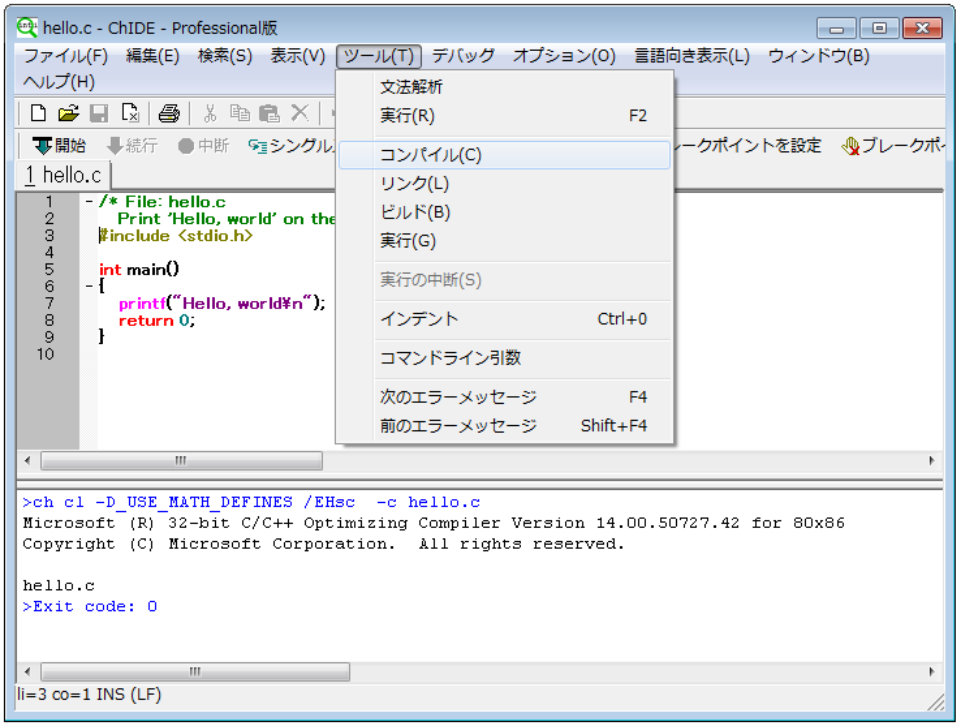


図 39: C/C++プログラムのコンパイル

C または C++プログラムのコンパイル時に発生したエラーメッセージは、ChIDE の入出力ウィンドウ内に表示されます。Windows 上では、プログラムをコンパイルすると、拡張子.obj をもったオブジェクトファイルが生成されます。このオブジェクトファイルは、［ツール］ → ［リンク］コマンドを用いてリンクされ、実行可能なプログラムが作成されます。Windows 上で実行可能なプログラムは、拡張子.exe をもちます。

カレントディレクトリ上で make ファイル makefile または Makefile が使用可能であれば、［ツール］ → ［ビルド］コマンドを実行することにより、その make ファイル起動されて、アプリケーションがビルドされます。

make ファイルは、ファイルタブ上のファイル名を右クリックし、図 40 のように、コマンド［make］（Linux または Mac の場合）、またはコマンド［make］あるいは［nmake］（Windows の場合）をクリックすることによっても起動することができます。

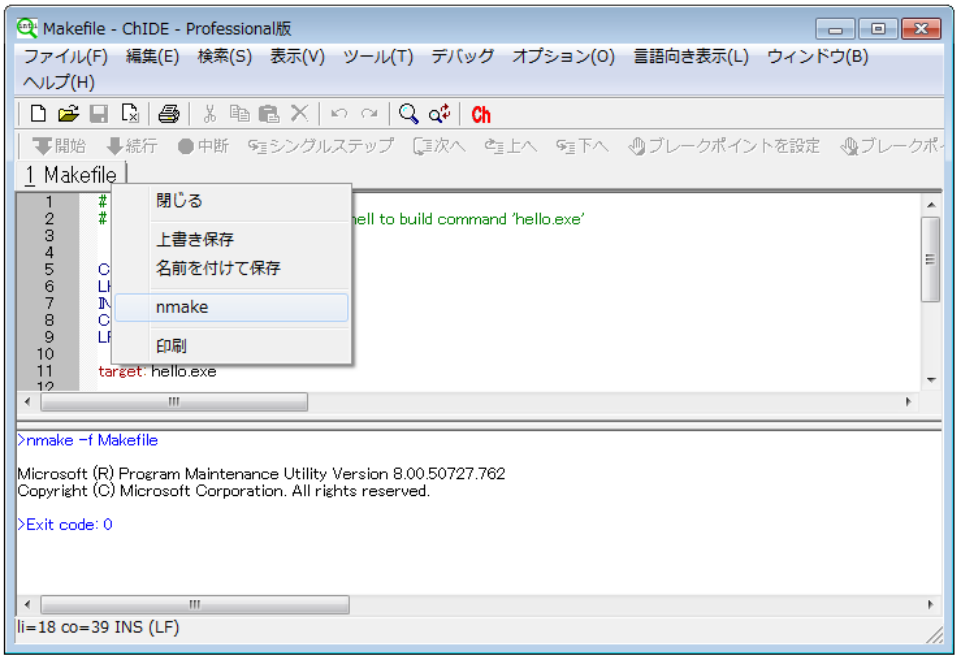


図 40: C/C++ プログラムをコンパイルするための makefile の使用

ChIDE が make ファイルの編集に使用される際には、その書式がハイライト表示されます。タブ文字は、ある make コマンドに対してはコマンドを開始する特殊文字として予約されていますので、この文字は保護され、空白文字では置き換えられません。ファイル拡張子 `.mak` をもつファイル、または以下の名前のファイルは、ChIDE では make ファイルとみなされます。

```
makefile
makefile.win
makefile_win
makefile.Win
makefile_Win
Makefile
Makefile.win
Makefile_win
Makefile.Win
Makefile_Win
```

［ツール］ → ［実行］コマンドを実行することにより、作成された実行可能プログラムが実行されます。

9　ChIDE で使用できる他のコンピュータ言語

ChIDE は、汎用目的のテキストエディタです。現在のところ、以下の言語に対して書式ハイライト機能が使用可能です。

- C/Ch/C++*
- CSS*
- HTML*
- Make
- SQL and PLSQL
- TeX and LaTeX
- XML*

シンボル'*'が付けられている言語は、セクション 3.5で記述されているような折りたたみがこの言語に対応していることを表しています。

言語設定はファイル拡張子から決定されますが、この設定は［言語向き表示］メニューから他の言語を選択することにより、変更可能です。

10 ChIDE で使用できる他のファイルフォーマット

ChIDE は can handle not only C/Ch/C++および他のコンピュータ言語だけでなく、いくつかの共通に使用されるメディアファイルも扱うことができます。Windows および Mac OS X では、ChIDE が拡張子 `.jpg`, `.png`, `.mpg`, `.avi`, `.mov`, `.wmv`, `.gif`, `.mp4`,`.doc`, `.docx`, `.xls`, または `.xlsx` をもったファイルを開く際に使用されると、これらの画像、ビデオ、文書タイプのファイルを開くように割り当てられた外部アプリケーションが起動されます。

11 ChIDE で対応している各国言語

Ch が、英語とは異なる言語のプラットフォーム上にインストールされると、ChIDE のメニューとダイアログはそのローカル言語になります。既定では、ChIDE は以下の 30 以上のローカル言語に対応しています。

アフリカーンス語、アラビア語、バスク語、ブラジル・ポルトガル語、ブルガリア語、カタロニア語、中国語・簡体字、中国語・繁体字、チェコ語、デンマーク語、オランダ語、フランス語、ガリシア語、ドイツ語、ギリシャ語、ハンガリー語、インドネシア語、イタリア語、日本語、韓国語、マレーシア語、ノルウェー語、ポーランド語、ルーマニア語、ポルトガル語、ロシア語、セビリア語、スロベニア語、スペイン語、スペイン語（メキシコ）、スウェーデン語、タイ語、トルコ語、ウクライナ語、およびウェールズ語

新規のローカル言語にも容易に対応することが可能です。

索引