しけぷり

某しけたい 4

2008 年 9 月 26 日

▷ もくじ ◁

1 ゼミノート（仮）

名前を書かれても死にません

しけたい４　(色々とごめんなさい)

1　オープニング・記事紹介

はい、しけたい４の代理です。ゼミの内容纏めは基本的に一人が喋り続けるだけです。なぜならネタに走る余裕がないから。今回は C++ 講座一回目の講義ログの纏めです。

【お断り】

・内容の正しさは保障できない

・しけたい４による追記などがある

・ネタ控えめ

・無断転載はやめちくれ

2　本題に入る前に、C++ の文法をちょっと確認

変数を扱う前にちょっとだけ、C++ のお作法を勉強しましょう。細かいところを省略しているのでこの文章をコンパイルしても無論上手く動きません。ここでは色々なサンプルで遊びましょう。まずは条件分岐の if 文からです。

```
if(A 君がちょんまげ){//ここに条件文が書かれる
    B 君は農民である//ここに条件が満たされた場合の結果
}//ここまで条件が満たされた結果
else{//条件が満たされなかった場合の結果
    B 君がちょんまげである
}//ここまで条件が満たされなかった場合の結果
```

A 君がちょんまげなら「B 君が農民」にそうでないなら「B 君がちょんまげ」になります。次は整数変数の宣言です。変数とは何かを覚えて良くてくれる人のことだと思うと扱いやすいでしょう。

```
int a=0;//a を 0 として宣言する
if(a==0){//a が 0 である場合と条件文を作る場合は a==0 と書く
        //a=0 と書いてしまうと a が 0 になるだけで判定が出来ない。
    a=1;
}
else{
    a=2;
}
```

だんだんソースコードらしくなってきましたね。この場合 a が 1 になります。a の宣言部分の値を 0 以外にすれば a は 2 になります。次は for 文です。for 文は繰り返しのための文章で
for(初期値；終了条件；ループの終点到達時の処理) {ループ内部}
と書きます。具体的には以下のようになります。cout の使い方は本編教材などを参考にしてください。

```
for(int i=5;i<10;i++){//ここを☆とする。i が 10 以上なら×へ
    cout<<ば～か<<endl;//（この場合 5 回ばーかといわれる）
}//ここで i++ が実行される。i が 10 より小さければ☆に戻る
//此処を×とする
```

```
for(int i=2;i<10;i=i*2){//ここを☆とする。i が 10 以上なら×へ
    cout<<ば～か<<endl;//（この場合 3 回ばーかといわれる）
}//ここで i=i*2 が実行される（*は掛け算）i が 10 より小さければ☆へ
//此処を×とする
```

次は while と do-while 文を使いましょう。基本的には for と似たような感じですが、ループ終点時の処理がなく、終了条件のみを while の部分に書く構造となります。

```
【while の書き方】
while(条件){
  ループ文章
}
【do-while の書き方】
do{
  ループ文章
}while(条件)
```

【while のサンプル】

```
int a=-2;//-2 なら▽が 2 回、0 以上なら▽は呼ばれない
while(a<0){//ここを☆とする a が 0 以上だったら×へ移行
 a=a+1;//此処を▽とする
}//☆へ戻る
//此処を×とする
```

【do-while のサンプル】

```
int a=2;//こっちでは a によらず▽は一度呼ばれる。
do{//ここを☆とする
 a=a+1;//此処を▽とする
}while(a<0)//a が 0 以上だったら×へ、そうでなければ☆へ移行
//此処を×とする
```

ちなみに、break は×へ。continue は☆へと強制移動をします。

```
int a=-2;
while(a<0){//☆はここだけど・・・
 break;//×へ強制移動
 a=a+1;//永遠に出番が無い
}//☆へ戻る
//此処を×とする
```

```
int a=-2;
while(a<0){//☆はここ
 a=1000;
 continue;//☆へ強制移動
 a=a+1;//永遠に出番が無い
}//☆へ戻る
//此処を×とする。a は 1001 でなく 1000 になる。
```

switch はちょっと面倒な分岐です。あまり出来は良くないといわれています。ある値を評価して、その結果によって【何処へ飛ぶか】を変えることが出来ます。

```
int a=0;
switch (a){
  case 0://a が 10 の時此処から下が実行される！
      cout<<"uho"<<endl;
  case 1://a が 10 の時此処から下が実行される！
      cout<<"iiotoko"<<endl;
  default://a が全ての選択肢に属さない時にここから下が実行される！
      cout<<"yaranaika"<<endl;
}
```

ちなみに、上記コードでは uho,iiotoko,yaranaika の全てが表示されます。a が 1 の時は iiotoko,yaranaika が表示されます。switch が何処に飛ぶかを指定しているだけのことがよく解るでしょう。a が 0 の時は uho だけを表示したい時は break を加えることが出来ます。

```
int a=0;
switch (a){
  case 0://a が 10 の時此処から下が実行される！
      cout<<"uho"<<endl;
      break;
  case 1://a が 10 の時此処から下が実行される！
      cout<<"iiotoko"<<endl;
  default://a が全ての選択肢に属さない時にここから下が実行される！
      cout<<"yaranaika"<<endl;
}
```

3 変数・関数紹介–global,local,static 変数

変数は簡単に言うと**何かを覚えておいてくれる奴**ですね。変数の宣言方法は【変数の型】【変数の名前】で宣言します。同じ名前の変数を宣言するとエラーが出ます。

`int a=0;//int` **が型で** `a` **が名前。**`a` **と言う名の** `int` **の変数を初期値** `0` **で宣言**

関数はある一定の作業を組み合わせたもので、例えば次のような使い方をします。

```
int a=fx(0);//x=0 で fx の値を聞く。つまり a は 2 になる。
a=fx(a);//x=2 で聞くので a は 4 になる。
int fx(int x){
    return x+2;//x+2 を値として返す
}
```

変数には有効範囲があり、それによって global,local,static に分かれます。基本的に有効範囲は変数の宣言位置から見て最も近い {} の入れ子です。[*1]

```
int z=1;
int b=fx(0);//b は 2 になる
int c=fy(0);//c は-2 になる
int fx(int x){//fx 内部の a の寿命はここから
    int a=1;//local な変数宣言。
    a=a+z;//a が 2 になる
    return x+a;//x+a つまり a+2 を値として返す。
}//fx 内部の a の寿命はここまで
int fy(int x){
    static int a=-1;//local な変数宣言。
    a=a-z;//a は-2 になる
    return y+a;//x+a つまり y-2 を値として返す。
}
```

上記サンプルの場合、z,b,c が global,a が local な変数です。z と言う名前の int 変数はこのソースコードの何処でも使うことが出来るため、fx,fy の中で z と言う名前の変数は宣言できません。a は関数に囲まれたところで宣言された local な変数です。fx 内部の a は fx の中で毎回宣言され、fx が終了すると消滅するので fx の中で宣言しても fy で宣言することが出来ます。static はちょっと特別な人で、【宣言を一回だけ】行い、その中で生き残り続ける変数です。

[*1] コラム・変数のスコープ参照

static を用いると前とは違う結果が得られます。

```
int z=1;
int b=fx(0);//b は 2 になる
int c=fy(0);//c は-2 になる
b=fx(0);//b は 3 になる。a が static でなければ 2
c=fy(0);//c は-3 になる。a が static でなければ-2
int fx(int x){
      static int a=1;//static な変数宣言。一回しか呼ばれない
      a=a+z;//a は 2,3,4・・・と増えていく
      return x+a;//x+a を値として返す。a は保存される
}
int fy(int x){
      static int a=-1;//static な変数宣言。一回しか呼ばれない
      a=a-z;//a は-2,-3,-4 と減っていく
      return y+2;//y-2 を値として返す。a は保存される
}
```

普通の変数は値を返す（return する）必要があります。値を返す必要がないけど、カプセル化すると便利なもの（変数を初期化するなど）を扱いたい場合は void の関数を使いましょう。

```
int a=10;
a=3;
a=6;
syokika();//a が 0 になる。
void syokika(){
      a=0;//a は global だから弄れる
}
```

3.1 スコープ、変数の寿命

変数の寿命は最寄の {} までです。つまりは、以下のソースコードは a が宣言範囲の外に出てしまっているので全てダメです。

```
int a;
addb();
a+=b;//b は此処では使えない。
void addb(){
   static int b=0;
   b+=1;
}
```

```
for(int i=0;i<10;i++){
   cout<<"i="<<i<<endl;
}
   cout<<i<<"回も呼びやがって！"<<endl;//この行はダメ
```

```
int a=0;
if(a==0){
    int b=5;
}
cout<<b<<endl;//ここでは b は使えない。
```

4　演算子の使い方

4.1　ラクチンな演算子

前にやったとおり、足し算、引き算、掛け算、割り算はこのように宣言すれば OK です。

【足し算】

```
int a=0; //変数の宣言
a=a+4; //a は 4 になる
```

【引き算】

```
int b=100; //変数の宣言
b=b-4; //b は 96 になる
```

【掛け算】

```
int a=1; //変数の宣言
int b=5;
int c= a*b*b; //c は 25 になる。
```

【割り算】

```
int a=100; //変数の宣言
int b=4;
int c=a/b;//c は 25 になる
```

++ 演算子は特別な演算子です。どちらも、値に 1 を足したり引いたりするのですが使い方により【参照】されるときに出てくる値が違います。まずは普通に使ってみましょう。

【例１】

```
int a=0;
a++;
int b=a;//b は 1 になる
```

【例２】

```
int a=0;
++a;
int b=a;//b はやっぱり 1 になる
```

参照するというのは具体的にはこんな感じです。

【例１】

```
int a=0;
a++;//a は 1 になる
int b=a++;//a++ の場合は、b は a に値が足される前の値の 1 になる
int c=a;//b=a++ でも a の値はしっかり足され c は 2 になる。
```

【例２】

```
int a=0;
++a;//a は 1 になる
int b=++a;//++a の場合は、b は a に値が足される後の値の 2 になる
int c=a;//上に同じく c は 2 になる
```

他にも +=（後ろの値を足す）などがあります。

```
int a=3;
int b=1;
b+=a;//b に a の値 3 を足す。
```

ですね。

5 論理記号

もし○○だったらという条件分岐は超重要な技ですね。論理記号の考え方自体はこの講義を読む殆どの人が理解できていることでしょう。ここでは、C++ における論理記号の使い方を説明します。ちょっと見慣れないものもあるかもしれませんが、C++ のお作法として割り切っやってください。

if() の () の中身が 0 以外なら中括弧の中身が実行されます。

【基本の復習】

`int a=1;int b=1;int c=0;//a,b,c` を 1,1,0 にする

`if(a==b){`メッセージ：ばーか`}//a` と `b` の値が等しいなら、ばーかといわれる

`if(a){`メッセージ：うほっ`}//()` の中が 0 でないなら、うほっと言われる

`a=(b==c);//b` と `c` が等しくないので `a` は 0 になる。

ちょっと意地悪な and 演算です。数学用語で言えば「かつ」ですね。

【and 演算】

`int a=1;int b=1;int c=0;//a,b,c` を 1,1,0 にする

`if(a==1 && b!=c){cout<<nya;}//a` が 1 かつ `b` と `c` 等しくないと実行

`if(a==0 && b++ == 1){cout<<ge;}//cout<<ge` は実行されない

`c=b;//`上の条件式は `a==0` の時点で結果が解り切っているので `b++` は呼ばれず `c` は 1

and があれば or もあります。数学用語で言えば「または」ですね。

【or 演算】

`int a=1;int b=1;int c=0;//a,b,c` を 1,1,0 にする

`if(a==1 || b!=c){cout<<nya;}//nya` と表示される

`if(a==0 || b++ == 1){cout<<ge;}//ge` とも表示される。

`c=b;//b++` が処理されたので `c` は 2

慣れてくると使いたくて溜まらない三項条件もあります。

【三項条件】

`int a=1;int b=1;int c=0;//a,b,c` を 1,1,0 にする

`a= b>1? 1:2;//a` は `b>1` なら 1、そうでないなら 2 となる

`c= a==b? 3:4//c` は `a` と `b` が等しいなら 3, そうでないなら 4 となる

6　演算子の優先順位

自信がないなら () で括るか、自分で調べるべきなのですがちょっとぐらいは知っておいて良いでしょう。基本は左結合で、代入は右結合です。

【例】

`int a=1;int b=1;int c=2;//a,b,c` を 1,1,0 にする

`a=a+b+c;//a=(a+b)+c` と扱われる

`a+=b+=c;//a+=(b+=c)` となる

`a=(b+(c+=b))+c;//`括弧で括れば安心

そして、優先順位に従って、変数を随時定数に書き換えていきます。例えば・・

```
int x=2;
x+=x+=x;//こいつは
//x+=(x+=x) と解釈されて
//x+=(x+=2) と変換。
//x+=(x=2+2) この時点で x は 2 なのでこうなる
//x+=4 よってこうなる。
//x=4+4 この時点で x は 4 なのでこうなる
```

7 型と型変換

7.1 皆大好き型

変数にはタイプがあります。例えばラーメン屋の注文を扱う変数でニンニク入れるか入れないかを保存するものがあったとしましょう。言うまでも無いですが、この変数は 0 と 1 が使えれば十分ですね。料金を扱う場合は大きな数字を扱える必要があります。カロリー計算なら、小数点も覚えておきたいものです。店へのアンケートなら文字を覚えさせておく必要があります。このように変数には用途に適合した「覚えるもの」があります。これが型です。

整数を覚えるタイプはこんなのがあります。

```
char a;//8bit 00000000 から 11111111 まで扱える。
short b;//16bit char と同じ感じだがサイズがでかい
int c;//32bit 皆大好きおなじみさん
long d;//32bit %int との差は？
long long e;//64bit 凄く大きいです
__int64 f;//64bit の大きい子。
```

大きいサイズの子はパソコンのスペックの進化に伴い生まれた子だったりして、ちょっと使い方に難があったりします。ここでは詳細は説明しないので適時調べましょう。とりあえず覚えておくべきは、「使い方に難があるからエラーの原因はそいつかも」とも疑う心構えです。

小数点付きで覚えるタイプはこんなのです。小数点付きの子は実際の桁と小数点の位置を覚えています。例えば 3.1415 は「31415」と「$\times 10^{-4}$」と覚えています。

```
float a;//32bit の大きい子
double b;//64bit の子。計算機の思い出が滾るでしょう。
```

文字を覚える子。日本語は日本語があるので面倒ですね。文字コードは国などに依存していて、日本のウィンドウズでは Shift __ JIS と Unicode16 のようです。

```
char a;//8bit
wchar_t b;//16bit の子
```

ちょっと特別な bool 型。真であるか偽であるか、先ほどのラーメン屋で言うニンニク入れるかに当たる人ですね

`bool a;//1bit`**。昔は違っただとか**

7.2 変換ってなんぞや

さて、変数にはルールがあります。整数を名乗る変数には整数しか扱えません。例えばこんなマネをすると・・・

`int a = (int) 3.1415;//a`**は切捨てされて** 3 **になる**
`int b=int(3.141592);//b`**もやっぱり切り捨てされる**
`if(a==b){`**メッセージ：ばーか**`}//a`**も** `b` **も** 3 **なのでばーかといわれる**

切捨てのやり方などは変数の型によって異なります。全部覚えておくのも良いですが、あまり自分から使いに行く技術ではありませんね。ひとまずは、このような処理がなされているということを覚えておいてください。

8 リテラル

リテラルは簡単に言うと**人間様のための表記方法**です。例えば CPU は数字の 5 を **0 と 1 の羅列**で覚えているわけです。かといって 0 や 1 の羅列がソースコードにあると汚くて読めませんね。コンパイラはその辺の通訳する仕事も含んでいると言うわけです。

`int a=16;//16`**はリテラル。つまり人間語**
`bool b=false;//false`**は** `if` **文での** 0 **に相当。逆は** `true`
`char c='A';//'A'`**は人間語**
`wchar_t d = L'A'//L'A'`**が人間語。**

9 サンプル

以下のソースコードを実行した場合、cout で何が表示されるか考えてみましょう。%を半角%にすれば、このソースコードはコピペで動くはずです。

```
#include <iostream>//これと
using namespace std;//これで cout が使えるようになる

int samplekansu1(int x);//関数は宣言する必要がある。
void samplekansu2();//関数は宣言する必要がある。
bool samplekansu3(int x,int y);

int main() //C++ で作られたものは main から実行される
{
        cout<<samplekansu1(2)<<endl;

        int i=0;
        for(i=0;i<5;i++){
                samplekansu2();
        }
        cout<<"loop1 end"<<endl;
        for(i=2;i<100;i=i*2){
                samplekansu2();
        }
        cout<<"loop2 end"<<endl;
        i=0;
        while(i<10){
                samplekansu2();
                i+=1;
        }
        cout<<"loop3 end"<<endl;
        do{
                samplekansu2();
        }while(i++==11);
        cout<<"loop4 end"<<endl;

        cout<<"i="<<i<<endl;
        switch(i){
                case 11:
                        cout<<"uho"<<endl;
                default:
                        cout<<"iiotoko"<<endl;
        }
        cout<<"loop5 end"<<endl;
        cout<<samplekansu3(2,1)<<endl;
        cout<<samplekansu3(2,2)<<endl;
```

```
        return 0;//main の実行が終わるとプログラムが終わる
}

int samplekansu1(int x){
        cout<<x+5<<endl;
        return x;
}

void samplekansu2(){
        int a=0;
        static int b=0;
        a+=1;
        b+=1;
        cout<<a<<"th caved!!!!"<<endl;
        cout<<b<<"th caved!!!!"<<endl;
}

bool samplekansu3(int x,int y){
        if(x==y){cout<<"x=y"<<endl;}
        //半角%は剰余
        if(x==y && x % 2==0){cout<<"x=y and x ha guusu!"<<endl;}
        if(x==y || x % 2==0){cout<<"x=y or x ha guusu!"<<endl;}
        return x++==++y;
}
```

10 サンプルこたえ

```
#include <iostream>//これと
using namespace std;//これで cout が使えるようになる

int samplekansu1(int x);//関数は宣言する必要がある。
void samplekansu2();//関数は宣言する必要がある。
bool samplekansu3(int x,int y);

int main() //C++ で作られたものは main から実行される
{
        cout<<samplekansu1(2)<<endl;//関数内の cout が先に仕事する
        //よって、7 と 2 が表示される

        int i=0;
        for(i=0;i<5;i++){
                samplekansu2();
                //static とそうでない関数があるので
                //以下出力は 1,1,1,2,1,3・・となる
        }
        cout<<"loop1 end"<<endl;
        for(i=2;i<100;i=i*2){
                samplekansu2();
                //i が 2,4,8,16,32,64 の時に実行される。よって 6 回。
        }
        cout<<"loop2 end"<<endl;
        i=0;

        while(i<10){
                samplekansu2();
                //10 回呼ばれる
                i+=1;
        }

        do{
                samplekansu2();
                //下の条件式に関係なく一度は呼ばれる
        }while(i++==11);
        //i++ は 10 なので一回だけ呼ばれる
```

```
        cout<<"i="<<i<<endl;
        switch(i){
                case 11://此処から下が実行される
                        cout<<"uho"<<endl;
                default:
                        cout<<"iiotoko"<<endl;
        }
        //uho iiotoko が表示される
        cout<<samplekansu3(2,1)<<endl;
        cout<<samplekansu3(2,2)<<endl;
        return 0;//main の実行が終わるとプログラムが終わる
}

int samplekansu1(int x){
        cout<<x+5<<endl;
        return x;
}

void samplekansu2(){
        int a=0;
        static int b=0;
        a+=1;
        b+=1;
        cout<<a<<"th caved!!!!"<<endl;
        cout<<b<<"th caved!!!!"<<endl;
}

bool samplekansu3(int x,int y){
        if(x==y){cout<<"x=y"<<endl;}
        //半角%は剰余
        if(x==y && x % 2==0){cout<<"x=y and x ha guusu!"<<endl;}
        if(x==y || x % 2==0){cout<<"x=y or x ha guusu!"<<endl;}
        return x++==++y;
}
```