[OpenGL 2D 2018 第02回]

プログラムはどうやってできるの？

# プログラムの流れ

プログラムは順番に実行されるのでしたね。  
C++言語では、この「順番」というのはだいたい次のような規則になっています。

1. 上から1行ずつ実行する。行の終わりは改行ではなく記号「;(セミコロン)」で示される。
2. 各行は左から実行する。
3. ただし、括弧の中身は先に実行する。
4. 数の計算は、算数・数学と同じく掛け算、割り算を先に実行する。

早速ですがApplication.cppがどのように動作するか、その流れを追いかけてみましょう。  
先頭から順番に見ていきます。

## コメント

1 /\*\*  
 2 \* @file Application.cpp  
 3 \*/

この部分は「コメント(注釈・意見)」と呼ばれるものです。コメントは「/\*(スラッシュとアスタリスク)」で始まり「\*/(アスタリスクとスラッシュ)」で終わります。  
コンピューターはコメントを無視します。ですから、プログラムとしては何の意味もありません。しかし、プログラムには、コンピューターにとって実行に必要な情報しか書かれません。そのため、例えばこのプログラムが何のために書かれたのか、なぜこの順番で命令を並べたのか、といった情報はプログラムには現れず、それを書いた人にしか分かりません。そこで、あとから誰かがプログラムを読んだときに、書いた人の意図を伝える手段としてコメントを使います。

NOTE: もっとも、この先頭のコメントにはそこまでの意図はなくて、単に見出しとしてファイル名を書いているだけなのですが。

気をつけてもらいたいのは、コメントの中にさらにコメントを含めることはできない、という点です。つまり、  
「/\*あい/\*う\*/えお\*/」のような書き方はできません。この場合、コメントとみなされるのは最初の「\*/」まで、つまり「/\*あい/\*う\*/」となり、残りの「えお\*/」は「プログラムの一部」とみなされます。

コメントは一般的に、その下あるいは左にあるプログラムの要素について書かれます。そうしなければならないわけではなく、ただの慣習です。ですが、実際に多くのプログラムはそのように書かれているので、慣れておいたほうがいいでしょう。

## インクルード文

4 #include "Command.h"

4行目は「インクルード文(またはインクルード指令)」といいます。記号「#」で始まる命令は「プリプロセッサディレクティブ(前処理器指令)」と呼ばれるものです。プリプロセッサディレクティブは、プログラムをコンピューターが理解できるデータへと翻訳するときに(これは後ほど説明します)、事前に行うべき作業を指示するために使われます。そして、インクルード文は「この指令の位置に後述するファイルを読み込め」という指示になります。この場合「後述するファイル」とは「Command.h」になります。つまりコンピューターは、Command.hというファイルに書かれている内容が、このインクルード文の位置に書かれていたかのようにプログラムを実行するわけです。

「なぜそんな面倒なことを？そこに直接書けばいいじゃないか」という声が聞こえてきそうですが、わざわざインクルード文を使うのは、「同じことを何度も書きたくない」からです。プログラムというものは、一般的には機能ごとに分かれた複数のファイルから構成されます。そして、インクルード文によってファイルを取り込むことで、それらの機能を使えるようになるわけです。こうしておけば、ある機能を別のプログラムで使い回したいときには、そのプログラムにインクルード文を追加するだけで済む、という仕組みです。

## 1行コメント

6 // 前回「あいこ」のときtrue、そうでなければfalseが格納される.  
 7 // trueは「真」、falseは「偽」という意味.  
 8 // ひとつのイコール記号は代入を意味する.

C++言語には「/\*～\*/」以外にもうひとつコメントの書き方があります。このコメントは「//(スラッシュ2個)」で始まり、改行で終わります。簡単なコメントを書くときにいちいち最後に「\*/」を書くのは面倒だ、という理由で導入されました。コンピューターは1行コメントを無視します。

## 変数

9 bool aiko\_flag = false;

C++言語は、「変数」というものでプログラムの状態を管理します。変数は名前のとおり「変化する数」という意味で、プログラムの進行にしたがって、名前に紐付けられたデータが変化していくことから名付けられました。

変数を使うには、その変数がどのようなデータを管理するか(C++言語ではこれを「型」といいます)と、変数の名前、それと、最初にどんなデータを設定するか(C++言語ではこれを「初期値」といいます)、をコンピューターに伝えておかなければなりません。C++言語ではこれを「変数を定義する」と呼びます。

この行では「bool(ぶーる)」というのがデータの種類(型)で、「aiko\_flag」が変数の名前、「=(いこーる)」をはさんで右にある「false(ふぉーるす)」が最初に設定するデータ(初期値)、になっています。bool型の変数には「true(とぅるー)」、「false(ふぉーるす)」のいずれかのデータしか設定できません。  
なお、6～8行のコメントにあるように、「aiko\_flag(あいこ・ふらぐ)」は「前の勝負があいこだったか」を記録するために使われます。このようにbool型は「2つの状態のどちらかしか取り得ない」ようなデータを表現するための型だといえるでしょう。

11 // プレイヤーの勝ち回数.  
 12 int player\_win\_count = 0;  
 13  
 14 // CPUの勝ち回数.  
 15 int cpu\_win\_count = 0;

11～15行目はまた別の変数を2つ定義しています。  
この2つの変数のデータの種類(型)は「int(いんと)」というものです。intはC++言語において「整数」を記録するための型で、正の数も負の数も扱えますが、小数は設定できません。もし小数を設定しようとすると、小数点以下は切り捨てられます。

プレイヤーの勝ち回数やCPUの勝ち回数が小数になることはない(はず)ので、int型は十分に役に立ってくれるはずです。また、プログラムを開始した直後はどちらの勝ち数も0のはずですから、初期値として0を設定しています。

## 関数の定義

17 /\*\*  
 18 \* アプリケーションの本体.  
 19 \*/  
 20 void application()  
 21 {

20行目は「関数」と呼ばれるものです。C++言語において、「関数」はプログラムを作るための部品です。プログラムは関数を組み合わせて作られます。プログラムで関数を使うには、事前に関数を定義しておかなくてはなりません。コンピューターは融通がきかないので、まだ定義していない関数を使ってしまうと「こんな命令は知らないぞ！？」とパニックを起こしてしまうからです。

さて、「関数」を定義するには、まず「関数の実行結果であるデータの種類(型)」、「関数の名前」、「関数が必要とするパラメータ(引数(ひきすう)といいます)のリスト」、そして「関数の内容」をコンピューターに伝えなければなりません。  
この行の場合、「void(ゔぉいど)」が実行結果を示すデータの種類(型)です。voidは「結果を返さない」ことを示す型で、結果自体はどうでもいい場合に使われます。その次の「application」が関数の名前です。関数名の後ろの括弧が「関数が必要とするパラメータ(引数)のリスト」になります。パラメータ(引数)は「,(かんま)」で区切って指定します。しかし、ここで定義するapplication関数は特にパラメータを必要としないため、括弧の中には何も書いていません。  
関数の内容は「{(はじめなみかっこ)」で開始し「}(おわりなみかっこ)」で終了します。

NOTE: C++言語の「関数」は、みなさんが数学で習ってきた「関数」とはかなり違うものです。例えば数学の関数では、同じ値を入力したら、かえってくる値は常に決まっています。しかし、プログラムの関数はそうとは限りません。また、数学の関数は常に値をかえしますが、プログラムでは何もかえさない関数を作ることができます。

## 関数呼び出し(set\_image関数)

22 set\_image(0, 0, 0, "janken\_background.png"); // 背景を表示.

22行目では「set\_image(せっと・いめーじ)」という関数を使っています。C++言語では、このように関数を使うことを「関数を呼び出す」といいます。

ところで、ここまでどこにもset\_image関数が定義されていませんよね。それなのにここで使えるのは何故なんでしょう？　実はset\_image関数は「Command.h」というファイルの中で「宣言」されているのです。4行目のインクルード文でCommand.hを取り込んでいるので、そこより下の行では使うことができるというわけです。

「宣言」というのは、「これからこういう関数を使いますが、その内容はあとで定義します」ということをコンピューターに伝えるためのC++の機能です。これが便利なのは、例えばAとBという2つの関数があり、どちらももう一方を呼び出している、というような状況です。この場合、「A関数はB関数を使いたいのでB→Aの順に定義したいけど、B関数もA関数を使いたいのでA→Bの順で定義したい」ということになり、定義順を決めることができません。こんなときは、まずB関数を宣言し、次にA→Bの順で定義します。すると、コンピューターはA関数がB関数を使おうとしたときに「あ、これはあとで定義するって言ってた関数だな。じゃあ、ちょっと後ろを見てくるか。」というふうにうまくやってくれるのです。

さて、関数を呼び出すには、「関数が必要とするパラメータ(引数)のリスト」を渡してあげなければなりません。set\_imageの場合、3つの数値となんだか文字が並んだものが１つありますね。この「,(かんま)」で区切られた数字や文字の並びがパラメータ(引数)です。4つありますので「set\_image関数は4つのパラメータ(引数)を必要とする」ことが分かります。

## 関数の宣言

パラメータの意味は、関数を定義(あるいは宣言)するときに決めます。  
set\_imageの場合は、Command.hで宣言されているのでした。  
実際にCommand.hの中身を見てみると、次のような記述が見つかることでしょう。

/\*\*  
\* 画像を配置する.  
\*  
\* @param no 画像の管理番号.  
\* @param x 表示位置(X座標).  
\* @param y 表示位置(Y座標).  
\* @param filename 画像ファイルの名前.  
\*  
\* noで指定した管理番号に既に別の画像を配置していた場合、  
\* まずその画像を消去してから、改めてfilenameで指定した画像を配置する.  
\*/  
void set\_image(int no, float x, float y, const char\* filename);

コメントの部分がset\_image関数の機能とパラメータについての説明になっています。  
そして、その下の関数の定義みたいな部分が「宣言」です。  
宣言は定義とよく似ています。定義と同様に「関数の実行結果を示すデータの種類(型)」、「関数の名前」、「関数が必要とするパラメータ(引数)のリスト」があります。ですが、「関数の内容」はなくて、そのかわりに行の終わりを示す「;(せみころん)」があります。宣言はコンピューターに関数の存在を教えておいて「内容はあとで定義する」ための機能なので、関数の内容を書くわけにはいかないのです。  
そのかわりにセミコロンをつけて「関数の宣言はここまでです」ということを示すのです。  
関数の実行結果の型はapplication関数と同じくvoid型です。void型は「結果を返さない」ことを示すのでしたね。

## 引数

パラメータ(いい加減に毎回「パラメータ(引数)」と書くのが面倒になってきたので、以後は単に「引数」と呼ぶことにします)について、もうすこし詳しく見てみましょう。  
引数は「変数の定義」とよく似ています。まず引数の型があり、引数の名前が続きます。今回は使っていませんが、変数と同様に「最初に設定するデータ」を指定することも可能です。引数が2個以上ある場合は「,(かんま)」で区切って次々に引数を設定できます。

引数の型には、「int(いんと)」、「float(ふろーと)」、そして「const char\*(こんすと・ちゃー・ぽいんた)」という3つが使われています。floatとconst char\*型は今回はじめて登場しましたね。  
floatは小数を扱うことのできる型です。整数も扱えますが、内部的には「1.0」のように小数を含むデータとして扱われます。「そんな便利なものがあるのなら、最初からintなんて使わずに全部floatを使えばいいじゃないか」と思われるかもしれませんが、そうしないのには理由があるのです。第一に、多くのコンピューターは小数の計算があまり得意ではありません。そのためみだりにfloatを使うとプログラムの実行が遅くなってしまう恐れがあります。第二に、整数しか取り得ないデータに対して、プログラムのミスで小数を加えたり引いたりしてしまうかもしれません。一人で開発しているとあまりそういったことはないのですが、複数人で開発している場合、「この引数(または変数)は小数も扱えるのだから、小数を足しても大丈夫だろう」ととらえられることが考えられます。最初からintを使っておけば、こういった問題は防げるはずです。こういった理由から、intとfloatを使い分ける必要があるわけです。

const char\*は、文字の並び(「文字列」といいます)を扱うための型です。  
C++言語において、文字列とは「文字の並び」を「"(ダブル・クオーテーション)」という記号で囲ったものを指します。22行目のset\_image関数では「”janken\_background.png”」という部分が文字列にあたります。文字列の場合に「"」で囲うのは、プログラムの命令としての文字の並びと、データとしての文字の並びを識別する必要があるからです。

さて、set\_image関数宣言の手前のコメントを見ると、引数の名前と、その引数の説明が書かれているのが分かるでしょうか。改めてset\_image関数の呼び出しを見てみましょう。

22 set\_image(0, 0, 0, "janken\_background.png"); // 背景を表示.

コメントによれば、最初の引数は「画像の管理番号」だそうです。このプログラムでは、配置した画像ごとに管理番号を付けるようになっています。画像を消したり動かしたりするときは、この番号で対象となる画像を指定することになります。22行目では管理番号として0番を設定しています。

2つめの引数は「画像の位置(X座標)」となっています。このプログラムでは、ウィンドウの中心を原点として位置を指定します。また、数学の表と同じく、左右がX軸、上下がY軸になっています。22行目ではこの引数のデータは0となっていますので、X軸上の原点に画像を配置することになります。

3つめの引数は「画像の位置(Y座標)」です。これもデータは0ですから、2つめの引数と同様にY軸上の原点に画像を配置することになります。

4つ目の引数は「画像ファイルの名前」です。これが実際に配置される画像になります。22行目の場合はjanken\_background.pngとなっていますね。このファイルは「Res」という名前のフォルダに入っていますので、確認してみてください。

set\_imageの説明と、これら4つの引数を組み合わせると、この行がコンピューターに与える命令は「janken\_background.pngという名前の画像に0番の管理番号を付けて原点に配置せよ」という意味になります。

## if文

24 // あいこフラグがtrueなら"あいこで"画像を表示. falseなら"じゃんけん"画像を表示.  
 25 if (aiko\_flag) {  
 26 set\_image(1, 0, 100, "janken\_aikode.png");  
 27 } else {  
 28 set\_image(1, 0, 100, "janken.png");  
 29 }

24行目はその下のプログラムを説明するためのコメントです。  
25行目には「if(いふ)文」というC++言語の構文があります。簡単に言うと、括弧の中の条件が成立したら(言いかえると「真であれば」)、直後の波括弧の中のプログラムを実行し、成立しなければ「else(えるす)文」の直後の波括弧の中にあるプログラムを実行する、という機能です。そのうちC++言語の授業で習うと思いますので、詳しい説明はそちらに譲ります。  
見た目は関数と似ていますが、if文はC++言語に組み込まれた機能のひとつで、関数とは区別されます。C++言語はif文以外にもいくつかの単語を組み込みの機能として持っていて、それらは変数や関数の名前に使うことはできません。これらの単語は「予約語」と呼ばれ、if以外だと既に登場したbool, void, intなどもそのひとつです。Visual Studioで青色で表示される単語は予約語です(標準的な色設定の場合)。

26,28行目はset\_image関数を使っています。ここは、どちらの管理番号も1である点に注目してください。じゃんけんの掛け声は「じゃんけんぽん」か「あいこでしょ」のどちらか一方だけです。使わない掛け声を管理する必要はないので、同じ管理番号を割り振っても大丈夫なわけです。

課題1: 26行目と28行目がコンピューターに与える命令を説明しなさい。

## 関数呼び出し(set\_text関数)

31 set\_text(-360, 260, "あなた(%d勝)", player\_win\_count);  
 32 set\_text(40, 260, "CPU(%d勝)", cpu\_win\_count);  
 33   
 34 set\_text(-360, 0, "手を選んでください");

31～34行目では、また別のset\_text関数が使われています。  
これもCommand.hで宣言されています。ちょっと宣言を見てみましょう。

/\*\*  
\* 文字を表示する.  
\*  
\* @param x 表示開始位置の左端座標.  
\* @param y 表示開始位置の上端座標.  
\* @param format 文字の書式指定(printf関数のものと同じ).  
\* @param ... 追加の引数(printf関数のものと同じ).  
\*/  
void set\_text(int x, int y, const char\* format, ...);

引数のかんじはset\_image関数と似ていますね。最初の2つの引数xとyは、コメントの説明は少し違いますが、機能としてはset\_image関数の同名の引数と同じで、表示位置を制御するためのものです。ウィンドウの中心を原点とすること、左右がX軸で上下がY軸なことも同じです。

format引数は表示する文字を指定するものです。set\_image関数の文字列は画像ファイル名の指定に使われていました。数値が管理番号や表示座標になるのと同様に、文字列もさまざまな目的で使われるということです。

さて、引数リストの最後に見慣れない「…」という記号がありますね。3つの連続したピリオドで構成されるこの記号は「省略記号」と呼ばれます。C++言語では、最後の引数として省略記号を使うと、プログラムからは0個以上の任意の引数を設定できる関数として扱われます。31,32行目と34行目で設定している引数の総数が違うのはこれが理由です。  
C++言語の授業でprintf関数というものを習ったと思いますが、printf関数にさまざまな引数を設定できる理由は「printf関数の最後の引数が省略記号だから」なのです。省略記号の扱いは講義の内容を越えてしまうため、これ以上は説明しません(詳しく知りたいひとは「可変長引数」で検索してみてください)。

31行目がコンピューターに与える命令は「引数として与えられた文字列”あなた(%d勝)”とplayer\_win\_count変数から新たな文字列を作成し、座標(-360,240)に表示せよ」となります。

課題2: 32行目と34行目がコンピューターに与える命令を説明しなさい。

## 関数の結果

36 // プレイヤーの手を選んでもらう.  
 37 // 選ばれたのがグーなら0が、チョキなら1が、パーなら2がplayer\_handに格納される.  
 38 const int player\_hand = select(-360, -40, 3, "グー", "チョキ", "パー");

36,37行目はコメントです。  
38行目は何をしているのでしょう？　最初に型らしきものがあり、その後に名前らしきものがあり、イコールが続きます。ここまでのところは変数の定義に見えませんか？　まさにそのとおりで、この行はplayer\_handという名前の変数を定義しています。ただ、初期値を書くべき場所に、数値ではなくselect関数が書かれています。C++言語では、関数が実行結果を返す場合、実行結果を初期値として使うことができるのです。

また、変数の型としてintではなく「const int」が使われています。このconst、どこかで見たような…。そうです「const char\*」で使っていました。実はconst(こんすと)というのは型に追加の属性を与えるための機能なんです。constによって追加される属性は「変更禁止」です。  
「int」型の変数、例えばplayer\_win\_countは、プレイヤーが勝利するたびに1ずつ増やすことができます。対してplayer\_handは「const int」型なので、設定された初期値を変えることはできません。一度手を決めたら、あとから変えるようなずるっこは許さない、ということですね。  
このように、constは変更されるとまずいデータを保護するために使われます。

さて、select関数もCommand.hで宣言されている関数です。  
Command.hファイルを開いて関数宣言を確認してみましょう。

/\*\*  
\* 選択肢を表示して、選択された結果を得る.  
\*  
\* @param x 表示開始位置の左端座標.  
\* @param y 表示開始位置の上端座標.  
\* @param count 選択肢の数.  
\* @param a ひとつめの選択肢.  
\* @param b ふたつ目の選択肢.  
\* @param ... みっつ以上の選択肢があるなら、カンマで区切って追加していく.  
\*  
\* @return 選択された項目の番号.  
\* ひとつめの選択肢を0番とする.  
\*/  
int select(int x, int y, int count, const char\* a, const char\* b, ...);

引数の説明には目立ったところはありません。これまで説明してきた内容である程度理解できると思います。これまでの関数との一番の違いは、この関数が「実行結果を返す」ことです。実行結果を示すデータの型はint型で、実際に返されるのは、コメントにあるように「選択された項目の番号」になっています。

38行目がコンピューターに与える命令は「3つの選択肢”グー”,”チョキ”,”パー”を座標(-360, -40)に表示し、実行結果として選ばれた選択肢を示す番号を、player\_hand変数の初期値とする」です。  
また、select関数の説明から、”グー”が選ばれた場合は0、”チョキ”が選ばれた場合は1、”パー”が選ばれた場合は2が実行結果になる…ような気がします。

## 乱数

40 // CPUの手を選ぶ.  
 41 // 選んだのがグーなら0が、チョキなら1が、パーなら2がcpu\_handに格納される.  
 42 const int cpu\_hand = random(0, 2); // 0～2のいずれかの整数が無作為に選ばれる.

40,41行目はコメントです。  
そして42行目は、先ほど見た38行目と似た構文になっていますね。この行はcpu\_handという名前の変数を定義しています。その初期値はrandomという関数の実行結果です。

例によってrandom関数もCommand.hファイルで宣言されています。  
Comamnd.hファイルをひらいて宣言を見てみましょう。

/\*\*  
\* 乱数を得る.  
\*  
\* @param min 取りうる範囲の最小値.  
\* @param max 取りうる範囲の最大値.  
\*  
\* @return min以上max以下のランダムな値.  
\*/  
int random(int min, int max);

random関数は2つのint型の引数を受け取り、int型の結果を返す関数のようですね。  
randomを日本語にすると「乱数」です。乱数とは「サイコロの目のように、法則性がなく、予測不可能な数値」のことです。42行目のrandom関数の引数に設定されているのは0と2ですから「0,1,2のうちひとつが予測不可能な方法で選ばれて結果として返される」ということになります。そして返された値がcpu\_hand変数の初期値になる、つまり「CPUの選んだ手」になるというわけですね。

38行目のselect文では、”グー”が0、”チョキ”が1、”パー”が2になる(気がする)のでした。乱数の実行結果が0,1,2のいずれかになるということは「コンピューターがグー、チョキ、パーのいずれかを選んだ」と考えることができそうです。

## 画像を消す

44 reset\_image(1); // "じゃんけん"(または"あいこで")画像を消す.

reset\_image関数は、これもCommand.hで宣言されている関数です。

/\*\*  
\* 画像を消す.  
\*  
\* @param no 画像の管理番号.  
\*  
\* 対象の画像がすでに消されていた場合は何もしない.  
\*/  
void reset\_image(int no);

この関数はそんなに難しくはないと思います。  
44行目がコンピューターに与える命令は「1番の管理番号を付けられた画像を消す」というものです。

## switch文

46 // "ほい!"  
 47   
 48 // 左側にプレイヤーの手を表示  
 49 switch (player\_hand) {  
 50 case 0: set\_image(2, -200, 100, "janken\_gu.png"); break;  
 51 case 1: set\_image(2, -200, 100, "janken\_choki.png"); break;  
 52 case 2: set\_image(2, -200, 100, "janken\_pa.png"); break;  
 53 }  
 54  
 55 // 右側にCPUの手を表示  
 56 switch (cpu\_hand) {  
 57 case 0: set\_image(3, 200, 100, "janken\_gu.png"); break;  
 58 case 1: set\_image(3, 200, 100, "janken\_choki.png"); break;  
 59 case 2: set\_image(3, 200, 100, "janken\_pa.png"); break;  
 60 }

49行目と56行目にあるのは「switch-case(すいっち・けーす)文」というC++の構文です。  
これは「switchに設定した変数の値を見て、対応するcaseに書かれた命令を実行する」機能を持っています。switch-case文は、if文のように、条件によって異なる処理をさせたいときに使われます。  
49～53行目は、「player\_handが0なら”janken\_gu.png”を、1なら”janken\_choki.png”を、2なら”janken\_pa.png”を座標(-200,100)に配置せよ」という命令になります。  
56～60行目もほとんど同じことをしていますが、「条件にplayer\_handではなくcpu\_handを使っている」、「配置する座標が(200, 100)」、「管理番号が2ではなく3」という点が違っています。

後で見ますが、if文には複雑な条件を扱えるという利点があります。対してswitch-case文には読みやすく理解しやすい書き方ができるという利点があります。  
switch文もそのうちC++言語の授業で学習すると思いますので、くわしい説明はそちらにゆずります。

NOTE: 46行目のコメントは、「このタイミングで”ほい!”とか”しょ!”とかの画像を表示しようと思ったものの、いい画面構成が思いつかなかったので何もしなかった」ことの名残です。

## 時間をつぶす

62 wait(2); // 2秒間待つ

waitもCommand.hファイルで宣言されている関数です。  
宣言をは次のようになっています。

/\*\*  
\* 一定時間待つ.  
\*  
\* @param seconds 待ち時間(秒).  
\*/  
void wait(float seconds);

wait関数の機能は「引数に設定された秒数だけ待つ」というものです。  
ですから、62行目がコンピューターに与える命令は「2秒待て」となるわけです。

## 複雑なif文

64 // 数値の比較は二重のイコール記号「==」で行う.  
 65 // 「&&」は「且つ」、「||」は「又は」という意味を持つ記号.  
 66 // 式の優先順位を調整するには算数と同様にカッコ「()」を使う.  
 67 // グーは0、チョキは1、パーは2なので、例えばプレイヤーの手が0(グー)でCPUの手が1(チョキ)  
 なら、プレイヤーの勝ちとなる.  
 68 // 勝ったり負けたりした場合は「あいこ」じゃないのでaiko\_flagを「偽」にしておく.  
 69 // 勝ってもいないし負けてもいない場合は「あいこ」なのでaiko\_flagを「真」にしておく.  
 70 if ((player\_hand == 0 && cpu\_hand == 1) ||  
 (player\_hand == 1 && cpu\_hand == 2) ||  
 (player\_hand == 2 && cpu\_hand == 0)) {

70行目と74行目は、複雑な条件を使ったif文の例となっています(紙面の都合で改行していますが、1行のif文だと思って見てください)。  
コメントにあるように「==(二重のイコール)」は数値を比較する記号です。ここではplayer\_hand変数(プレイヤーが選んた手に対応する番号、つまり数値が設定されています)と数値のあいだの比較と、cpu\_hand変数(コンピューターが選んだ手に対応する数値が設定されています)と数値のあいだの比較に使っています。

最初にプログラムの実行順のルールを説明したとき「括弧の内側を先に実行する」というルールがあったことを覚えているでしょうか。そのルールに従って、まずは最初の括弧の中を見てみましょう。これは次の式になります。

player\_hand == 0 && cpu\_hand == 1

中間部分に「&&(二重アンパサンド)」という記号があります。コメントにあるように、「且つ」という意味を持つ記号です。これを組み合わせると、このプログラムは

プレイヤーの手が0と等しい、且つ、CPUの手が1と等しい

というように読み取れます。さらに、数値を実際の手に置き換えると

プレイヤーの手がグー、且つ、CPUの手がチョキ

となります。  
次の括弧の中も見てみましょう。これは次の式です。

player\_hand == 1 && cpu\_hand == 2

先ほどと同じように読み替えると

プレイヤーの手がチョキ、且つ、CPUの手がバー

となります。3つめの括弧の中も同様に読み替えると、

プレイヤーの手がパー、且つ、CPUの手がグー

になります。

3つの括弧の中身は解読できたので、次はその外側を見てみましょう。これらの括弧のあいだには「||(二重バーティカルバー、二重縦線)」がはさまっています。これは「又は」という意味の記号です。  
括弧の中身と||記号の意味、そしてif文をまとめると、次のようになるでしょう。

(プレイヤーの手がグー且つCPUの手がチョキ)  
又は(プレイヤーの手がチョキ且つCPUの手がバー)  
又は(プレイヤーの手がパー、且つ、CPUの手がグー)ならば、波括弧の中身を実行する。  
そうでなければelse以下を実行する。

ここで括弧の中に注目してみましょう。すると、3つともプレイヤーが勝つ条件になっていますね。  
つまりこのif文、結局は「プレイヤーの勝ちならば、波括弧の中身を実行する」という意味なのです。既にじゃんげんを知っている人間から見ると「なんて面倒なことをしているんだ」と思ってしまいますが、コンピューターはじゃんけんのルールを知らないので、くわしく条件を指示してあげる必要があるんですね。

71 set\_image(4, 0, -150, "janken\_kachi.png");  
 72 player\_win\_count += 1; // プレイヤーの勝利数を1増やす.  
 73 aiko\_flag = false;

続いて、プレイヤーが勝った場合に実行される、波括弧の中のプログラムを見ていきます。  
71行目は既に何度も出てきたset\_image関数です。ここでは”janken\_kachi.png”という画像を配置しています。

72行目には初めて見る記号「+=(ぷらすいこーる)」が登場しています。+=記号は「右辺の数値(または変数に設定されている数値)を左辺の変数に加算する」という意味です。72行目の+=記号の右辺は1、左辺はplayer\_win\_countですから、この行がコンピューターに与える命令は「player\_win\_count変数に1を加算せよ」ということになります。

73行目は初期値を設定するときにも使った「=(イコール)」記号が使われています。これは数学と同じで「代入」を意味します。「プレイヤーの勝ち→あいこではない」ということで、この行がコンピューターに与える命令は「aiko\_flagにfalseを代入せよ」です。

## if文はいくつでもつなげられる

74 } else if ((player\_hand == 0 && cpu\_hand == 2) ||  
 (player\_hand == 1 && cpu\_hand == 0) ||  
 (player\_hand == 2 && cpu\_hand == 1)) {  
 75 set\_image(4, 0, -150, "janken\_make.png");  
 76 cpu\_win\_count += 1; // CPUの勝利数を1増やす.  
 77 aiko\_flag = false;  
 78 } else {  
 79 aiko\_flag = true;  
 80 }

if文を解説したとき「条件が成立しなければelseの後ろにある波括弧の中身を実行する」と説明しました。実は、C++言語ではelseの直後にもう一度if文をつなげることもできるのです。if文をつなげることで、「条件AがだめならB、BもだめならC」というように、段階的な条件づけを行うことができます。if文のないelseはさいごにひとつだけ付けられます。また、「条件が成立しなかった場合は何もしない」というときは、elseもなくすことができます。

課題: 74行目がコンピューターに与える命令を説明しなさい

## すべての文字を消す

82 reset\_all\_text(); // いったんすべての文字を消す.

reset\_all\_textもCommand.hファイルで宣言されている関数です。  
この関数は次のように宣言されています。

/\*\*  
\* すべての文字を消す.  
\*/  
void reset\_all\_text();

コメントにも書いてあるように、現在画面に表示されている全ての文字を消したいときに使う関数です。実行結果の型がvoidなので、この関数は結果を返しません。また、引数リストに何も書いてないので、設定する引数もありません。

## 内側のif文

84 // 勝利回数が変化した可能性があるので表示しなおす.  
 85 set\_text(-360, 260, "あなた(%d勝)", player\_win\_count);  
 86 set\_text(40, 260, "CPU(%d勝)", cpu\_win\_count);  
 87   
 88 // 「あいこ」だった場合は自動的に勝負を継続する.  
 89 if (aiko\_flag == false) {  
 90 wait(3); // 3秒間待つ.  
 91 set\_text(-360, -40, "勝負を続けますか？");  
 92 const int yes\_or\_no = select(-360, -80, 2, "続ける", "やめる");  
 93 // "やめる"が選ばれたらアプリケーションを終了する.  
 94 if (yes\_or\_no == 1) {  
 95 quit(); // アプリケーションを終了させる.  
 96 }  
 97 }

84～92行目までは、これまで説明してきたC++言語の機能を使っているだけです。  
94行目のif文も説明済みですが、このようにif文の内側にさらにif文があるのは初めて見るものと思います。C++言語では、このようにif文の内側にさらにif文を書くことができるのです。内側に書けるのは何もif文に限ったことではなく、switch-case文なども使えます。また、switch-case文の内側でif文を使うことだってできてしまいます。

95行目、つまり内側のif文のさらに内側では、quitという関数が使われています。例によってCommand.hファイルで宣言されている関数で、次のように宣言されています。

/\*\*  
\* アプリケーションを終了する.  
\*/  
void quit();

1.18節で出てきたreset\_all\_text関数と同じく、「結果を返さない」、「引数を設定しない」というお気楽な関数になっています。その機能は「アプリケーションを終了する」こと。ただそれだけです。  
正確には、この関数が実行されたあと、application関数が終了した時点でアプリケーションが終了するのですが、とにかく「ゲームを終わらせたいとき」に使う関数だと思ってください。

課題: 89行目がコンピューターに与える命令を説明しなさい。  
 92行目がコンピューターに与える命令を説明しなさい。

## すべての画像を消す

98 reset\_all\_image(); // 次の勝負に備えてすべての画像を消す.  
 99 reset\_all\_text(); // 次の勝負に備えてすべての文字を消す.

98行目にはreset\_all\_imageという関数が使われています。みなさん予想が付いていることでしょうが、これもまたCommand.hで宣言されている関数です。  
この関数は次のように宣言されています。

/\*\*  
\* すべての画像を消す.  
\*/  
void reset\_all\_image();

reset\_all\_image関数は、現在画面に表示されている全ての画像を消したいときに使います。

## 関数の定義の終端

100 }

ようやく最後の行までたどり着きました。この「}(終わり波括弧)」は、1.5節で開始したapplication関数の定義の終了を示す記号です。つまり、21行目からこの100行目までがapplication関数ということになります。

なお、appliction関数は「quit関数を実行する」などの方法で直接・間接にゲームの終了を指示されない限り、何度でも繰り返し実行されます。