入門講習会 第四回

1. 今日の内容

お題となる問題はなし。前回の演習の続きを行う。

おまけとして、テーマとして未分類のことを説明する。

1. 不定値

いままで変数は、代入または初期化をした後に利用していた。では、代入も初期化も行っていない変数の中身はどうなっているのだろうか。これは試しに次のプログラムを組んでみればわかる。

|  |
| --- |
| #include <stdio.h>  int main(void) {  int x, y, z;  printf("%d %d %d\n", x, y, z);  return 0;  } |

実は、初期化されていない変数には値が入っている。しかしこの値は意味のない不定値である。厳密には、かつては何かのプログラムで利用されていた値だったが、そのプログラムの終了とともに使われなくなったゴミの値である。

実行結果は、筆者の場合は以下のようになった。表示される値は環境や時によって異なる。

|  |
| --- |
| 2 2 0 |

1. キャスト演算子

プログラミングをしていると、型を変換したいときがたまにある。そんなとき、ある型の値を別の型に変換する演算子がCでは用意されている。これをキャスト演算子という。以下のような書式で用いる。

|  |
| --- |
| (型)値 |

型には、intやdoubleやcharなどが入れられる。値には、変数や定数を入れる。これによって、変数や定数を目的の型に変換することができる。

例を見てみよう。以下は、実数xの値を整数値に変換してyに代入するプログラムである。

|  |
| --- |
| #include <stdio.h>  int main(void) {  double x;  int y;  scanf("%lf", x);  y = (int)x;  printf("x: %f\n", x);  printf("y: %d\n", y);  return 0;  } |

実行結果は以下のようになる。15.13が入力例である。

|  |
| --- |
| *15.13*  x: 15.130000  y: 15 |

15.13がxに代入された後、

y = (int)x;

によって、15.13を整数型に変換してyに代入される。整数型に変換すると小数点以下が切り捨てられ、結局15がyの値となる。

キャスト演算子は型を明示的に変換したいときに使える。次に述べる「暗黙の型変換」によっても型は変換されるが、型を任意のタイミングで変換したいならキャスト演算子を使うべきである。

もっとも、キャストを多用する機会はあまりない。無理やりキャストするくらいなら変数の型を最初から合わせておくべきであるからである。

1. 暗黙の型変換

整数型の変数に無理やり実数を代入するとどうなるのだろうか。試しにやってみよう。以下のプログラムは、前項のプログラムからキャスト演算子の記述を取り除いたものである。

|  |
| --- |
| #include <stdio.h>  int main(void) {  double x;  int y;  scanf("%lf", x);  y = x;  printf("x: %f\n", x);  printf("y: %d\n", y);  return 0;  } |

実行結果は以下のようになる。15.13が入力例。

|  |
| --- |
| *15.13*  x: 15.130000  y: 15 |

キャスト演算子を用いなくても、15.13がyに代入されるとき、整数値に変換されていることが分かるだろう。yが整数型の変数だったから、自動的に15.13は整数値に変換されたのだ。

この変換を「暗黙の型変換(暗黙のキャスト)」と言う。

暗黙の型変換が行われるのは代入ときのだけではない。実は異なる型同士で何かしらの演算を行った際にも暗黙の型変換が起こる。

演算時の型変換のとき、なるべく表現力の高い型に変わると思っておけばよい。例えば、整数と実数の計算の時、計算式中の整数値は実数にキャストされてから計算が行われる。

例を見てみよう。整数a, bの平均値を実数c1, c2に代入し、それを出力するプログラムである。c1とc2で微妙に計算式を変えていることに注目してほしい。

|  |
| --- |
| #include <stdio.h>  int main(void) {  int a, b;  double c1, c2;  scanf("%d %d", a, b);  c1 = (a + b) / 2;  c2 = (a + b) / 2.0;  printf("%f\n%f\n", c1, c2);  return 0;  } |

実行結果は以下のようになる。3, 4が入力例。

|  |
| --- |
| *3 4*  3.000000  3.500000 |

c1では小数点以下が切り捨てられている。なぜならば、c1の計算では整数同士でしかやっていないからだ。整数同士の割り算は切り捨てのため、小数点以下が切り捨てられた後に、実数への型変換が行われ、c1に代入される。

一方c2では、まず足し算は整数同士なので、a+bは整数値7に読み替えられる。7/2.0を計算するとき、式中に2.0という実数が入っているため(実数定数を明示したい場合は小数点をつける)、計算式中の7は実数7.0に型変換される。計算結果は実数であり、それがそのままc2へと代入される。

a = 3, b = 4のとき

(3+4)/2 = 7/2 = **3**

代入時の型変換

**3.0**

c1

代入

(3+4)/2.0 = **7**/2.0

演算時の型変換

**7.0**/2.0 = 3.5

c2

代入

暗黙の型変換はその名の通り、「暗黙」のうちに起こる。そのため、型変換について考慮していなかったことによるバグが起こるかもしれない(筆者は起こりました)。

異なる型同士の演算をしなければならなくなった場合、暗黙の型変換について考えよう。double型、int型、long long int型のうちどれか二つ以上の型の変数が一つのプログラムで宣言されているときは、特に注意が必要である(long long intについては別のときに話します)。

1. 四捨五入

ここでは、非負実数の小数第一位を四捨五入した整数値を求めることを考えよう。

単に0.5を加えて、整数型にキャストすればよい。なぜなら、もし小数第一位が6以上なら、繰り上がりが起こるからである。

実数 3.819の小数第一位を四捨五入

3.819 + 0.5 = 4.319

4

int型へキャスト

例を見てみよう。以下は、入力する整数値を四捨五入して、整数値で表示するプログラムである。

|  |
| --- |
| #include <stdio.h>  int main(void) {      double x;      int y;      scanf("%lf", &x);      y = (int)(x + 0.5);      printf("%d\n", y);  return 0;  } |

実行結果は以下のようになる。3.613, 5.12が入力例。

|  |
| --- |
| *3.613*  4 |
| *5.12*  5 |

1. 切り上げと整数同士の割り算

小数点以下切り上げというのは、「ある数について、小数部分が0より大きいならば、その数に1を加えて小数点以下を切り捨てる」という意味である。よって、前項をまねた方法で、「実数値に変換して、0.9を加えて、整数値に戻す」という方針ではうまくいかないことに注意しよう。なぜなら、0.9を加えても小数第二位以下に何も影響を及ぼさないからだ。例えば、割り算の結果が3.001の場合が反例である。

まず、簡単に思いつくのは次の方法である。 (実数値) – (実数値の整数部分) > 0であるかどうかを判定し、真なら小数点以下を切り捨てた数に1加えるという方法である。

入力値を切り上げるプログラムは以下のように書ける(実行結果は省略)。

|  |
| --- |
| #include <stdio.h>  int main(void) {      double x;      scanf("%lf", &x);      if(x - (int)x > 0) printf("%d\n", (int)x + 1);      else printf("%d\n", (int)x);  return 0;  } |

ちなみに、このようにしなくても、切り上げを行うための関数ceilがあらかじめCでは用意されている。

|  |
| --- |
| ceil(値) |

実数値を値の部分に書くと、その値を切り上げた値を返す。ただし返ってくるのは実数型であるため、計算や出力時には注意が必要である。

ただしabsと同様、ソースコード上部に#include <math.h>を記述する必要がある。以下はceilを使って先のコードを書き換えたものである。

|  |
| --- |
| #include <stdio.h>  #include <math.h>  int main(void) {      double x;      scanf("%lf", &x);      printf("%d\n", (int)ceil(x));  return 0;  } |

[補足]

double型の精度の限界により、切り上げが正しく行われない場合があるので注意が必要である。例えば先のプログラムにおいて、5に限りなく近い数5.0000000000000001を入力すると、6ではなく5の値が出力されてしまう。

|  |
| --- |
| *5.0000000000000001*  5 |

一般に、実数を扱うときは誤差のことを考えなくてはならない。この辺りはやや込み入った話になるため、詳細は割愛する。

ここで、正整数a, bの割り算の結果を切り上げ、それをcに代入するという問題を考えてみよう。

C言語では、整数同士の割り算は小数点以下を切り捨てる。しかし、小数点以下を切り上げたいときがたまにある。では先ほどのようにceilを使って、

|  |
| --- |
| c = (int)ceil((double)a / b); |

と書けばよいのだろうか。それも一つの方法だろう。しかし、できるだけ整数同士の計算は整数だけで扱いたい。なぜなら、実数には誤差が入ることがあるからだ。

以下、割り算記号「/」の結果が切り捨てであることに注目しよう。

実数への変換を介さずに割り算の結果を切り上げる方法が2つ考えられる。一つは、「割った余りが1以上なら、割り算の結果に1加える」という方法である。

|  |
| --- |
| if (a % b > 0) c = a / b + 1;  else c = a / b; |

もしaがbで割り切れなければ、必ず割り算の結果は小数点以下の数を持ってしまう。だから割り算の結果に1加えなくてはいけない。

もう一つは、aを割られる数、bを割る数として、割り算の結果を(a+b-1)/bと計算することである。

|  |
| --- |
| c = (a + b - 1) / b; |

もし(a+b)/bだったら、(a+b)/b = a/b + 1となってしまい、小数点以下に0以外の数字があろうが1を加えることになってしまうのだが、bより1小さい(b-1)をaに加えることで、切り上げを実現している。

厳密にこのことを示したいなら、aをbで割った商をn、余りをmとしてa = nb + mと表して、(a+b-1)/bに代入してみよう。

m = 1, 2, …, b – 1のとき、であるから、この数は切り捨てによって0となる。よって、割り算の結果として(n+1)が残り、切り上げができていることが分かる。

先に説明したif文を用いる方法のほうが直感的でわかりやすいが、この方法の方がより簡潔に書ける。どちらがいいかは好みによる。

試しに、二つの方法でそれぞれ試したプログラムを書いてみよう。以下は、入力された正整数a, bに対して、a÷bを切り上げた値を表示するプログラムである。

|  |
| --- |
| #include <stdio.h>  int main(void) {      int a, b;      scanf("%d %d", &a, &b);      printf("Method1: ");      if(a % b > 0) printf("%d\n", a / b + 1);      else printf("%d\n", a / b);      printf("Method2: ");      printf("%d\n", (a + b - 1) / b);  return 0;  } |

実行結果は以下のようになる。

|  |
| --- |
| 37 10  Method1: 4  Method2: 4 |

37÷10 = 3.7だが、これを切り上げて4としている。