内容

[**前書き　Visual Studioの使い方** 3](#_Toc515837349)

[**プログラムの作成手順** 3](#_Toc515837350)

[**・章末テスト** 9](#_Toc515837351)

[**Lesson 1 はじめの一歩** 10](#_Toc515837352)

[**1.1 C++のプログラム** 10](#_Toc515837353)

[**1.2 コードの入力** 10](#_Toc515837354)

[**1.3プログラムの作成** 10](#_Toc515837355)

[**1.Ex C言語の標準出力関数を知ろう(補足)** 11](#_Toc515837356)

[**Lesson 1 章末テスト** 11](#_Toc515837357)

[**Lesson 2 C++の基本** 12](#_Toc515837358)

[**2.1 画面への出力** 12](#_Toc515837359)

[**2.2 コードの内容** 12](#_Toc515837360)

[**2.3 文字と数値** 13](#_Toc515837361)

[**Lesson 2 章末テスト** 15](#_Toc515837362)

[**Lesson 3 変数** 16](#_Toc515837363)

[**3.3 型** 16](#_Toc515837364)

[**3.5 変数の利用** 16](#_Toc515837365)

[**3.5ex(補足) printf関数を使用して、変数の値を出力する。** 17](#_Toc515837366)

[**Lesson 3 中間テスト1** 17](#_Toc515837367)

[**3.6 キーボードからの入力** 18](#_Toc515837368)

[**3.6.ex(補足) C言語の標準入力関数scanf\_s** 19](#_Toc515837369)

[**3.7 定数** 19](#_Toc515837370)

[**Lesson 3 章末テスト** 19](#_Toc515837371)

[**Lesson 4 式と演算子** 20](#_Toc515837372)

[**4.1 式と演算子** 20](#_Toc515837373)

[**Lesson4 中間テスト１** 21](#_Toc515837374)

[**4.2 演算子の種類** 21](#_Toc515837375)

[**Lesson 4 中間テスト２** 22](#_Toc515837376)

[**Lesson 4 中間テスト3** 25](#_Toc515837377)

[**4.3 演算子の優先順位** 25](#_Toc515837378)

[**4.4 型変換** 25](#_Toc515837379)

[**Lesson 4 中間テスト４** 28](#_Toc515837380)

[**Lesson 4 章末テスト** 29](#_Toc515837381)

[**Lesson 5 場合に応じた処理** 30](#_Toc515837382)

[**5.1　関係演算子と条件** 30](#_Toc515837383)

[**5.2　if文** 30](#_Toc515837384)

[**5.2 実習** 30](#_Toc515837385)

[**5.3 if～else文** 30](#_Toc515837386)

[**5.3 実習** 30](#_Toc515837387)

[**5.4 if～else if～else** 31](#_Toc515837388)

[**5.4 実習** 31](#_Toc515837389)

[**5.ex\_1** 31](#_Toc515837390)

[**Lesson 5 中間テスト１** 33](#_Toc515837391)

[**5.5 switch文** 33](#_Toc515837392)

[**5.6　論理演算子** 33](#_Toc515837393)

[**Lesson 5 章末テスト** 35](#_Toc515837394)

[**Lesson 6 何度も繰り返す** 36](#_Toc515837395)

[**Lesson6を始める前に** 36](#_Toc515837396)

[**6.1 for文** 36](#_Toc515837397)

[**Lesson 6 中間テスト１** 38](#_Toc515837398)

[**6.2 while文** 39](#_Toc515837399)

[**6.3　do～while文** 40](#_Toc515837400)

[**6.4　文のネスト** 40](#_Toc515837401)

[**6.4 ex デバッガの機能のウォッチを使ってみよう。** 40](#_Toc515837402)

[**Lesson 6 中間テスト２** 41](#_Toc515837403)

[**6.5 処理の流れの変更** 42](#_Toc515837404)

[**Lesson6 章末テスト** 44](#_Toc515837405)

[**Lesson 9 配列** 45](#_Toc515837406)

[**9.1 配列の基本** 45](#_Toc515837407)

[**9.2 配列の宣言** 47](#_Toc515837408)

[**9.3 配列の利用** 47](#_Toc515837409)

[**Lesson\_09 実習課題\_1** 47](#_Toc515837410)

[**課題 1** 47](#_Toc515837411)

[**課題２** 48](#_Toc515837412)

[**課題3** 48](#_Toc515837413)

[**課題４** 48](#_Toc515837414)

[**Lesson 9 中間テスト１** 48](#_Toc515837415)

# **前書き　Visual Studioの使い方**

## **プログラムの作成手順**

プロジェクトLesson0を作ってみましょう。大文字の入力はshiftキーを押しながらで行えます。

１. プロジェクトの作成

StartメニューからVisualStudioを起動する。



ファイル→新規作成→プロジェクトを選択



ドキュメントの下にc\_purapura\_1というフォルダを作って、そこにプロジェクトを作りましょう。\_(アンダースコア)はshift + /で入力できます。図１の赤枠を参照。

図１



2. プロジェクトのプロパティを変更する。

ソリューションエクスプローラーでプロジェクトを選択→右クリック→プロパティ



3.ソースファイルの追加

ソリューションエクスプローラーでプロジェクトを選択→右クリック→追加→新しい項目を選択。



4. ソースコードの入力

　追加したソースファイルを選択。



選択が出来たら、下記のようにコードを入力する。



次の４点に注意する。

①　スペースは半角。入力を忘れないように。図２，３の青枠。

　②　大文字、小文字に注意。

　③　#　、<　、（　、)、{ 、}、”(ダブルクォーテーション)などはShiftキーを押しながら入力する。図２，３の赤枠。

　④　;(セミコロン)を忘れない。上記コード、図２，３の緑枠。

⑤ 日本語入力→英字入力は半角/全角ボタンで切り替えることができる。図２の黄色枠。

**図２**



**図３**



5. ソースコードのビルド→実行

　Ctrl+F5でデバッガなしで実行できます。下記図の赤枠。



６. コマンドプロンプト(黒いウィンドウ)がすぐに消える

　　この現象が発生した場合は、2をやり直す。

## **・章末テスト**

下記のURLのテストを行いなさい。

<https://docs.google.com/forms/d/e/1FAIpQLSfWXuo1SqrWla4WEOIGEVUOJlqHg_vBP6p-5FZtuyqzRLNs5A/viewform?usp=sf_link>

# **Lesson 1 はじめの一歩**

## **1.1 C++のプログラム**

機械語(p.3)

　　→こんなの。



## **1.2 コードの入力**

統合環境を使ってみる(p.5)

本校の授業ではVisual Studio(統合環境)を使います。

テキストエディタにコードを入力する。(p.6)

　新しいプロジェクト、Lesson1を作成して、ソースファイルを追加して、p.7のSample1.cppの内容を入力する。

コンパイルエラーを起こしてみる

コードを書くことができて、実行出来たら、わざとコンパイルエラーを起こしてどうなるか確認してみましょう。

## **1.3プログラムの作成**

コンパイラを実行する(P9)

オブジェクトファイルを確認してみる。

オブジェクトファイルをリンクする(P9)

実行ファイルができていることを確認する。

## **1.Ex C言語の標準出力関数を知ろう(補足)**

printf関数

　coutはC++の標準出力関数ですが、printfはC言語の標準出力関数となります。C++はC言語のスーパセット言語となっており、C言語の機能はすべて使えます。では、下記のコードを入力してみてください。

|  |
| --- |
| #include <stdio.h>  int main()  {  printf(“ようこそ C++へ！”);  return 0; } |

C言語検定では標準出力関数として、printf関数が使用されていることと、ほかの言語でも似たようなprint関数が用意されてるので、こちらを使えるようになることも重要です。

## **Lesson 1 章末テスト**

下記のURLのテストを行いなさい。

<https://docs.google.com/forms/d/1NPbcFTvH37P5kJ3pHvOIB0GWXZoNYUG93aCOaEYAXYA/edit>

# **Lesson 2 C++の基本**

## **2.1 画面への出力**

新しいコードを入力する(p.16)

新しいプロジェクトLesson2を作成して、Sample.1.cppの内容を入力して実行してください。

色々な出力方法を知る(p.18)

Sample2.cppの内容を入力して実行してください。

改行には\nを使う

　テキストにも書かれていますが、重要です。しっかりと覚えましょう。

## **2.2 コードの内容**

main()関数

　main関数はエントリーポイントと呼ばれる特殊な関数です。プログラムのスタート地点となる関数です。

#include <iostream>をコメントアウトしてみる。

#include <iostream>をコメントアウトして、コンパイルしてみましょう。下記のようなエラーが表示されたと思います。



エラーをダブルクリックすると、エラーが起きている箇所にジャンプできます。試してみてください。

using namespace std;をコメントアウトしてみる。

こちらもコメントアウトしてコンパイルしてみてください。先ほどと同様のエラーが表示されたと思います。



重要

　エラーをダブルクリックするとコンパイルエラーが起きている箇所にジャンプできることをしっかりと覚えましょう。

## **2.3 文字と数値**

数値リテラル

　123、579、30.0など

文字リテラル

　‘A’、’B’、’c’など。**シングルクォーテーションで囲まれたもの。**一文字を表す。文字列とは違う！！！

文字列リテラル

　“ABC”、”ようこそC++”など。**ダブルクォーテーションで囲まれたもの。**

エスケープシーケンス

　全部を覚える必要はない。重要なのは下記の３点です。

　\n 改行コード

　\0 文字列の終わり(詳細はLesson9.7)

\マークをつけることで、\や’や”などの特殊文字を文字列に組み込むことができる。

　　→どういうこと？

例えば、下記のような文字列を表示する場合を考えてみましょう。



ダブルクォーテーションは文字列リテラルを囲むものという、特殊文字となっているため、下記のようなプログラムではコンパイルエラーになります。



文字としてダブルクォーテーションを出力したい場合は、下記のように記述する必要があります。



その他のエスケープシーケンスが必要になったら、ネットで検索をすればＯＫです。ググりましょう。

実習1 (時間 5分)

　一度のcoutの実行で下記のような表示ができるようにしてみよう。



printf関数でも同じ

　エスケープシーケンスのルールはprintf関数でも同じです。



実習２(時間 5分)

一度のprintf関数の実行で下記のような表示ができるようにしてみよう。



８進数

10を8とする表記法。

16進数

10を16とする表記法

## **Lesson 2 章末テスト**

下記のURLのテストを行いなさい。

<https://docs.google.com/forms/d/e/1FAIpQLSdNjkkwZ4M0L5_ujSSc-GULWBWXiA_dRPSHZ8ct2XpUrocsjg/viewform?usp=sf_link>

# **Lesson 3 変数**

## **3.3 型**

数が多くて、いきなり覚えるのは難しいと思いますが、今は下記の4点を覚えてください。

　整数型　 　　　　int

浮動小数点型 　　　　float

文字型 　　　　char

unsignedをつけると　 符号なしになる。(頭の隅の方にでも置いておいてください。)

|  |
| --- |
| int hoge = －10; //hogeは符号付き整数型。負数も記憶できる。  unsigned int hoge2 = 20; //hoge2は符号なし整数型。負数は記憶できない。 |

## **3.5 変数の利用**

変数に値を代入する

|  |
| --- |
| int num;  num = 3; |

＝が代入であることに注意してください。数学であればイコールですが、プログラムでは代入です。右辺の値を左辺に代入します。簡単に思えるかもしれませんが、実は、ここの理解で躓く人が多いです。

p.50のコードを入力して結果を確認しよう

　新しいプロジェクトLesson3を作成して、p.50のコードを入力して確認しましょう。

変数を初期化する

　変数は宣言することで、数値を記録するための領域がメモリ上に確保されます。では下記のようなコードの場合、どのような値が表示されるのでしょうか？

|  |
| --- |
| int num; //numという変数を用意する。  cout << num << “\n”; 何が表示される？ |

変数の値を変更する

　p.53のコードを入力して、動作を確認しましょう。

他の変数の値を代入する

　p.55のコードを入力して、動作を確認しましょう。

値の代入についての注意

　p.56のコードを入力して、動作を確認しましょう。

　doubleはfloatでもＯＫです。

## **3.5ex(補足) printf関数を使用して、変数の値を出力する。**

coutを使用して変数の中身を出力する方法は見てきましたが、printf関数も変数の値を出力することができます。int型の変数の値を出力する場合は下記のように記述します。



また、下記のように記述することで、複数の変数の値を出力できます。



## **Lesson 3 中間テスト1**

下記のURLのテストを行いなさい。

<https://docs.google.com/forms/d/e/1FAIpQLScDqY-iHav90m71Sr1uTCXsHrTW1t5TJLkW1oNA0xNZUAtfmg/viewform?usp=sf_link>

## **3.6 キーボードからの入力**

キーボードから入力する

最近使ったプロジェクトからLesson3を起動して、P.60のSample5.cppの内容を打ち込んで下さい。



C++標準入力関数 cin

cinを実行するとキーボードからの入力待ちになります。

二つ以上の数値を入力する

　p.61のSample6の内容を打ち込んでください。

## **3.6.ex(補足) C言語の標準入力関数scanf\_s**

C++の標準入力関数はcinですが、C言語にも同様の関数のscanf\_sがあります。下記のようなコードを入力することで、cinと同じ動作になります。入力して確認してください。



(注意)正確には、C言語の標準入力関数はscanf関数なのですが、この関数はセキュリティホールが存在しているため、マイクロソフトがより安全なscanf\_s関数を用意しています。使い方はscanfと同じなので、混乱しないようにしてください。

## **3.7 定数**

P.63のSample7.cppのコードを入力してみてください。

変更することのできない変数にconstをつける。

プログラマのミスで値を変更してしまうヒューマンエラーをなくす。

今、定数の利点を理解するのは難しいと思うので、これは後期に詳しくやります。

## **Lesson 3 章末テスト**

下記のURLのテストを行いなさい。

<https://docs.google.com/forms/d/e/1FAIpQLSfgUfWrA1P6vhlj-UqzQr3CBhHYmOxpzsFYOo0mdsmuQpSU9A/viewform?usp=sf_link>

# **Lesson 4 式と演算子**

## **4.1 式と演算子**

式の値を出力する(p.71)

新しいプロジェクトLesson4を作成して、Sample1.cppを入力して実行してください。

プログラムの世界では＊が掛け算になります。＊はshift＋：、と+はshift＋；で入力できます。下記の図の赤枠を参照。



色々な演算をする(p.71)

下記の２点をしっかりと意識しながら、Sample2.cppのコードを入力してください。

・ 変数と変数の演算を行える。

* 変数と数値リテラルとの演算を行える。

変数num1の値に１を足し、その値を再度num1に代入する(p.73)

　下記のコードに注目してください。

|  |
| --- |
| num1 = num1 + 1; |

数学的にはおかしな式です。 = の記号が等しいではなく、代入であったことを思い出してください。右辺の結果(num1+1)を左辺に代入しているため、このような記述が可能になります。

キーボードから入力した値を足し算する(p.74)

Sample3.cppの内容を入力して、実行してください。

実習 1(10分)

Lesson4の内容を改造して、下記のような表示をできるようにしなさい。

また、除算の演算子は / です。



## **Lesson4 中間テスト１**

下記のURLのテストを行いなさい。

<https://docs.google.com/forms/d/e/1FAIpQLSfyMQIL4RSN9mjOLdaeRnAJv2mDBi4U-WjkY4mywBdxOnw-uQ/viewform?usp=sf_link>

## **4.2 演算子の種類**

いろいろな演算子(p.76)

非常に多くの演算子があります。残念なことに、ここに記載されている演算子はすべて重要です。世のプログラマーはすべてを覚えていると思います。でも、安心してください。覚えようとしなくてもプログラムを書いていると、自然と覚えてしまいます。

　まずは四則演算(＋－×÷)+αをマスターしましょう。

では、Lesson4を最近使ったプロジェクトから開いて、p.77のSample4.cppのコードを入力して、実行してください。

実習2(10分)

Lesson4を改造して、次の動画のような挙動になるプログラムを作成しなさい。

出席番号で０～４のグループに振り分けるプログラム。

<https://www.youtube.com/watch?v=lRwpa-nbEuI&feature=youtu.be>

インクリメント・デクリメント演算子(p.79)

これも四則演算子なのですが、少し変わっています。よく使う演算子なのでしっかりと覚えましょう。

インクリメント・デクリメントの前置と後置(p.80)

前置と後置で実行結果が変わることがあります。普段プログラムを書くときは、この規則を意識しなくてもいいように書く方が優れている場合がほとんどですが、資格・検定の試験でこれを問う問題がでることがあるので、覚えましょう。

Lesson4にp.81のSample.cppのコードを入力して、実行してください。

下記が前置と後置の挙動の違いの覚え方です。

**・前置なので、代入する前にインクリメント・デクリメントする。**

**・後置なので、代入した後でインクリメント・デクリメントする。**

ただし、これに依存するようなコードを書くことは可読性を下げることになるので、下記のようなコードを書くことを推奨します。

代入する前にインクリメントしたい場合

|  |
| --- |
| a++; 　//インクリメントしてから  b = a; //代入する。 |

代入した後でインクリメントしたい場合

|  |
| --- |
| b = a; //代入してから、  a++; //インクリメントする。 |

## **Lesson 4 中間テスト２**

下記のURLのテストを行いなさい。

<https://docs.google.com/forms/d/e/1FAIpQLSdAUAv0FP5S5zUoNbznYQUhUVoGBl82IVA6wdh3LZrc0I-oRA/viewform?usp=sf_link>

代入演算子(p.83)

Lesson4にp.85のSample6.cppを入力して実行してください。

sizeof演算子

　この演算子は、配列(Lesson9)の要素数を調べるときなどに使用されることがあります。国家試験でもよく出てくる演算子です。例えば下記のように使います。

|  |
| --- |
| int num = sizeof(int); |

この演算子は読んで字のごとく、「size of int」int型 の(of) サイズ(size)を求めてくれます。

では、int型のサイズはいくつだったでしょうか？教科書のp.43に戻って確認してみましょう。

確認テスト

　下記のプログラムの実行結果を答えなさい。

|  |
| --- |
| int main()  {  int hoge = 0;  int size = sizeof(hoge);  cout << size << “\n”;  return 0;  } |

答え

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |
| --- |
| int main()  {  char hoge = ‘w’;  int size = sizeof(hoge);  cout << size << “\n”;  return 0;  } |

答え

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4Hands-On 2

Lesson 4に下記のコードを入力して動作を確認しなさい。



シフト演算子(p.88)

下記の２点を覚える。

　・１ビット左にシフトすると値は倍になる。

　・１ビット右にシフトすると値は半分になる。

Hands-On 3

Lesson4に下記のコードを入力して動作を確認しなさい。



## **Lesson 4 中間テスト3**

下記のURLのテストを行いなさい。

<https://docs.google.com/forms/d/e/1FAIpQLScoCxdw4wk-kCYEglROeYF9pFD-tRO5MOZj9S-LSrc4pMY2Jw/viewform?usp=sf_link>

## **4.3 演算子の優先順位**

演算子の優先順位とは(P.90)

下記の２点を抑えましょう。

**・四則演算の優先順位は数学と同じ。**

**・優先順位が分からなかったら()を使えば良い。**

この２点を押さえておけば、p.91～p92の表は覚える必要はありません。この表を答えられるプログラマーなどいないのですから。

同じ優先順位の演算子を使う(P93

これも難しいことを考える必要はありません。抑える点は同じです。

**・四則演算の優先順位は数学と同じ。**

**・優先順位が分からなかったら()を使えば良い。**

## **4.4 型変換**

大きなサイズの型に代入する(P.94)

Lesson4にSample8.cppの内容を入力して実行してください。

小さなサイズの型に代入する(P.95)

小さなサイズの型に代入すると、値が失われるのは正しいですが、このテキストに書かれている内容は、ちょっと不正確です。このSample9は正しくは、**浮動少数点型の変数の値を整数型の変数に代入すると、小数点の値が失われる**です。

Lesson4に下記のコードを入力して実行してください。

|  |
| --- |
| #include <iostream>  using namespace std;  int main()  {  float fnum = 50.5f;　 //floatは32bitの浮動小数点。  int inum = fnum; 　　 //intも32bitなので、値は失われないはず？  cout << inum << '\n'; //教科書の説明は正しくない。小数点は失われる！  return 0;  } |

では、「小さなサイズの型に代入すると値が失われる」の正しい説明となるコードを見てみましょう。

Lesson4に下記のコードを入力して実行してください。

|  |
| --- |
| #include <iostream>  using namespace std;  int main()  {    int inum = 100000; //intは32bitは符号付きの整数型。  //表現できる範囲は、-2,147,483,648 ～ 2,147,483,647  short snum; 　　　　　　//shortは16bitの符号付きの整数型。  //表現できる値の範囲は、-32,768 ～ 32,767  snum = inum; //snumに100000という値は表現できないので、代入すると・・・  cout << snum << '\n'; //値が失われる！  return 0;  } |

キャスト演算子を使う(P.96)

明示的な型変換を行う。

→教科書に書かれていることをしたところで、結果は何も変わらない。

　→え？じゃぁキャストってなんのためにするの？

型変換を行うと、小さなサイズの型に代入すると値が失われてしまいます。つまり下記のようなコードを書いてしまった場合、致命的な不具合を生み出すことがあります。

(このコードは書かなくていい)

|  |
| --- |
| #include <iostream>  using namespace std;  int g\_playerHp = 100000; //int型のプレイヤーのHPを表すグローバル変数。  int main()  {  short playerHp = g\_playerHp; //プレイヤーのHPをshort型のローカル変数に代入！  playerHp -= 100; //プレイヤーのHPを100減らすことが目的なのだが・・・。  g\_playerHp = playerHp; //変更したHPを書き戻す。  cout << g\_playerHp << '\n'; //なんてこった。  return 0;  } |

このようなヒューマンエラーを防ぐために、明示的ではない、小さな型への変換を行うコードを書いた場合にコンパイルエラーにすることができます。

次の設定を行ってから、コンパイルを行ってみてください。

プロジェクトのプロパティを開く



プロパティページ/C++/全般/警告レベルを4にする。



プロパティページ/C++/全般/警告をエラーとして扱うを「はい」にする。



この設定でLesson4のコンパイルを行うと、下記のようなエラーと警告が出てくると思います。



こうすることで、意図していない型変換を行うコードを書いてしまった場合はコンパイルエラーとなって、ヒューマンエラーをなくすことができます。そして、このエラーは下記のように明示的にキャストすることによって、消すことができます。

|  |
| --- |
| #include <iostream>  using namespace std;  int main()  {  int inum = 100000; //intは32bitは符号付きの整数型。  //表現できる範囲は、-2,147,483,648 ～ 2,147,483,647  short snum; //shortは16bitの符号付きの整数型。  //表現できる値の範囲は、-32,768 ～ 32,767  snum = **(short)**inum; //snumに100000という値は表現できないので、代入すると・・・  cout << snum << '\n'; //値が失われる！  return 0;  } |

つまり、キャストというのは「小さな型変換で、値が失われるのは知ってるけど、これは正しいコードだから、黙ってコンパイルしろ！」ということをコンパイラーに教えてやる行為となります。

## **Lesson 4 中間テスト４**

　下記のURLのテストを行いなさい。

<https://docs.google.com/forms/d/e/1FAIpQLScNMpAM-h4_Bek28g8_-tVNiXn8Rj-bos_NAHbE0QlagOpyTA/viewform?usp=sf_link>

異なる型どうしで演算する(P.98)

この説明もちょっと正しくありません。例えば下記のコードの場合、結果は小さい型の浮動小数点となります。

|  |
| --- |
| #include <iostream>  using namespace std;  int main()  {  long long llValue = 10; //long longは64bitの整数型。  float fValue = 0.5f; //floatは32bitの浮動小数点型。  long long llResult = llValue \* fValue; //教科書の説明なら64bitの整数型になるはずが、  　 //32bitの小数点型になっている！！！  return 0;  } |

型変換のルールは下記です。

　・どちらか一方のオペランドが浮動所数点なら、結果は浮動小数点になる。

　・オペランドが両方とも整数型　or　浮動小数点型なら、型サイズの大きい方になる。

## **Lesson 4 章末テスト**

　下記のURLのテストを行いなさい。

<https://docs.google.com/forms/d/e/1FAIpQLSeDBBADoB0WS-1l8sT8foEhJoobCE0-_bRu_UVdXioWAZqWgA/viewform?usp=sf_link>

# **Lesson 5 場合に応じた処理**

## **5.1　関係演算子と条件**

条件の仕組みをしる(p.106)

例えばゲームであれば、「**もしも**、コントローラーのＡボタンが押されたら」

　　　　　　　　　　　　　　→ジャンプする

条件を記述する(p.107)

　C++では==が数学の＝と同じになる。左辺と右辺が等しければtrueを返す演算子。

関係演算子は、条件が成立する時にtrue、成立しなければfalseを返してくる演算子です。

## **5.2　if文**

if文の仕組みを知る(p.111)

新しいプロジェクトLesson5を作成して、Sample1.cppの内容を入力して、実行してください。

if文で複数の分を処理する(p.113)

Lesson5を改造して、Sample2.cppのコードを入力して、実行してください。

ブロックにしないと(p.116)

if文を使うときは、条件を満たすときの処理が、たとえ１文であったとしても、必ず{}で囲むのをお勧めします。ただし、{}で囲んでいないコードを読むこともあるので、知識として知っておくことは重要です。

## **5.2 実習**

下記の動画のようなプログラムを実装しなさい。

<https://www.youtube.com/watch?v=EM0hp-jT15I&feature=youtu.be>

## **5.3 if～else文**

Lesson5を改造して、Sample3.cppの内容を入力して、実行してください。

## **5.3 実習**

下記の動画のようなプログラムを実装しなさい。

<https://www.youtube.com/watch?v=qEYbe7RDEkg&feature=youtu.be>

## **5.4 if～else if～else**

if～else if～elseの仕組みを知る(p.122)

Lesson5を改造して、Sample4.cppを入力して実行してください。

## **5.4 実習**

下記の仕様を満たすプログラムを実装しなさい。

・年齢の入力を促す。

・20歳未満なら、「未成年ですね」と表示する。

・20歳以上なら「成人ですね」と表示する。

・ただし、下記の年齢の場合は長寿の祝いを表示する。

60歳の場合は「還暦おめでとうございます。」

77歳の場合は「喜寿おめでとうございます。」

88歳の場合は「米寿おめでとうございます。」

99歳の場合は「白寿おめでとうございます。」

下記の動画を参考にして実装しなさい。

<https://www.youtube.com/watch?v=JmguNYN1aaU&feature=youtu.be>

## **5.ex\_1**

if文はbool型のtrue、falseの値によって分岐します。そのため、下記のようなコードも合法です。

|  |
| --- |
| #include <iostream>  using namespace std;  int main()  {  int age;  cout << "年齢を入力してください。\n";  cin >> age;  bool result = age >= 20; //>=のような関係演算子はbool型の値を返す。  if (result){ //if文は単にbool型の値によって分岐するだけなので、これもＯＫ。  cout << "あなたは成人ですね。\n";  }    return 0;  } |

また、整数の０はfalseに０以外はtrueに暗黙的に変換されるため、下記のようなコードも合法です。

|  |
| --- |
| #include <iostream>  using namespace std;  int main()  {  int age;  cout << "年齢を入力してください。\n";  cin >> age;  //int型の変数ageをif文にそのまま書く。  //整数型は暗黙的に、bool型に変換される。  if (age){  cout << "あなたは0歳ではないですね。\n";  }  else {  cout << "あなたは0歳ですね。\n";  }  return 0;  } |

！(否定演算子)を使うと、bool型の結果が反転します。

|  |
| --- |
| #include <iostream>  using namespace std;  int main()  {  int age;  cout << "年齢を入力してください。\n";  cin >> age;  bool result = age >= 20; //>=のような関係演算子はbool型の値を返す。  　　　//resultの結果を反転しているので、このif文の条件が成立するということは、  //未成年だということになる。  if (!result){  cout << "あなたは未成年ですね。\n";  }  return 0;  } |

下記のようにも書ける。

|  |
| --- |
| #include <iostream>  using namespace std;  int main()  {  int age;  cout << "年齢を入力してください。\n";  cin >> age;  　　　//age>=20の結果を反転しているので、このif文の条件が成立するということは、  //未成年だということになる。  if (!(age >= 20)){  cout << "あなたは未成年ですね。\n";  }  return 0;  } |

## **Lesson 5 中間テスト１**

下記のURLのテストを行いなさい。

<https://docs.google.com/forms/d/e/1FAIpQLSfu0l2a0RE_FFheu8jhl6mmcJVzDEyxFF6kIhvOvKfVe5RE6w/viewform?usp=sf_link>

## **5.5 switch文**

switch文の仕組みを知る(p.126)

switch文で出来ることは、if～else if ～else文でもできるので、どちらを使ってもＯＫ。

Lesson5を改造して、Sample5.cppを入力して実行してみてください。

break文が抜けていると(p.129)

break文がないと下のケースが実行されます。このようなことを意図して行うコードもありますが、慣れないうちは必ずbreakを書くようにしましょう。

## **5.6　論理演算子**

論理演算子の仕組みを知る(p.131)

(条件Ａ)　&& (条件Ｂ)

　　→条件Ａがtrue **かつ** 条件Ｂがtrue

(条件Ａ)　|| (条件Ｂ)

　　→条件Ａがtrue **または** 条件Ｂがtrue

では、&&(論理積)を使えるケースを見てみましょう。

例) 20歳以上かつ男性かどうかを判断する場合の条件式(論理演算を使わない場合)

|  |
| --- |
| #include <iostream>  using namespace std;  int main()  {  int age, gender;  cout << "年齢を入力してください。\n";  cin >> age;  cout << "性別を入力してください。0 : 男性、1 : 女性\n";  cin >> gender;  //&&はif文のネストでも表現できるが・・・  if (age >= 20) {  if (gender == 0) {  cout << "あなたは２０歳以上で、男性ですね。\n";  }  }    return 0;  } |

例) 20歳以上かつ男性かどうかを判断する場合の条件式(論理演算を使う場合)

|  |
| --- |
| #include <iostream>  using namespace std;  int main()  {  int age, gender;  cout << "年齢を入力してください。\n";  cin >> age;  cout << "性別を入力してください。0 : 男性、1 : 女性\n";  cin >> gender;  //&&を使って、条件をまとめることができる。  if (age >= 20 && gender == 0) {  cout << "あなたは２０歳以上で、男性ですね。\n";  }    return 0;  } |

続いて、||(論理和)を使えるケースを見てみましょう。

例)飲食店で女性と子供は2割引のサービスを行っている場合の条件式(論理演算を使わない場合。)

|  |
| --- |
| #include <iostream>  using namespace std;  int main()  {  //例えば、飲食店で女性と子供は2割引のサービスを行っている場合。  int age, gender;  cout << "年齢を入力してください。\n";  cin >> age;  cout << "性別を入力してください。0 : 男性、1 : 女性\n";  cin >> gender;  //||は複数のif文でも表現できるが・・・  if (age < 13) {  cout << "あなたは２割引きのサービスを受けられます。\n";  }  if (gender == 1) {  cout << "あなたは２割引きのサービスを受けられます。\n";  }  return 0;  } |

例)飲食店で女性と子供は2割引のサービスを行っている場合の条件式(論理演算を使う場合。)

|  |
| --- |
| #include <iostream>  using namespace std;  int main()  {  //例えば、飲食店で女性と子供は2割引のサービスを行っている場合。  int age, gender;  cout << "年齢を入力してください。\n";  cin >> age;  cout << "性別を入力してください。0 : 男性、1 : 女性\n";  cin >> gender;  //||を使って、条件をまとめる。  if (age < 13 || gender == 1) {  cout << "あなたは２割引きのサービスを受けられます。\n";  }    return 0;  } |

更に論理演算を組みあわせると、じゃんけんの勝敗判定も行えます。

|  |
| --- |
| //playerという変数にプレイヤーの手、comという変数にコンピュータの手の情報が入っている。  //0がグー、１がチョキ、２がパーです。  if (( player == 0 && com == 1 ) //プレイヤーがグー、コンピュータがチョキ。  || ( player == 1 && com == 2 ) //プレイヤーがチョキ、コンピュータがパー。  || ( player == 2 && com == 0 ) //プレイヤーがパー、コンピュータがグー。  ) {  cout << "あなたの勝ちです。\n";  } |

## **Lesson 5 章末テスト**

下記のURLのテストを行いなさい。

<https://docs.google.com/forms/d/e/1FAIpQLScWLHDqPBaJc3ISuLyL13M93NhYJJwO010PPaCld7PK5EUltw/viewform?usp=sf_link>

# **Lesson 6 何度も繰り返す**

## **Lesson6を始める前に**

Lesson 6に記載されているサンプルコードですが、今後の勉強に悪い影響を与えかねない要素があるので、教科書のサンプルコードは打ち込まずに、PDFファイルのサンプルコードを打ち込んでください。この理由はLesson9で配列を勉強する時に説明ます。また、ループの繰り返しの処理は、for文、while文、do~while文の３つがありますが、for文だけ覚えてもらえばＯＫです。while文、do～while文で出来ることは、すべてfor文で実現可能なので、プログラムに慣れてきたときに覚えてもらえれば十分です。ただし、国家試験、検定にはすべて出てくる可能性があるので、国家試験を取りたい人は、すべて覚える必要があります。

## **6.1 for文**

for分の仕組みを知る(p.142)

VisualStudioでLesson\_6のプロジェクトを作成して、Sample1.cppを入力してください。

for文は一行に３つの処理が記述されていることに注意！

for( int i = 0 ; i < 5; i++)

ループブロックの処理を一度、実行した後で行われる処理。

ループ変数 iの宣言と初期化

ループの継続条件

Sample1.cpp

|  |
| --- |
| #include <iostream>  using namespace std;  int main()  {  for (int i = 0; i < 5; i++) {  cout << "繰り返しています。\n";  }  cout << "繰り返しが終わりました。\n";  return 0;  } |

もう少し、見ていきましょう。下記のコードを見てください。

Sample1.cppは下記のようなコードと同義です。

|  |
| --- |
| #include <iostream>  using namespace std;  int main()  {  int i = 0; //変数iの宣言と初期化。  LOOP: //これはラベルと呼ばれるもの。  if (i < 5) { //ループの継続判定。  cout << "繰り返しています。\n";  i++; //ループ変数のインクリメント。  goto LOOP; //goto文でラベルLOOPにジャンプする。  }  cout << "繰り返しが終わりました。\n";  return 0;  } |

**注意！今回for文の説明のためにgoto文を使用しましたが、goto文はスパゲッティコード(読みづらいコード)を生み出しやすくするものとして、多くの開発で使用することが非推奨となっている構文です。絶対に使用しないように！！！**

for文の３つの式の意味が分かれば、下記のようなコードが書けることが分かります。

下記のコードを入力して、F10キーでステップ実行を行い、処理の流れを確認してください。

|  |
| --- |
| #include <iostream>  using namespace std;  int main()  {  //ループ変数は0以外で初期化することもできるし、  //ループの継続判定もi>=0などにもできる。  //ループブロックの処理が終わった後の処理もデクリメントでも良い。  for (int i = 5; i >= 0; i--) {  //ループ変数はループ内で使用できる。  cout << i << "\n";  }  return 0;  } |

このコードをgoto文を使って書くと下記のようになります。

|  |
| --- |
| #include <iostream>  using namespace std;  int main()  {  int i = 5; //変数iの宣言と初期化。  LOOP: //これはラベルと呼ばれるもの。  if (i >= 0) { //ループの継続判定。  cout << i << "\n";  i--; //ループ変数のインクリメント。  goto LOOP; //goto文でラベルLOOPにジャンプする。  }    return 0;  } |

変数をループ内で使う(p.145)

ここまでの例で見てきたように、ループ変数をループ内で使うことができます。Lesson\_6にSample2.cppを入力して実行してください。

for文を応用する(p.146)

Lesson\_6にSample3.cppとSample4.cppを入力して、動作を確認してください。

Sample3.cpp（入力した数だけ\*を表示する。）

|  |
| --- |
| #include <iostream>  ループの終了判定に変数が使えることに注目！  using namespace std;  int main()  {  int num;  cout << "いくつ\*を表示しますか？\n";  cin >> num; //数を入力させる。    for (int i = 0; i < num; i++) {  cout << "\*"; //入力した数だけ\*を繰り返し表示する。  }  cout << "\n";  return 0;  } |

Sample4.cpp (入力した数までの合計を求める。)

|  |
| --- |
| #include <iostream>  using namespace std;  int main()  {  int kazu;  int goukei = 0;    cout << "いくつまでの合計を求めますか？\n";  cin >> kazu; //数を入力させる。  for (int i = 1; i <= kazu; i++) {  goukei += i ; //例えば、kazuに5が入力されたら、iの値は１～５となる。  　　//なので、iの値を加算していくと1～5までの数値の合計になる。  }  cout << "１から" << kazu << "までの合計値は" << goukei << "です。\n";  return 0;  } |

## **Lesson 6 中間テスト１**

下記のＵＲＬのテストを行いなさい。

<https://docs.google.com/forms/d/e/1FAIpQLScAPHvnMY3l2-PuMblKSBgfP0XIBRcexBqBHIzURobXUUCUFA/viewform?usp=sf_link>

## **6.2 while文**

while文の仕組みを知る(p.149)

while文は、for文をgoto文に置き換えて記述したものとよく似ています。

Sample5.cppを入力して実行出来たら、F10を押して、ステップ実行をおこなって動作を確認しなさい。

Sample5.cpp

|  |
| --- |
| #include <iostream>  using namespace std;  int main()  {  int i = 0; //これがループ変数。  while (i < 5) { //これがループの継続条件。  cout << i << "番目の繰り返しです。\n";  i++; //これがループの処理が終わったときの処理。  }  cout << "繰り返しが終わりました。\n":  return 0;  } |

条件の記述を省略する(p.151)

p.152のようなコードを書くときはforではなく、whileが使われることが多いです(私もたぶんwhileを使います)。whileが使われる理由は、このようなコードを書くときは、for文に比べて、キータイプの量が減ることと(キータイプ量が減ると腱鞘炎に苦しむプログラマーが減りますよね？)、whileを覚えている人からすると、読みやすいコードになるからです。

　では、Sample6.cppを記入して、動作を確認してください。

Sample6.cpp

|  |
| --- |
| #include <iostream>  using namespace std;  int main()  {  int num = 1;  while (num == 1) { //numが1の間は処理を繰り返す。  cout << "整数を入力してください。(0で終了)\n";  cin >> num;  cout << num << "が入力されました。\n";  }  cout << "繰り返しが終わりました。\n";  return 0;  } |

## **6.3　do～while文**

　たぶん学生のうちに使うことはないと思います。do～while文は下記の一点だけを覚えておきましょう。

**ループの中の処理が必ず最低一回は行われる。**

## **6.4　文のネスト**

文をネストする(p.157)

Sample8.cppを入力して動作を確認したら、F10を押してステップ実行で処理の流れを確認しなさい。

Sample8.cpp

ループ変数がiとjで違うことに注意！

|  |
| --- |
| #include <iostream>  using namespace std;  int main()  {  for (int i = 0; i < 5; i++) {  for (int j = 0; j < 3; j++) {  cout << "iは" << i << "、jは" << j << "\n";  }  }  return 0;  } |

## **6.4 ex デバッガの機能のウォッチを使ってみよう。**

プログラムを書いていくうえで、強力な武器となるデバッガの機能のウォッチ(変数の値を見るため)を使って、6.4のループ変数のiとjの中身を見てみましょう。ウォッチの使い方は下記の動画を参考にしてみてください。

<https://www.youtube.com/watch?v=dAwYii65J40&feature=youtu.be>

if文などと組み合わせる(‘p.159)

Sample9.cppを入力して動作を確認してください。動作が確認出来たら、下記の動画のように、ステップ実行を行い、変数i、j、chをウォッチに追加して、処理の流れを追いかけなさい。

Sample9.cpp

|  |
| --- |
| #include <iostream>  using namespace std;  int main()  {  int ch = 0;  for (int i = 0; i < 5; i++) {  for (int j = 0; j < 5; j++) {  if (ch == 0) {  cout << '\*'; //\*を出力したら、次は-を表示  ch = 1; //するように、chに1を代入。  }  else {  cout << '-'; //\*を出力したら、次は\*を表示  ch = 0; //するように、chに0を代入。  }  }  cout << "\n"; //内側のループが終わったら改行します。  }  return 0;  } |

動画(音声が入っているので再生する時は注意してください。)

<https://www.youtube.com/watch?v=MPCf2NUSCUA&feature=youtu.be>

## **Lesson 6 中間テスト２**

　下記のURLのテストを行いなさい。

<https://docs.google.com/forms/d/e/1FAIpQLSfPy-snsI_VVDKWYV18xdYtVq171je48_McWFPbH9udXML3sQ/viewform?usp=sf_link>

## **6.5 処理の流れの変更**

break文の仕組みを知る(p.161)

Sample10.cppを入力して、動作を確認したら、F10キーでステップ実行して、処理の流れを確認しなさい。

Sample10.cpp

|  |
| --- |
| #include <iostream>  using namespace std;  int main()  {  int res;  cout << "何番目でループを注視しますか？(1～10)\n";  for (int i = 0; i < 10; i++) {  cout << i << "番目の処理です。\n";  if (i == res) {  break; //指定した回数で繰り返しを終了します。  }  }  } |

break文は、例えば下記のような仕様を実装する時などに使えます。RPGの製作を行っていて、一緒に戦う仲間のAIを作っている場合を考えて下さい。「ヒーラーのAIであれば、HPが500以下になった味方にヒーリングを行う。」といった仕様を実装することがあるかもしれません。そのような場合に、下記のコードのように味方からＨＰ500以下のキャラクターを検索する必要があります。

|  |
| --- |
| for (int i = 0; i < mikataCharacterNum; i++) {  if (mikataCharacterHP[i] < 500) {  //500以下のキャラクターを発見したので、ヒールをかけて  //ループを抜ける。  ・  ・  省略  ・  ・  break;  }  } |

繰り返しをネストしている場合、その内側の文でbreak文を使うと、外側のブロックに処理が移る(p.162)

これは下記のようなコードの話です。入力して動作を確認したのち、F10でステップ実行を行い、処理の流れを確認してください。

内側のループを抜ける。

|  |
| --- |
| #include <iostream>  using namespace std;  int main()  {  for (int i = 0; i < 3; i++) {  for (int j = 0; j < 5; j++) {  if (j == 3) {  break;  }  //jは2までしか出力されない！  cout << "i : " << i << "、j : " << j << "\n";  }  }  return 0;  } |

continue分の仕組みを知る(p.165)

Sample12.cppの内容を入力して動作を確認してください。

|  |
| --- |
| #include <iostream>  using namespace std;  int main()  {  int res;  cout << "何番目の処理を飛ばしますか？(0～9)\n";  cin >> res;  for (int i = 0; i < 10; i++) {  if (i == res) {  continue;  }  cout << i << "番目の処理です。\n";  }  return 0;  } |

continue文は下記のような仕様を実装したい場合に使えます。

「1000人のグループの名簿から20歳の人にだけ、成人おめでとうと表示する。」

この仕様をcontinue文を使って、実装すると下記のようになります。

|  |
| --- |
| for (int i = 0; i < 1000; i++) {  if (ageList[i] != 20) {  //20歳以外はスキップ。  continue;  }  cout << "成人おめでとう。\n";  } |

ただし、このコードは下記のように書くこともできます。

|  |
| --- |
| for (int i = 0; i < 1000; i++) {  if (ageList[i] == 20) {  cout << "成人おめでとう。\n";  }    } |

このように、continue文を使用しなくても、ほかの方法があるため、今無理して覚える必要はないと思います。

## **Lesson6 章末テスト**

下記のURLのテストを行いなさい。

<https://docs.google.com/forms/d/e/1FAIpQLSe9Qi4I82H6LzhIYxmw-wPiGth-ihf3PCZ0zmdcEPjd4MANgg/viewform?usp=sf_link>

# **Lesson 9 配列**

## **9.1 配列の基本**

配列の仕組みを知る(p.256)

「ただし、こうしてたくさんの変数が登場すると、コードが複雑で読みにくくなってしまう場合があります。(p.256)」

　→読みやすいコードを書くと言ことは、ソフトウェアを開発していくうえで、とても重要

要素となります。今のうちから意識してコードを書いていきましょう。

例えば、あなたがMMORPGの100人参加可能な、レイドバトルのボスキャラクターのAIを実装している場合を考えてみて下さい。そのボスキャラクターのAIは「HPが一番低いプレイヤーに対して攻撃を行う。」という仕様だったとします。配列を知らない不幸なプログラマー(プロのプログラマーに、そんな人はいないと思いますが)は、悪夢のようなコードを書いてしまうでしょう。

|  |
| --- |
| //プレイヤー構造体。  struct Player {  int hp;  int attack;  ・  ・  省略  ・  ・  };  //プレイヤー100人を表すグローバル変数を100個用意する。  Player player00;  Player player01;  Player player02;  Player player03;  Player player04;  Player player05;  Player player06;  Player player07;  Player player08;  Player player09;  Player player10;  ・  ・  省略  ・  ・  Player player98;  Player player99;  //攻撃対象となるプレイヤーを検索する関数。  Player\* FindAttackTargetPlayer()  {  Player\* attackTargetPlayer = nullptr;  //一番HPが小さい奴を探す。  int hp = 9999999; //とりあえずあり得ないくらい大きくしておく。40億超えたらダメだよ。  //プレイヤー00  if (player00.hp < hp) {  //プレイヤー00の方がHPが小さいので更新。  hp = player00.hp;  attackTargetPlayer = &player00;  }  //プレイヤー01  if (player01.hp < hp) {  //プレイヤー01の方がHPが小さいので更新。  hp = player01.hp;  attackTargetPlayer = &player01;  }  //プレイヤー02  if (player02.hp < hp) {  //プレイヤー02の方がHPが小さいので更新。  hp = player02.hp;  attackTargetPlayer = &player02;  }  //プレイヤー03  if (player03.hp < hp) {  //プレイヤー03の方がHPが小さいので更新。  hp = player03.hp;  attackTargetPlayer = &player03;  }  //プレイヤー04  if (player04.hp < hp) {  //プレイヤー04の方がHPが小さいので更新。  hp = player04.hp;  attackTargetPlayer = &player04;  }  //プレイヤー05  if (player05.hp < hp) {  //プレイヤー05の方がHPが小さいので更新。  hp = player05.hp;  attackTargetPlayer = &player05;  }  //プレイヤー06  if (player06.hp < hp) {  //プレイヤー06の方がHPが小さいので更新。  hp = player06.hp;  attackTargetPlayer = &player06;  }  //プレイヤー07  if (player07.hp < hp) {  //プレイヤー07の方がHPが小さいので更新。  hp = player07.hp;  attackTargetPlayer = &player07;  }  ・  ・  省略  ・  ・  //プレイヤー98  if (player98.hp < hp) {  //プレイヤー98の方がHPが小さいので更新。  hp = player98.hp;  attackTargetPlayer = &player98;  }  //プレイヤー99  if (player99.hp < hp) {  //プレイヤー99の方がHPが小さいので更新。  hp = player99.hp;  attackTargetPlayer = &player98;  }  //見つかった攻撃対象のプレイヤーを返す。  return attackTargetPlayer;  } |

もし、このプログラムを配列を知っていた幸福なプログラマーが書けばこうなるでしょう。

|  |
| --- |
| //プレイヤー構造体。  struct Player {  int hp;  int attack;  ・  ・  省略  ・  ・  };  //プレイヤー100人を表すグローバル変数を要素数100の配列として個用意する。  Player player[100];  //攻撃対象となるプレイヤーを検索する関数。  Player\* FindAttackTargetPlayer()  {  Player\* attackTargetPlayer = nullptr;  //一番HPが小さい奴を探す。  int hp = 9999999; //とりあえずあり得ないくらい大きくしておく。40億超えたらダメだよ。  for (int i = 0; i < 100; i++) {  if (player[i].hp < hp) {  //このプレイヤーの方がHPが小さいので更新する。  hp = player[i].hp;  attackTargetPlayer = &player[i];  }  }  //見つかった攻撃対象のプレイヤーを返す。  return attackTargetPlayer;  } |

とても短くシンプルで分かりやすいプログラムになりました。

## **9.2 配列の宣言**

**配列の添え字は0からはじまるので、最後の添え字は「要素数-1」にとなる。**

## **9.3 配列の利用**

　新しいプロジェクト、Lesson09を作成してSample1.cppを入力してください。

配列を初期化する(p.262)

# **Lesson\_09 実習課題\_1**

　下記のURLからzipファイルをダウンロードして、次の１～4の実習を行いなさい。

<https://1drv.ms/u/s!AnpsoGtakIkVi41brqVH5jmOnCTrVQ>

## **課題 1**

サンプルプログラムのkadai\_09\_01を改造して、試験の合計点を求めて表示するプログラムを作成しなさい。

## **課題２**

　サンプルプログラムのkadai\_09\_02を改造して、試験の平均点を求めて表示するプログラムを作成しなさい。

## **課題3**

　サンプルプログラムのkadai\_09\_03を改造して、試験の平均点以上の点数を取っている学生の人数を表示するプログラムを作成しなさい。

## **課題４**

　サンプルプログラムのkadai\_09\_04を改造して、試験の点数を昇順（小さい順）に表示できるようにしなさい。

課題４のヒント動画

　ヒント１　値の交換のアルゴリズムの解説と、ソースコードの解説

<https://www.youtube.com/watch?v=Y-nZ8dYaY-A&feature=youtu.be>

　ヒント２　バブルソートアルゴリズムの解説

<https://www.youtube.com/watch?v=xvbUuWzBOHY&feature=youtu.be>

　ヒント３　バブルソートのソースコード解説

<https://www.youtube.com/watch?v=gOmwRxBH8WM&feature=youtu.be>

# **Lesson 9 中間テスト１**

　下記のURLのテストを行いなさい。

<https://docs.google.com/forms/d/e/1FAIpQLSe-NfkV14Qi6lv8wgo6qYVaa8ODC3LK89Qy35QdTaVa_7juCA/viewform?usp=sf_link>

# **Lesson 7 配列**

## **7.1 関数の仕組みを知る**

　「C++でも、複雑なコードを書くようになってくると、たびたび行わなければならない一定の処理が出てくる場合があります。(P.170)」

例えば、これまでいくつかの関数をみなさんは使ってきています。

・コンソールに文字列を表示する処理

　printf関数、cout関数(実はこれも関数。ただしちょっと特殊な書き方をす

しているので注意が必要。)

・ユーザーからの入力を受け付ける関数。

scanf関数、cin関数(coutと同様に、これも関数。)

・キーボードの入力判定

GetAsyncKeyState関数。これはmicrosoftが提供している関数です。

・パックマン、落ち物などの文字表示関数。

kbcDrawMoji関数。この関数は私が作成した関数です。

もしこの関数という機能がなかったら、あなたはコンソールに文字を出力する時に下記のようなコードを毎回コピペで複製する羽目になります！！！

|  |
| --- |
| int main()  {  //Hello worldと表示する！  char\* fmt = "Hello world\n";  for (;;) {  unsigned long long ui;  long long i;  char \*s = NULL;  double d = 0.0;  int sign = 0;  int flags = 0;  int length = 0;  int precision = 0;  int tmp = 0;  INTEGER int\_type = 0;  while (\*fmt && \*fmt != '%') \_\_putc(\*fmt++);  if (\*fmt == '\0') { va\_end(ap); break; }  fmt++;  while (mystrchr("'-+ #0", \*fmt)) {  switch (\*fmt++) {  case '\'': flags |= THOUSAND\_GROUP; break;  case '-': flags |= LEFT\_JUSTIFIED; break;  case '+': flags |= WITH\_SIGN\_CHAR; sign = '+'; break;  case '#': flags |= ALTERNATIVE; break;  case '0': flags |= ZERO\_PADDING; break;  case ' ': flags |= WITH\_SIGN\_CHAR; sign = ' '; break;  }  }  if (\*fmt == '\*') { length = va\_arg(ap, int); fmt++; }  else { while (\_isnumc(\*fmt)) length = (length \* 10) + \_ctoi(\*fmt++); }  if (\*fmt == '.')  {  fmt++;  if (\*fmt == '\*') { fmt++; precision = va\_arg(ap, int); }  else { while (\_isnumc(\*fmt)) precision = precision \* 10 + \_ctoi(\*fmt++); }  }  while (mystrchr("hljzt", \*fmt)) {  switch (\*fmt++) {  case 'h': int\_type--; break;  case 'l': int\_type++; break;  case 'j': /\*intmax : long \*/  case 'z': /\*size : long \*/  case 't': /\*ptrdiff : long \*/  int\_type = L; break;  }  }  switch (\*fmt) {  case 'd':  case 'i':  i = get\_signed(ap, int\_type);  if (i < 0) { i = -i; sign = '-'; }  put\_integer(\_\_putc, i, 10, length, sign, flags);  break;  case 'u':  ui = get\_unsigned(ap, int\_type);  put\_integer(\_\_putc, ui, 10, length, sign, flags);  break;  case 'o':  ui = get\_unsigned(ap, int\_type);  put\_integer(\_\_putc, ui, 8, length, sign, flags);  break;  case 'p':  length = sizeof(long) \* 2;  int\_type = L;  sign = 0;  flags = ZERO\_PADDING | ALTERNATIVE;  case 'X':  flags |= CAPITAL\_LETTER;  case 'x':  ui = get\_unsigned(ap, int\_type);  put\_integer(\_\_putc, ui, 16, length, sign, flags);  break;  case 'c':  i = get\_signed(ap, C);  \_\_putc(i);  break;  case 's':  s = va\_arg(ap, char \*);  if (s == NULL) s = "(null)";  tmp = strlen(s);  if (precision && precision < tmp) tmp = precision;  length = length - tmp;  if (!(flags & LEFT\_JUSTIFIED))  {  while (length > 0) { length--; \_\_putc(' '); }  }  while (tmp--) { \_\_putc(\*s++); }  while (length-- > 0) { \_\_putc(' '); }  break;  case '%':  \_\_putc('%');  break;  default:  while (\*fmt != '%') fmt--;  break;  }  fmt++;  }  return 0;  } |

このプログラムはHello worldという文字列を表示するためだけに127行ものプログラムを書いています。これが一度だけというならまだしも、なにか文字列を表示するたびに、あなたはこれをコピーアンドペーストする必要があるのです！

しかし、関数を利用すると、この127行ものプログラムをまとめることができて、下記のように簡単に再利用することができます。

|  |
| --- |
| int main()  {  //HelloWorldと表示。  printf("Hello world\n");  //こんにちは世界と表示。  printf("こんにちは世界\n");  return 0;  } |

## **7.2 関数の定義と呼び出し**

関数を定義する

関数を定義する＝関数の処理を実装する＝関数の本体といった意味合いになります。後々関数宣言というものが出てきますが、それと混合しないように注意してください。

関数定義は関数の本体、プログラムの本体といった意味です。

関数は下記のように定義します。

戻り値　関数名( 引数リスト )

{

波カッコの間が関数の処理！！！

}

戻り値と引数リストは7.3と7.4で説明します。今は関数名だけ覚えてください。

関数呼び出し

新しいプロジェクト、Lesson7を作成してSample1.cppを入力してください。

関数を呼び出すには下記ように書く必要があることに注意してください。

関数名();

かっこが必要なことを忘れてしまう学生が多いです。()を忘れないようにしてください。

## **7.2 Ex\_1 ステップイン(F11)**

ステップインは関数の中にジャンプすることができる機能です。

参考動画

<https://www.youtube.com/watch?v=vljzUcT3XiU&feature=youtu.be>

## **7.2 Ex\_2関数化を行う理由**

プログラムを綺麗に書くため

　Lesson7では関数化を行う理由として、「プログラムの再利用性」を挙げています。これはもちろん正しいのですが、再利用を行うことができるプログラムというのは、それほど多くありません。そのため、それが最大の理由だと考えてしまうと、関数化を行うことが難しくなってきます。しかし、多くのプロジェクトでは、一か所でしか呼ばれない関数というものが沢山あります。つまり、**再利用されていない関数**が沢山あるのです。しかも、それらの関数は、再利用されている関数よりも多いのです！

では、なぜ関数化するのでしょうか？答えは、プログラムを見やすく、綺麗に描くためです。下記のプログラムは制作演習で行っている落ち物ゲームのゲームループです。

**関数化を行っていない落ち物ゲームのゲームループ**

|  |
| --- |
| //ここからゲームループ。  while (true) {  //1フレームの開始。  //フォントのサイズの設定。  const HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);  CONSOLE\_FONT\_INFOEX info = { sizeof info };  info.dwFontSize.X = 32;  info.dwFontSize.Y = 32;  SetCurrentConsoleFontEx(hConsole, 0, &info);  //コンソールの縦と横の行数を設定。  HANDLE hCons = GetStdHandle(STD\_OUTPUT\_HANDLE);  COORD dwSize = { 9999, 9999 };  SetConsoleScreenBufferSize(  hCons, // HANDLE hConsoleOutput  dwSize); // COORD dwSiz  //コンソールの高さを設定。  SMALL\_RECT consoleWindow = { 0, 0, 20, 20 };  BOOL ret = SetConsoleWindowInfo(  hCons, // HANDLE hConsoleOutput  TRUE, // BOOL bAbsolute  &consoleWindow);// CONST SMALL\_RECT \*lpConsoleWindow  //フレームバッファをクリア。  memset(sFrameBuffer, 0, sizeof(sFrameBuffer));  //背景を描画。  //二次元配列のmapを参照して、背景を描画していく。  for (int y = 0; y < 16; y++) {  for (int x = 0; x < 16; x++) {  if (map[y][x] == 0) {  //2次元配列のmap[y][x]に0が入っていたら、  //座標x, yの場所に空白文字を描画する。  kbcDrawMoji(x, y, ' ');  }  else if (map[y][x] == 1) {  //2次元配列のmap[y][x]に1が入っていたら、  //座標x, yの場所に@を描画する。  kbcDrawMoji(x, y, '@');  }  }  }  //プレイヤー１の移動処理。  for (int playerNo = 0; playerNo < 2; playerNo++) {  if (playerNo == 0) {  //1プレイヤー。  //プレイヤー１の移動処理。  if (GetAsyncKeyState(VK\_LEFT) != 0) {  //左のキーが押されたときの処理をここに記述する。  player[playerNo].posX--;  }  if (GetAsyncKeyState(VK\_RIGHT) != 0) {  player[playerNo].posX++;  }  }  else if (playerNo == 1) {  //2プレイヤー。  //続いてプレイヤー２の移動処理。  if (GetAsyncKeyState('A') != 0) {  //左のキーが押されたときの処理をここに記述する。  player[playerNo].posX--;  }  if (GetAsyncKeyState('D') != 0) {  player[playerNo].posX++;  }  }  //壁判定。  if (player[playerNo].posX < 1) {  player[playerNo].posX = 1;  }  if (player[playerNo].posX > 14) {  player[playerNo].posX = 14;  }  //落ち物との衝突判定。  for (int i = 0; i < otimonoArray.size(); i++) {  if (otimonoArray[i].posX == player[playerNo].posX  && otimonoArray[i].posY == player[playerNo].posY  ) {  otimonoArray[i].dead = 1;  //ゲット出来ていたらカウントアップ。  player[playerNo].score++;  }  }  //プレイヤー１を描画する。  if (player[playerNo].posX < FRAME\_BUFFER\_WIDTH && player[playerNo].posY < FRAME\_BUFFER\_HEIGHT) {  sFrameBuffer[player[playerNo].posY][player[playerNo].posX] = 'P';  }  }  //落ち物のプログラムを実行。  gameCount++;  if (gameCount == 10) {  //30フレーム経過したら新しい落ち物を発生させる。  Otimono otimono;  //落ち物のX座標をランダムで決定。  otimono.posX = (rand() % 14) + 1;  otimono.posY = 0;  //ゲームカウントを0に戻す。  gameCount = 0;  otimonoArray.push\_back(otimono);  }  //全ての落ち物を落下させる。  for (auto& otimono : otimonoArray) {  if ((gameCount % 5) == 0) {  //5フレームに一度落下する。  otimono.posY += 1;  }  }  //リストから削除。  otimonoArray.erase(std::remove\_if(otimonoArray.begin(), otimonoArray.end(), [&](auto& otimono)->bool {  return otimono.posY == 18 || otimono.dead == 1;  }), otimonoArray.end());  //落ち物を描画。  for (auto& otimono : otimonoArray) {  if (otimono.posX < FRAME\_BUFFER\_WIDTH && otimono.posX < FRAME\_BUFFER\_HEIGHT) {  sFrameBuffer[otimono.posY][otimono.posX] = '\*';  }  }  //スコアを表示する。  char scoreText[256];  sprintf\_s(scoreText, "Player1　スコア %d", player[0].score);  for (int i = 0; scoreText[i] != '\0'; i++) {  kbcDrawMoji(0 + i, 18, scoreText[i]);  }  sprintf\_s(scoreText, "Player2　スコア %d", player[1].score);  for (int i = 0; scoreText[i] != '\0'; i++) {  kbcDrawMoji(0 + i, 19, scoreText[i]);  }  //1フレームの終了。  //フレームバッファの内容を画面に表示する。  for (int i = 0; i < FRAME\_BUFFER\_HEIGHT; i++) {  for (int j = 0; j < FRAME\_BUFFER\_WIDTH; j++) {  std::cout << sFrameBuffer[i][j];  }  std::cout << "\n";  }  //32ミリ秒眠る。  Sleep(16);  //標準出力コンソールのハンドルを取得。  hCons = GetStdHandle(STD\_OUTPUT\_HANDLE);  COORD pos;  pos.X = 0;  pos.Y = 0;  //標準コンソールのカーソルを0行目、0列に戻す。  SetConsoleCursorPosition(hCons, pos);  CONSOLE\_CURSOR\_INFO cci;  // CONSOLE\_CURSOR\_INFO構造体の現在の状態を取得する  GetConsoleCursorInfo(hCons, &cci);  // メンバ変数であるbVisibleがカーソルの表示・非表示を制御する変数なので、これをFALSEにする事でカーソルを非表示にできる  cci.bVisible = FALSE;  // 変更した構造体情報をコンソールWindowにセットする  SetConsoleCursorInfo(hCons, &cci);  } |

いかがでしょうか。これが関数化を行っていない落ち物のゲームループのプログラムです。150行ほどのプログラムとなっております。では、続いて、関数化を行っている落ち物のゲームループを見てみましょう。

**関数化を行っている落ち物ゲームのゲームループ**

|  |
| --- |
| //ここからゲームループ。  while (true) {  //1フレームの開始。  **kbcBeginFrame();**  //背景を描画。  **DrawBackground()**;  //プレイヤーの処理。  for (int i = 0; i < 2; i++) {  **MovePlayer(player[i], i)**;  //落ち物との衝突判定。  **CheckHitPlayerAndOtimono(player[i])**;  //プレイヤー１を描画する。  **DrawPlayer(player[i])**;  }  //落ち物の描画処理。  **DrawOtimono()**;  //落ち物の移動処理。  **MoveOtimono();**  //スコアを表示する。  **DrawScore(player);**  //1フレームの終了。  **kbcEndFrame();**  } |

いかがでしょうか？関数化を行っている方だと、１フレームに何をしているのか？という処理の流れが一目霊山ではないでしょうか？この二つのプログラムは処理的には同じことをしています。ですが、１フレームの処理の流れをどちらが把握しやすいか？と問われると後者だと思います。このプログラムで網掛けになっている関数は再利用性を考慮して関数化を行っているわけではありません。一か所にしか記述されていませんね。そして、驚くべきことにすべての関数がそうです。

　関数化の最も大きなメリットは、「意味のある処理に名前を付けて、分割することでコードが読みやすくなる」ということだということが分かります。関数の再利用性は棚上げにして、まずは処理を分割するということを行っていきましょう。

この二つのプログラムは下記のURLにアップされています。両方とも動作するので、興味がある方はどうぞ。

<https://1drv.ms/u/s!AnpsoGtakIkVi41lbigVIJ8xjmOKHw>

## **Lesson\_07 ハンズオン 1**

サンプルプログラムのLesson\_07\_01/Questionを使用して、ハンズオン1～3を行いなさい。

## **Lesson\_07 実習課題\_1**

　サンプルプログラムのLesson\_07\_01/Questionを使用して、実習1～3を行いなさい。

サンプルプログラムのURL