Vircon32:C言語を学ぶ

資料作成日2023.01.23年Carra著

これは何だ?

このドキュメントは、C言語でプログラミングを始めるためのクイックガイドです。このガイドの目的は、プログラムの構造、C言語の基本的な機能についての基本的な理解を提供し、Vircon32コンソール用のCプログラムを作成したい人に必要なものを示すことです。ただし、関数、変数、配列などの基本的なプログラミング概念を知っている必要があります。

知っておいた方がいいこと!

このガイドでは、Cの基本的な機能、つまりCプログラムを作成するために最低限必要な機能についてのみ説明します。高度な言語機能のいくつかを意図的に省略しているまた、標準C言語とVircon32で使用されている言語との間には小さな違いがあることに注意してください。このガイドでは、Vircon32バージョンに焦点を当てています。この詳細については、Vircon32 Cコンパイラのガイドを参照してください。

まとめ

このドキュメントはいくつかのセクションで構成されており、それぞれのセクションでは、Vircon32とそのコンパイラで使用されているC言語について、より詳細に説明します。

[Summary 1](#_Toc125398256)

[Introduction 2](#_Toc125398257)

[Structure of a C program 2](#_Toc125398258)

[Comments 3](#_Toc125398259)

[Data types 3](#_Toc125398260)

[Literal values 5](#_Toc125398261)

[Variables 5](#_Toc125398262)

[Pointers 7](#_Toc125398263)

[Functions 7](#_Toc125398264)

[Expressions 9](#_Toc125398265)

[Flow control 12](#_Toc125398266)

[The preprocessor 14](#_Toc125398267)

[The standard library 15](#_Toc125398268)

# はじめに

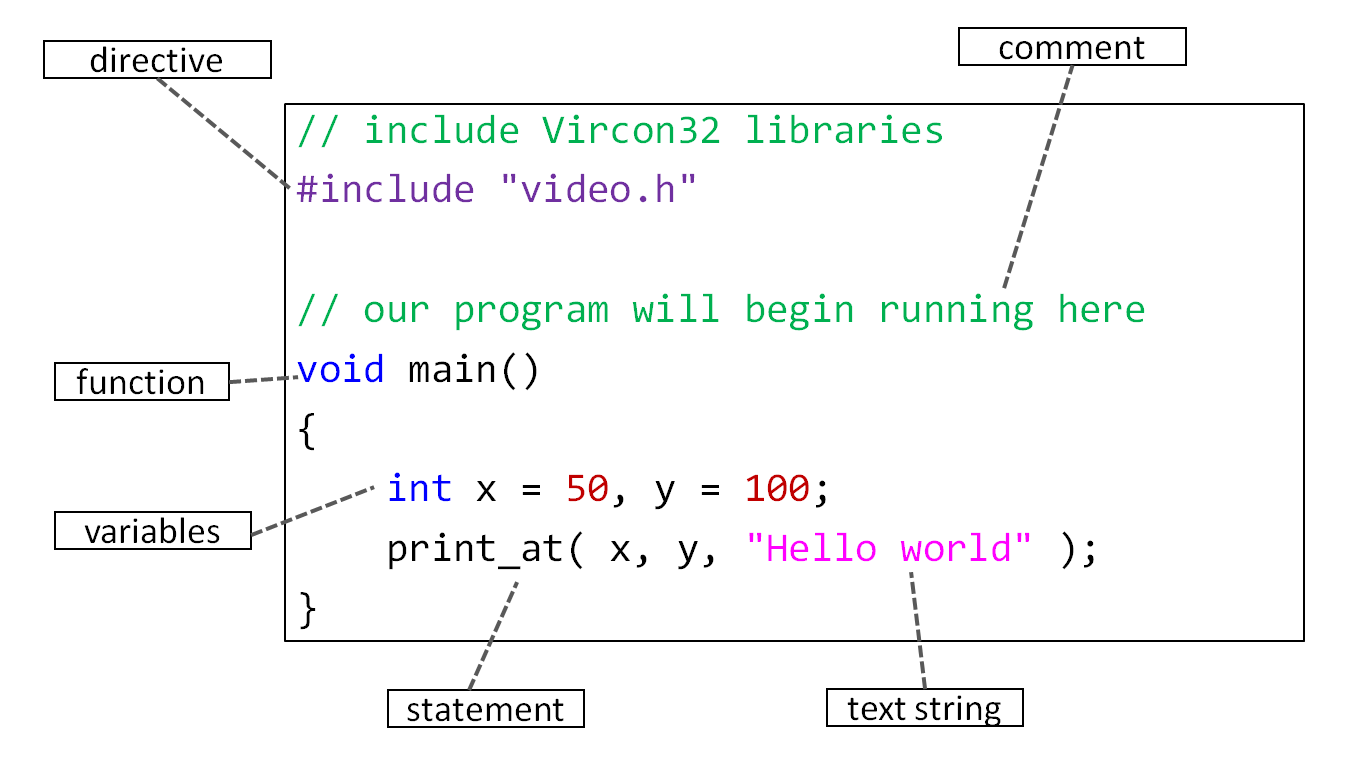
Cはコンパイルされた言語です。つまり、実行する前に、作成したコードをVircon32 CPUが実行できるマシン命令に変換する必要があります。このコンソールでは、プロセスは次のようになります。



CPU命令を含むバイナリファイルが作成されたら、そのプログラムを必要なイメージとサウンドとともに.v32ファイルにパックする必要があります。しかし、このガイドはプログラムの作成にのみ焦点を当てています。ゲーム作成の全プロセスについては、他のVircon32ガイドで読むことができます。

Cプログラムの構造

簡単なhello worldプログラムを使用して言語チュートリアルを開始する必要があります。ここでは、Cプログラムの最も基本的な構造を確認できます。



## 主な機能

コードの実行は常にこの関数の先頭から開始されるため、main関数は常に存在する必要がある特別な関数です。このコンパイラのmain関数は引数を取らず、値を返さないため、次のプロトタイプで宣言する必要があります:

|  |
| --- |
| 空所関数main()//thisまたは:void main(void)のいずれか  {を  //.//.//  }をクリックし |

コメント

コメントは、コンパイラが解釈しようとすることなく、独自の説明をプログラムに含める方法です(これによりエラーが発生します)。つまり、コメントはプログラム自体の一部ではありません。コメントには2種類あり、標準C言語と同じように機能します。

## 行コメント

これらのコメントは次で始まります。//////の順にクリックし、線の終点で終了します。

|  |
| --- |
| 内部x=7+5//コメントはこの行にのみ影響します  内部y=xである。 |

## ブロックコメント

ブロックコメントは次で始まります。/\*(該当日本語なしで終わるまで終わらない\*/\*/\*/(別の行で実行された場合でも)。

|  |
| --- |
| /\*このコメントは、次の行を否定します  int x=7+5;  \*/\*/\*/  内部y; |

データ型

言語によって処理されるすべての値または変数は、その値の解釈方法を示すデータ型に属します。ここでは、使用できるデータ型を確認します。

## 基本タイプ

Vircon32は、メモリを32ビットワードで構成します。このコンソールでは、メモリの最小単位がワード(バイトではなく)であるため、次のことを考慮する必要があります。

* データ型のすべてのサイズはバイト単位ではなくワード単位で測定される
* すべてのメモリアドレスとオフセットはバイト単位ではなく、ワード単位で指定されます。

このコンパイラには、次の4つの基本データ型があります。

|  |
| --- |
| int float bool void(int浮動小数点ブール値 |

これらの型はすべてサイズ1(つまり32ビット)を持ちます。void型は特殊であり、関数が値を返さないことを示すためにその基本形式でのみ使用できます。

## 派生タイプ

配列とポインタの基本的な型を使用して新しい型を作成できます。これは、次のように複数回実行できます。

|  |
| --- |
| 内部※印空所\*\*\*\*\*\*浮く[3][5]空所\*[4] |

voidを使用してポインタを作成できますが、配列は作成できません(void型の値が存在しないため)。Cでは、情報がビットとして読み取られたり格納されたりするメモリアドレスを参照する方法として、ポインタをvoidにすることができます(voidは型が存在しないことを示します)。

## 複合型

構造体と共用体を作成することによって、いくつかのデータ型をより大きな型にグループ化できます。これらのグループの各メンバーは、名前でアクセスされるフィールドです。

|  |
| --- |
| //Point型を宣言  構造体[点]  {を  内部x,y;x,y;x  }  //Word型を宣言  ユニオンワード  {を  内部AsInteger  空所※印AsPointer  }  //いくつかの構造体を使用  点P1、P2  P1.x=10  P1.y=-7  P2=P1  //いくつかの共用体を使用  単語W;  W.AsInteger=0xFF110AF  int\*ポインタ=W。AsPointer |

構造体や共用体には、他の構造体や共用体を含めることもできます。構造体や共用体自体を含めることもできますが、明らかな理由からポインタを使用する必要があります。

## 列挙

一連の整数定数を定義し、それらを独自の型にグループ化することができます。C言語では、列挙によってこれを可能にします。一般に、これらの値は整数として扱われ、同じ方法で使用できますが、型はより限定的です。

値が指定されていない場合、最初の定数の数値は0になり、後続のすべての定数の値は、前の定数に1を加えた値になります。

|  |
| --- |
| //タイプSemaphoreを宣言する  列挙値セマフォー  {を  赤=1  黄//の値は2  緑//の値は3  }  //これらの操作は正常です。  セマフォーS1、S2  S1=赤色  S2=S1である。  内部値=黄+緑//安全にenumをintに変換できる  //これらの割り当てはエラーを生成します!  S1=<数値>緑-赤  S2=1//Redと同じ値ですが、intをenumに変換できません |

リテラル値

私たちのプログラムでは、文字どおりに記述された定数値を使用できます。これらの値のそれぞれが表すデータ型に応じて、異なる表記と数値表現があります。このコンパイラには次のものがあります:

|  |
| --- |
| -15インチ//10進数のint  0xFF1A、0 xFF 2 A、0 x//int(16進数)  0.514人//浮動小数点  真//ブール  「a」は//intを文字として使用(charは存在しません)  「ハイ!」//int[4](3文字の文字列+終了0)  特殊//NULLポインタ |

ブールリテラルの値は、true=1、false=0です。

変数

定数データに加えて、変数も処理できます。変数は、変化する値が格納されるメモリアドレスです。この値にアクセスするには、変数を識別する名前と、格納された値を解釈するデータ型を使用します。

Cでは、変数は宣言された値の型(整数など)のみを格納できます。他のいくつかの言語では、型のない変数を宣言し、異なる値(数値、文字列など)を格納し続けることができます。これはCでは許可されていません。

## 変数の宣言

変数は型と名前で宣言され、オプションで値で初期化できます。

|  |
| --- |
| 浮く[速度]=2.5  ブール有効: |

このコンパイラでは、型は常に名前とは別に保持されます。この場合は<type><name>です。たとえば、このコンパイラで配列を宣言するには、次のようにする必要があります。

|  |
| --- |
| //これは5つのfloatの配列です。  浮く〔5〕BallSpeeds  //これは20個の位置を持つ配列で、それぞれが10個のintの配列です。  内部[20][10](英語)LevelBricks |

## 複数の宣言

1つの宣言で同じ型の複数の変数をカンマで区切って宣言することもできます。この方法で宣言された変数はすべて同じ型になります。

|  |
| --- |
| //intへの3つのポインタを宣言する  内部\*ptr1=&数値,ptr2=ptr1,ptr3; |

## 初期化リスト

配列および構造体は、複数の値のリストを使用して初期化できます。これらのリストは、次の例に示すように、より複雑なタイプに対してネストすることもできます:

|  |
| --- |
| //構造体を初期化する  構造体[点]  {を  内部x,y;x,y;x  }  点P={3,-7}  //ここでは、構造の配列にネストされたリストを使用します。  点[3]TrianglePoints={{0,0},{1,0},{1,1}}  //int配列の場合、リストの代わりに文字列を使用することもできます。  内部[10]を参照してください。テキスト=「こんにちは」//慎重に、Cは終了として1文字(0)を追加します。 |

## 変数のスコープ

C言語には、ローカル変数とグローバル変数という2種類の基本的な変数があります。

* ローカル変数は関数の本体で宣言され、そのスコープ内でのみアクセス可能です(スタックに格納され、実行によって変更されるため)。
* グローバル変数は関数の外部で宣言され、アドレスが固定されているため、宣言後にプログラム全体からアクセスできます。

ポインタ

ポインタは、基本的なゲームを作成するのに実際には必要ではありませんが、それらが何であるかを把握したい場合があります。ポインタは、メモリアドレス(つまり、メモリ内の位置を表す番号)を含む単なる変数です。たとえば、次のようになります。

|  |
| --- |
| 内部変数=7;//変数がメモリ位置1250に格納されているとします。  内部\*ポインタ=&変数;//ポインタには現在、番号1250が含まれています |

この場合、ポインタはint\*型として宣言されます。これは、ポインタが指しているメモリ位置が何であれ、整数として解釈されることを意味します。そのため、ポインタを仲介として使用することで、変数に格納されている値を間接的に読み取るか、変更することができます。

## ポインタが役立つのはなぜですか。

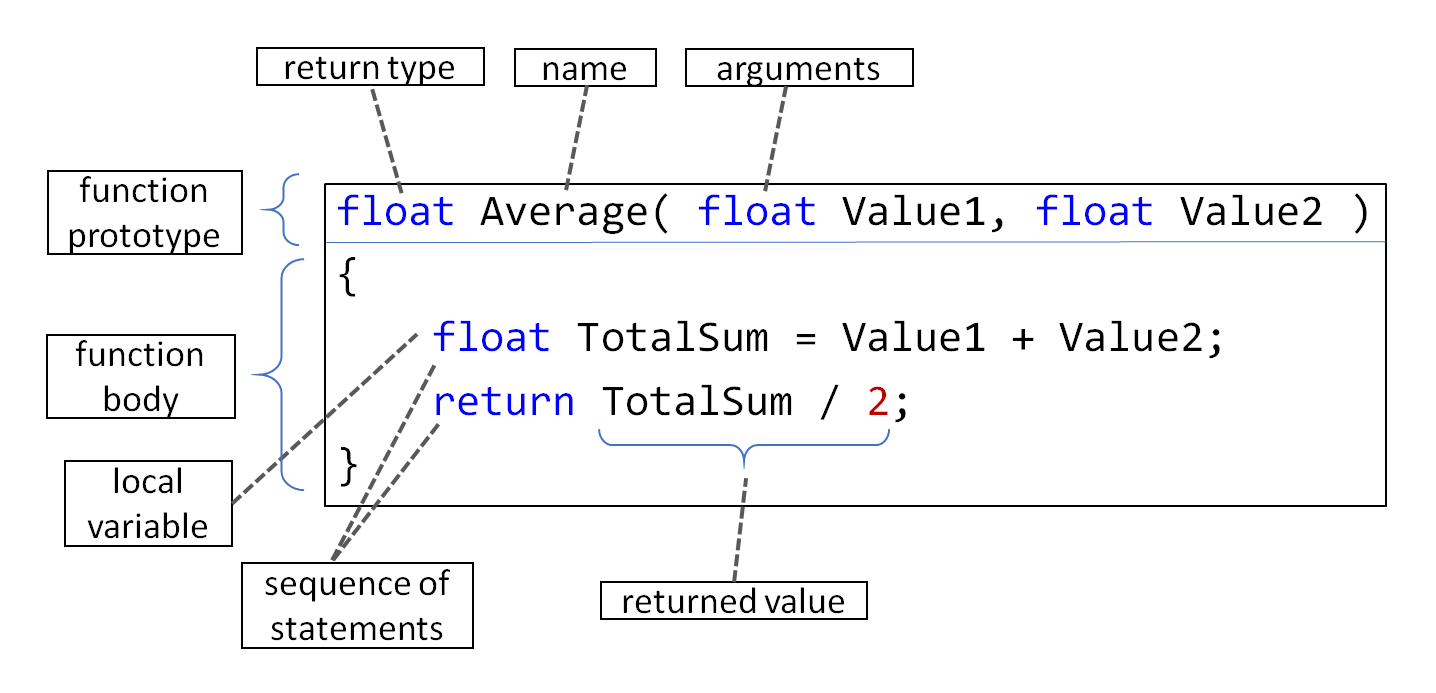
ポインタはさまざまな用途に使用できますが、典型的な例としては、関数にポインタを渡して外部データを操作できるようにすることが挙げられます。たとえば、外部変数に2つのポインタを渡すことによって、2つの値を返す関数を作成できます。または、構造体や配列が大きすぎてパラメータとして渡すことができない場合は、その構造体へのポインタを操作して、その構造体を適切に読み取り/変更します。

## NULLポインタ

Cには、ポインタ用の特別なリテラル値NULLもあります。NULLは、ポインタが現在何も指していないことを指定できるように提供されています。NULLの値は通常、無効なメモリアドレスに設定されるため、そこから読み書きしようとするとメモリ違反が発生します。これは、プログラムエラーの検出に役立ちます。

関数

C言語では、関数の外部で文を実行することはできません(宣言以外)。プログラムでコードを実行するには、実行する文を含む関数を宣言する必要があります。C言語の関数は、次のように宣言されます。



関数の本体には、順次実行される複数のステートメントを含めることができます。

## 関数からの復帰

関数本体の内部では、次のように使用できます。返却関数を終了します。実行は、その関数が呼び出されたプログラム内のポイントに戻ります。

returnは、値を返すためにも使用されます(関数の戻り値の型がvoidでない場合)。その場合、returnは互換性のある型の値とともに使用する必要があります。

|  |
| --- |
| //値を返さない関数  空所DoNothing()関数  {を  返却  }をクリックし  //ポインタを返す関数  int\*(整数)FindLetterA(のint\*(整数)テキスト)を  {を  一方(テキスト)  {を  もし(\*テキスト==「A」キー)を  返却テキスト;    テキスト++;  }をクリックし    返却特殊  }をクリックし |

## 機能の制限

標準Cとの重要な違いは、コンパイラの制限により、関数は1とは異なるサイズのパラメータや戻り値を受け取ることができないことです。つまり、配列、共用体、または構造体を使用することはできません(サイズが1つの単語でない限り)。代わりに、それらへのポインタを使用して操作する必要があります。

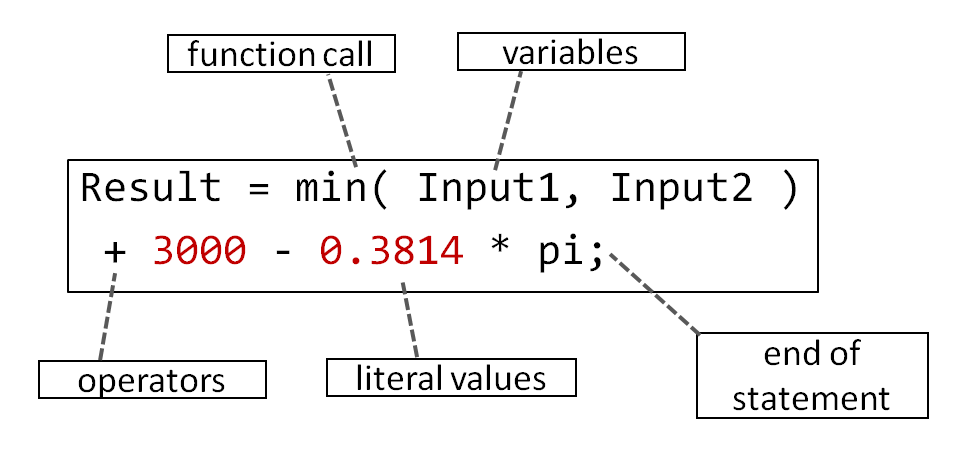
しかし、標準C言語と同じように、配列がポインタに分解されるときに、パラメータとして「配列を渡す」ことができます。例として、次のようにします。

|  |
| --- |
| //N個の整数を加算する関数  内部合計値(int\*(整数)値内部数)  {を//.//.//}をクリックし  //10個の整数の配列  内部[10]値  //配列の最初の5つの値を加算する  内部Sum=SumValues(値,5); |

この場合、「array decay」により、array Valuesはその最初の要素へのポインタとして評価されるため、整数へのポインタを期待する関数によって受け入れられました。

式

式は、変数とリテラル値を組み合せることによって実行できる操作です。これらの組み合せは、結果として単一の最終値が得られるまで、関数または演算子を適用することによって実行できます。次の例の式の要素を参照してください:



## 演算子

ほとんどの場合、演算子は、関数として記述されるのではなく、オペランド間に記号(+や-など)として挿入される数学的演算を表します。このコンパイラでは、標準C言語と同じ演算子が使用され、優先順位と関連性も同じです。

使用可能な演算子はすべて、タイプ別にグループ化されています。

算術演算子

|  |  |
| --- | --- |
| オペレータ | 意味 |
| +aキー | a(影響なし) |
| -aキー | 符号の変更 |
| aとb | 合計 |
| ア～イ | 差異 |
| ア\*イ | 製品 |
| あ/b | 事業部 |
| a%b個 | モジュロ |

比較演算子

|  |  |
| --- | --- |
| オペレータ | 意味 |
| a==bと入力します。 | 等しい |
| a!=b<数値> | 次と等しくない |
| a<bの場合 | より小さい |
| a<=bの場合 | 以下 |
| a>bの場合 | より大きい |
| a>=bの場合 | より大きいか等しい |

論理演算子

|  |  |
| --- | --- |
| オペレータ | 意味 |
| 行 | 論理否定 |
| アb | 論理OR |
| aとb | 論理積 |

ビット演算子

|  |  |
| --- | --- |
| オペレータ | 意味 |
| ~aへ | ビット方式のNOT |
| アb | ビット単位のOR |
| aとb | ビット単位のAND |
| ア^イ | ビット単位のXOR |
| a<<b<値> | ビットシフト左 |
| a>>b<該当日本語なし> | ビットシフト右 |

インクリメント演算子とデクリメント演算子

|  |  |
| --- | --- |
| オペレータ | 意味 |
| a++<数値> | aは最初に評価され、その後1ずつ増分されます。 |
| あ… | aは最初に評価され、その後1ずつデクリメントされます。 |
| ++aキー | aは、最初にによって増分されます。  1およびTHEN評価 |
| --aを | aは最初に1だけデクリメントされ、その後評価されます。 |

代入演算子

|  |  |
| --- | --- |
| オペレータ | 意味 |
| a=bの場合 | アサイン |
| a+=b<数値> | a=a+bです。 |
| a-=b<数値> | a=a-b<整数> |
| a\*=bとなります。 | a=a\*bとなります。 |
| a/=b<値> | a=aまたはb |
| a%=bの場合 | a=a%b(%) |
| a=bの場合 | a=a b<整数> |
| aおよび=b | a=aおよびb |
| a^=b<数値> | a=a^b<数値> |
| a<<=bの場合 | a=a<<bの場合 |
| a>>=bである | a=a>>bとなります。 |

ポインタの演算子

|  |  |
| --- | --- |
| オペレータ | 意味 |
| &A(& | のメモリアドレス  (aへのポインタを作成) |
| \*ア | によってポイントされる値  (そのアドレスの内容) |
| 行 | a==NULL(ヌル) |

配列の演算子

|  |  |
| --- | --- |
| オペレータ | 意味 |
| a[b]の場合 | 配列a内にインデックスbを持つ要素 |

構造体と共用体の演算子

|  |  |
| --- | --- |
| オペレータ | 意味 |
| ア・イ | a内のメンバーb |
| ア→イ | aが指す構造体内のメンバーb |

その他の演算子

|  |  |
| --- | --- |
| オペレータ | 意味 |
| イ | 内部の操作はグループ化され、他のオペレータより先に評価されます。 |
| sizeof(a)[サイズ変更] | 引数のサイズをワード単位で返します。 |

## プロモーションを入力

プログラマとしては、この言語が必要に応じて基本データ型間の自動変換を実行することに注意する必要があります。たとえば、次のようにします。

|  |
| --- |
| 内部PlayerScore=28.3;//floatはintに変換する必要があります |

コンパイラはこの値を自動的に整数に変換し、変数は実際に28を格納します。同様のことは、式が異なる型で動作する場合にも起こります。int+floatを追加するには、まずintをfloatに変換する必要があります。これにより、コヒーレントな型(float+float)間で動作が有効になります。

## 関数呼び出し

括弧とカンマの標準表記を使用して、すでに宣言されている任意の関数を呼び出すことができます。

|  |
| --- |
| //変数と定数を追加  内部Sum=SumValues(変数,5);  //関数呼び出しをネストできる  PrintNumber(SumValues(変数,5)); |

関数が値を返さない場合(戻り値の型がvoidの場合)、その関数の呼び出しは、結果を生成しない式であるため、他の式では使用できません。

## Sizeof演算子

の関数sizeof()演算子は、データ型または式の結果のサイズ(常に32ビットワード単位)を決定します。

|  |
| --- |
| //式の結果のサイズ  内部サイズ1=サイズ(2+5)//サイズ1=1(int)  //データ型のサイズ  内部サイズ2=サイズ(の内部[2][5])//サイズ2=10 |

フロー制御

Cプログラムは、式のシーケンスを使用して構築されるだけでなく、実行フローを制御するためのツールも必要です。次のものを使用できます。

## 条件

実行を継続する場所を制御する最も基本的な方法は、単純な条件です。もしおよびそれ以外私たちは条件を評価し、それが満たされた場合に何が起こるか、満たされなかった場合に何が起こるかを決定することができます。

また、いくつかのifをチェーンして、関連する条件をチェックすることもできます。

|  |
| --- |
| //チェーニングされたif  もし(x>0)  印刷(「正」)  else if文(x<0の場合)  印刷(「負」)  それ以外  {を  印刷(「ゼロ」)  ShowAlert x  }をクリックし |

## スイッチ

複数の整数値から選択する必要がある場合は、複数のif-else条件をチェーンする代わりにスイッチ一連のケースから選択します。

|  |
| --- |
| スイッチ(WeaponPowerLevel)  {を  //中電力の影響  事件1点:  事件2点:  Player.健康=WeaponPowerLevel;  ブレーク    //プレイヤーを破壊するハードインパクト  事件3点:  MakePlayerExplode x  ブレーク  //影響が弱すぎる場合、またはエラーの可能性がある場合  デフォルト(該当日本語なし  MakeBulletBounce x  ブレイク;  }をクリックし |

## ループ

Cは、次を使用した直接ジャンプをサポートし移動Cには3種類のループがあります。一方するおよび対象です。

|  |
| --- |
| //whileは先頭の条件をチェックします。  一方(x!=0)を返します。  x/=2;  //最後に条件をチェックする  する  {を  x-=7;  y=xである。  }をクリックし  一方(x>0)  //forは、より構成可能なループです。  //ステートメント、終了条件、および反復ステートメント  対象(の内部i=-5;i<=5;i+=2)  print\_number(i);<数値> |

## ループ制御

ループの内側ではブレークを使用してループを終了し、終了後も実行を継続します。続けるループの最後に到達することなく、ループの次の繰り返しに進むことができます。

|  |
| --- |
| 一方(敵.EnemyID<MaxEnemyID)  {を  ++敵;    //アクティブな敵のみを処理  もし(!敵アクティブ)  続ける    ProcessEnemy(&プレイヤー,敵);    //プレーヤーが終了した場合、続行する必要はありません  もし(Player.Health<=0)  ブレーク  }をクリックし |

プリプロセッサ

C言語は、Cプログラム内のすべてのテキストの前処理を実行します。プリプロセッサは、プログラムを構成するファイルを1行ずつ移動してディレクティブを検索し、検索および認識可能なディレクティブを適用します。

ディレクティブはハッシュ文字で始まります番号これは、行の最初の文字である必要があります(空白を除く)。使用できるディレクティブは次のとおりです。

## ディレクティブ#include

使用方法#含むでは、コードのある時点で別のファイルの内容を挿入できます。これは、異なる機能を別々のファイルに分離することによってプログラムを編成できるので便利です。

|  |
| --- |
| //ヘッダーを含める  #include"敵\\ボス.h" |

プリプロセッサは、まず標準ライブラリフォルダでファイルを検索し、次にソースファイル自体のディレクトリで検索します。パスは他のテキスト文字列と同様に扱われます。「\」などの特殊文字が含まれている場合は、エスケープシーケンスを使用して記述する必要があります。

## ディレクティブ#defineおよび#undef

プリプロセッサは内部変数のリストを保持します(Cプログラム変数と混同しないようにしてください)。#定義の順にクリックし#定義解除定義には値を指定できますが、空にすることもできます。

|  |
| --- |
| //リテラル値を持つ単純な定義  #define BallDiameter 16  //式で前の定義を使用する定義  #defineボール半径(BallDiameter/2)  //他の古い定義を削除する  #undef BallSize |

このコンパイラのdefineディレクティブにはいくつかの制限があります。標準のCプリプロセッサのようにパラメータと一緒に使用することはできません。

標準ライブラリ

Cコンパイラは、プログラムが一定の方法で動作する最小限の機能にアクセスできるように、異なる主題(数学、文字列処理など)に対して一連の関数を実装する必要があります。これらの関数は、コードに標準言語ヘッダーを含めることによってプログラムからアクセスできます。

Vircon32の標準Cライブラリには、多数の標準C関数が含まれていますが、すべてではありません。一方、これは特定のコンソール用のコンパイラであるため、Vircon32標準ライブラリには、オーディオ、ビデオ、ゲームパッドなど、さまざまなコンソールシステムで動作する関数も追加されています。

Vircon32コンパイラの現在のヘッダコレクションは次のとおりです。

## "オーディオ.h"

オーディオチップを操作してサウンドを再生するために使用されます。

## "入力.h"

ゲームパッドの状態を読み取ることができます。

## "math.h(数学.h)"

C言語の最も一般的な数学関数。

## "memcard.h"と入力します。

メモリカードにアクセスして、データを読み取ったり保存したりするために使用します。

## "misc.h(その他のファイル)"

その他の関数:メモリ管理、乱数など。

## "文字列.h"

テキスト文字列を構築および処理するための関数が含まれています。

## "時間.h"

これにより、時間の流れを測定し、プログラムの速度を制御することができます。

## "ビデオ.h"

ビデオチップにアクセスし、画面上にイメージを表示するために使用されます。