

## 1 問題設定

被乗数と乗数の 2 つの整数を端末から入力し、その積を出力するプログラムを作成する。

1: 例

```
1 [onosans@onosans-shyvana w02]$ ./ex.out
2 被乗数と乗数を入力してください
3 被乗数: 365
4 乗数: 17
5 365 x 17 = 6205
```

## 2 問題分析

今回の問題はまずユーザーに対しどのような値が欲しいのか示す必要がある。そのため `printf` で「被乗数と乗数を入力してください」と出力する。

次に乗数、被乗数の入力を受け付ける。まず入力された値を一度メモリに保存する必要があるため整数 (`int`) 型の変数を 2 つ宣言する。次にどちらの値を受け付けるかなどの細かい表示をして、`scanf` を使い先ほど宣言した変数に入力された値を代入する。

最後にそれらを計算した結果を出力すれば終了となる。

## 3 設計

今回作成するプログラムのフローチャートを以下に示す。

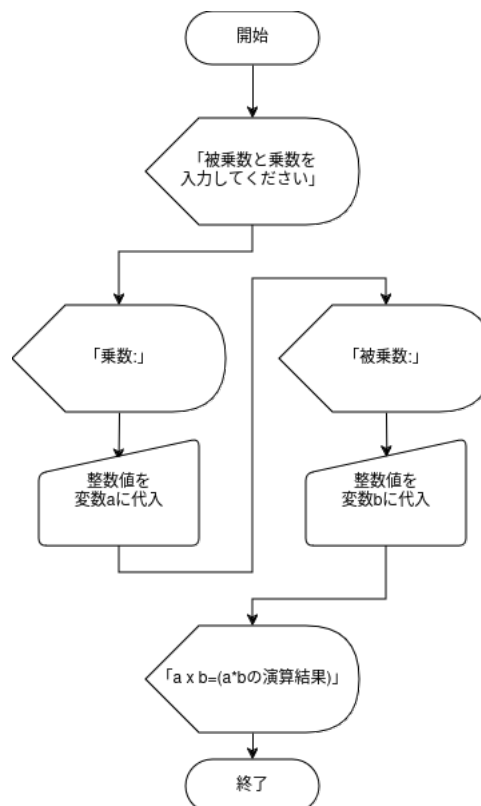


図1: フローチャート

## 4 実装

まず変数表を以下に載せる。

表1: 変数表

データ	変数名	データ型	説明
入力	a	int	被乗数
入力	b	int	乗数

それぞれの変数の役割は載せた表の通りである。次に今回書いたコードを載せる。

2: ソースコード

```

1 #include <stdio.h>
2
3 int main(void){
4     int a, b; // 整数型の変数a, bを宣言
5     printf("被乗数と乗数を入力して下さい\n");
6     printf("被乗数:");
7     scanf("%d", &a); // aに入力された値を代入
8     printf("乗数:");
9     scanf("%d", &b); // bに入力された値を代入
10
11     // 乗算した結果の出力
12     printf("%d x %d = %d\n", a, b, a*b);
13
14     return 0;
15 }
```

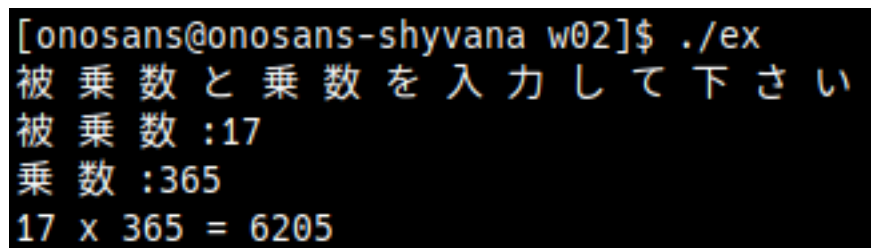
前回から使っているものの説明は省く。scanf 関数はターミナルからの入力を受け付ける関数である。入力された値をどこの変数にどのような形式で保存するのかを、第 1 引数に書式指定子、第 2 引数に変数のメモリ上のアドレスを渡すことで指定の変数に値を保存している。10 進数の場合、書式指定子は%d、変数のアドレスは変数名の前に&をつける。

次に printf の新しい使い方として書式指定子を用いて変数の値を代入する方法がある。使い方はソースコード2のように対応する場所に書式指定子を入れ、対応する値(変数や演算)をコンマ区切りで入れていく。

これらの機能を使って以上のようにコードを組み立てた。

## 5 検証

実行結果は以下の通りである。



```

[onosans@onosans-shyvana w02]$ ./ex
被乗数と乗数を入力して下さい
被乗数:17
乗数:365
17 x 365 = 6205
```

図2: 実行結果

実行環境は前回と異なり、ChromeBook に ArchLinux を入れたもので実行した。

## 6 考察

### 6.1 オーバーフローさせてみる

私の使用しているものも `int` 型のサイズは 32bit のため -2147483648 から 2147483647 までの整数を表現することができる。試しに 2147483648 と 1 を入力してみる。すると結果は以下のようになる。

#### 3: オーバーフロー 1

```
1 [onosans@onosans-shyvana w02]$ ./ex.out
2 被乗数と乗数を入力してください
3 被乗数: 2147483648
4 乗数: 1
5 2147483648 x 1 = -2147483648
```

表2

	10 進数	2 進数
被乗数	2147483648	1000 0000 0000 0000 0000 0000 0000 0000
答え	-2147483648(2 の補数)	1000 0000 0000 0000 0000 0000 0000 0000

このように 2 進数で見ても正しいことがわかる。2 の補数の仕組み上、表現できる最大値の次の値は表現できる最小値になることを改めて確認することができた。

### 6.2 乗算を使わない実装

この授業とは関係ないが計算機アーキテクチャで「掛け算は足し算の繰り返しで実装できる」ということを聞いたので、足し算を繰り返す `ver.` を作ってみた。尚、繰り返しの記述方法は既に知っていたので参考文献には記述していない。

#### 4: 加算のみでの実装

```
1 #include <stdio.h>
2
3 int main(void){
4     int a, b;
5     printf("被乗数と乗数を入力して下さい\n");
6     printf("被乗数:");
7     scanf("%d", &a);
8     printf("乗数:");
9     scanf("%d", &b);
10
11     int ans=0; // 積を保持する変数
12
13     // b回繰り返し実行するという意味
14     for(int i=0; i < b; i++){
15         ans+=a; // ansにaの値を加算する
16     }
17
18     // 乗算した結果の出力
19     printf("%d x %d = %d\n", a, b, ans);
20
21     return 0;
22 }
```

こちらは乗数の回数分だけ加算を繰り返しているだけである。今回の場合整数型なのでこれで実装することができるが、浮動小数点型になった場合は 1 以下の部分については被乗数を割れば実装できそうだがそもそも割り算をどう実装するのかあまり思いついていない状況ではある。

表3: 変数表 2

データ	変数名	データ型	説明
出力	num	int	演算途中の値の保存
入力	a	int	被乗数
入力	b	int	乗数

### 6.3 仮想的な乗算器を用いた実装

ここからは完全に遊びなのだが仮想的な乗算器を実装することによって、今回のプログラムを作ってみようと思う。ここでは乗算器のみ仮想的に実装するので加算等は用いるが許してほしい。

また新たに使用した変数のみ変数表にまとめた。

表4: 変数表 3

データ	変数名	データ型	説明
入力	a	int	被乗数
入力	b	int	乗数
内部	bool_a[32]	bool	被乗数を配列で表現
内部	bool_b[32]	bool	乗数を配列で表現
内部	n[32]	int	int 型整数を配列で表すために使用する

2 の補数で表された数を式で書くなら以下ようになる。

$$a = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$$

ここで用いている  $a_n$  は  $A$  を 2 進数で表したときの  $n$  bit 目の値を表している (そのまま計算してもちゃんと-になるようにしてある)。この状態で加算を考えるなら以下ようになる。

$$a \cdot b = (-a_{n-1}2^{n-1} + \dots + a_1 2^1 + a_0 2^0)(-b_{n-1}2^{n-1} + \dots + b_1 2^1 + b_0 2^0)$$

-のつく項をまとめると

$$\begin{aligned} & -a_{n-1}b_0 2^{n-1} - a_{n-1}b_1 2^n + \dots - a_{n-1}b_{n-2} 2^{2n-3} \\ & -a_0b_{n-1} 2^{n-1} - a_1b_{n-1} 2^n + \dots - a_{n-2}b_{n-1} 2^{2n-3} \end{aligned}$$

となっており、これらは「各桁ごとの乗算の最後の値」と「最後の桁の乗算の最後以外の値」に分類される。これらを if 文で条件分岐させ負の数にした後足し合わせることで負の数の乗算も実装することができる。

実際に論理回路で実装する場合はビット反転ぐらいしかできないので、このマイナスのつく値の部分(後ろの 2 の指数)がかぶらないようにまとめるとちょうどよく 2 つの組ができることがわかるだろう。これらを 2 の補数で表し計算に組み込むことで実装できる。

いちいち 2 の補数にするといった動作をしていては効率が悪いので一気に計算してみると  $n + 1$  bit 目と  $2n + 1$  bit 目に 1 が加わることがわかる。この 1 を実装すれば乗算器の完成だ。

C 言語上で実装する場合、ビット反転をしている方が追加で書く量や変数も増えてしまうのでそのまま単純にマイナスをつけて実装している。以下に機能ごとにソースコードを載せる。

## 7 所感

### 参考文献

- [1] ” 乗算器”. Wikipedia. 2024 年 6 月 1 日. <https://ja.wikipedia.org/wiki/%E4%B9%97%E7%AE%97%E5%99%A8>, (2024 年 10 月 19 日).
- [2] ” 符号付き 2 進数の乗算”. Qiita. 2019 年 8 月 18 日. [https://qiita.com/ryo\\_i6/items/f665a2267be6ba59c346](https://qiita.com/ryo_i6/items/f665a2267be6ba59c346), (2024 年 10 月 19 日).