岐阜工業高等専門学校

第 10 回課題

電子制御工学科: 情報処理 I

担当教員：岡崎憲一

柴田 健琉

(学籍番号：2024D14　名列番号：15)

提出日：令和 7 年 06 月 24 日

令和 7 年 06 月 19 日

Made with LaTeX + vim

目次

1　はじめに

1.1　実行環境

この課題のプログラムは以下の環境で動作することが確認されている：

- OS: Arch Linux
- CPU アーキテクチャ: `x86_64`
- C コンパイラ: `gcc バージョン 14.2.1 20250322 (GCC)`
- C コンパイラオプション: `-Wall`

2　今回の構文

2.1　`switch` 文

`switch` 文は `if` 文と同じように条件分岐を行う文である。if 文と違い、論理式・論理値による分岐ではなく、評価後の整数値によって分岐する。それぞれの条件は `case` ラベルで記載され、コロン (`:`) の後に処理を記述する。処理の終わりは `break` 文を書く。ケースは縦落ちすることができ、`break` 文を書かずにしておくことで複数の値に同じ処理を割り当てることができる。`if` 文における最後の `else` は `switch` 文では `default` ラベルとなっている。[2]

`switch` 文

```
switch (式) {
    case <値1>:
        <処理>...   // 値1の場合の処理
        break;
    case <値2>:
        <処理>...   // 値1の場合の処理
        break;
    default:
        <処理>...   // それ以外の場合の処理
}
```

縦落ち (Fall-through) する `switch` 文

```
switch (式) {
    case <値1>:
        <処理>...   // 値1の場合の処理
        break;
```

```
    case <値2>:
    case <値3>:
        <処理>...   // 値2と値3の場合の処理
        break;
    case <値4>:
        <処理>...   // 値4の場合の処理
        break;
    default:
        <処理>...   // それ以外の場合の処理
}
```

3　演習 3-12

教科書の List 3-4 を Switch 文で書き換えたプログラム。

3.1　コードリスティング

演習 3-12

```c
#include <stdio.h>

int main(void) {
    int n;

    printf("Input␣Integer:␣");
    scanf("%d", &n);

    switch (n % 2) {
        case 0:
        puts("The␣number␣is␣even.");
        break;
    case 1:
        puts("The␣number␣is␣odd.");
        break;
    default:
        puts("Unknown␣Error");
    }

    return 0;
}
```

3.2　実行結果

```
information-processing-1_10th-class/programs/p312 on  main [!] via C v14.2.1-gcc via ❄  impure (LaTeX-Environment-env)
❯ gcc -Wall main.c -o main
main.c: 関数 ‘main’ 内:
main.c:7:5: 警告: ignoring return value of ‘scanf’ declared with attribute ‘warn_unused_result’ [-Wunused-result]
    7 |     scanf("%d", &n);
      |     ^~~~~~~~~~~~~~~

information-processing-1_10th-class/programs/p312 on  main [!?] via C v14.2.1-gcc via ❄  impure (LaTeX-Environment-env)
❯ ./main
Input Integer: 12
The number is even.

information-processing-1_10th-class/programs/p312 on  main [!?] via C v14.2.1-gcc via ❄  impure (LaTeX-Environment-env)
❯ ./main
Input Integer: 67
The number is odd.

information-processing-1_10th-class/programs/p312 on  main [!?] via C v14.2.1-gcc via ❄  impure (LaTeX-Environment-env) took 2s
❯
```

4　演習 3-13

教科書の List 3-18 を Switch 文で書き換えたプログラム。

4.1　コードリスティング

演習 3-13

```c
#include <stdio.h>

int main(void) {
    int month;

    printf("月を入力してください:␣");
    scanf("%d", &month);

    switch (month) {
        case 12:
        case 1:
        case 2:
            printf("%d月は冬です。\n", month);
            break;
        case 3:
        case 4:
        case 5:
            printf("%d月は春です。\n", month);
            break;
        case 6:
        case 7:
        case 8:
            printf("%d月は夏です。\n", month);
            break;
        case 9:
        case 10:
        case 11:
            printf("%d月は秋です。\n", month);
            break;
        default:
            puts("不明な月です。");
    }

    return 0;
}
```

4.2　実行結果

```
information-processing-1_10th-class/programs/p313 on  main [!?] via C v14.2.1-gcc via ❄  impure (LaTeX-Environment-env)
❯ gcc -Wall main.c -o main
main.c: 関数 ‘main’ 内：
main.c:7:5: 警告：ignoring return value of ‘scanf’ declared with attribute ‘warn_unused_result’ [-Wunused-result]
    7 |     scanf("%d", &month);
      |     ^~~~~~~~~~~~~~~~~~~

information-processing-1_10th-class/programs/p313 on  main [!?] via C v14.2.1-gcc via ❄  impure (LaTeX-Environment-env)
❯ ./main
月を入力してください：3
3月は春です。

information-processing-1_10th-class/programs/p313 on  main [!?] via C v14.2.1-gcc via ❄  impure (LaTeX-Environment-env)
❯ ./main
月を入力してください：5
5月は春です。

information-processing-1_10th-class/programs/p313 on  main [!?] via C v14.2.1-gcc via ❄  impure (LaTeX-Environment-env)
❯ ./main
月を入力してください：11
11月は秋です。

information-processing-1_10th-class/programs/p313 on  main [!?] via C v14.2.1-gcc via ❄  impure (LaTeX-Environment-env) took 4s
❯ ./main
月を入力してください：12
12月は冬です。

information-processing-1_10th-class/programs/p313 on  main [!?] via C v14.2.1-gcc via ❄  impure (LaTeX-Environment-env)
❯ ./main
月を入力してください：68
不明な月です。

information-processing-1_10th-class/programs/p313 on  main [!?] via C v14.2.1-gcc via ❄  impure (LaTeX-Environment-env) took 3s
❯
```

5　簡易電卓

2 つの実数を入力し、四則演算を指定し、小数点 6 桁で結果を表示する。0 div も考慮すること。

5.1　コードリスティング

簡易電卓

```c
#include <stdio.h>

#define ADD 1
#define SUB 2
#define MUL 3
#define DIV 4

int main(void) {
    double a, b;
    int op;

    printf("Input␣first␣number:␣");
    scanf("%lf", &a);
    printf("Input␣second␣number:␣");
    scanf("%lf", &b);

    printf("Select␣Operation:\n"
           "[1]:␣Addition\n"
           "[2]:␣Subtraction\n"
           "[3]:␣Multiplication\n"
           "[4]:␣Division\n"
           ">␣");
    scanf("%d", &op);

    switch (op) {
        case ADD:
            printf("ANS:␣%lf\n", a + b);
            break;
        case SUB:
            printf("ANS:␣%lf\n", a - b);
            break;
        case MUL:
            printf("ANS:␣%lf\n", a * b);
            break;
```

```c
        case DIV:
            if (b == 0.0) {
                puts("Zero Division");
                return 1;
            }
            printf("ANS: %lf\n", a / b);
            break;
        default:
            puts("Undefined Operation");
            return 1;
    }

    return 0;
}
```

5.2　実行結果

```
information-processing-1_10th-class/programs/calc on  main [!?] via C v14.2.1-gcc via ❄  impure (LaTeX-Environment-env)
❯ gcc -Wall main.c -o main
main.c: 関数 ‘main’ 内:
main.c:13:5: 警告: ignoring return value of ‘scanf’ declared with attribute ‘warn_unused_result’ [-Wunused-result]
   13 |     scanf("%lf", &a);
      |     ^~~~~~~~~~~~~~~~
main.c:15:5: 警告: ignoring return value of ‘scanf’ declared with attribute ‘warn_unused_result’ [-Wunused-result]
   15 |     scanf("%lf", &b);
      |     ^~~~~~~~~~~~~~~~
main.c:23:5: 警告: ignoring return value of ‘scanf’ declared with attribute ‘warn_unused_result’ [-Wunused-result]
   23 |     scanf("%d", &op);
      |     ^~~~~~~~~~~~~~~~

information-processing-1_10th-class/programs/calc on  main [!?] via C v14.2.1-gcc via ❄  impure (LaTeX-Environment-env)
❯ ./main
Input first number: 12.3
Input second number: 3
Select Operation:
[1]: Addition
[2]: Subtraction
[3]: Multiplication
[4]: Division
> 1
ANS: 15.300000

information-processing-1_10th-class/programs/calc on  main [!?] via C v14.2.1-gcc via ❄  impure (LaTeX-Environment-env)
 took 4s
❯ ./main
Input first number: 23.1
Input second number: 2.1
Select Operation:
[1]: Addition
[2]: Subtraction
[3]: Multiplication
[4]: Division
> 2
ANS: 21.000000

information-processing-1_10th-class/programs/calc on  main [!?] via C v14.2.1-gcc via ❄  impure (LaTeX-Environment-env)
 took 8s
❯ ./main
Input first number: 43.1
Input second number: 2.2
Select Operation:
[1]: Addition
[2]: Subtraction
[3]: Multiplication
[4]: Division
> 3
ANS: 94.820000
```

```
information-processing-1_10th-class/programs/calc on  main [!?] via C v14.2.1-gcc via ❄  impure (LaTeX-Environment-env)
 took 9s
❯ ./main
Input first number: 32.4
Input second number: 4
Select Operation:
[1]: Addition
[2]: Subtraction
[3]: Multiplication
[4]: Division
> 4
ANS: 8.100000

information-processing-1_10th-class/programs/calc on  main [!?] via C v14.2.1-gcc via ❄  impure (LaTeX-Environment-env)
 took 17s
❯ ./main
Input first number: 1
Input second number: 0
Select Operation:
[1]: Addition
[2]: Subtraction
[3]: Multiplication
[4]: Division
> 4
Zero Division

information-processing-1_10th-class/programs/calc on  main [!?] via C v14.2.1-gcc via ❄  impure (LaTeX-Environment-env)
 took 10s
❯ ./main
Input first number: 12
Input second number: 2
Select Operation:
[1]: Addition
[2]: Subtraction
[3]: Multiplication
[4]: Division
> 5
Undefined Operation

information-processing-1_10th-class/programs/calc on  main [!?] via C v14.2.1-gcc via ❄  impure (LaTeX-Environment-env)
 took 3s
❯
```

A　付録

A.1　Deep Dive - if と switch の根本的な違い

初学者は if 文と switch 文の使い分けで困まる時があるようだ。前述したとおり、どちらも条件分岐を行うことができる。

しかし、2 つには文法以外に明確な違いがある。それは、switch 文のほうが「**圧倒的**」に処理速度が速いというところである。まず、その処理速度の違いを見てみる。

以下の Python スクリプトで 10 万個の条件文を含む C ソースコードを生成する：

if 文 10 万個

```python
with open("./ifs.c", mode='w') as file:
    num = 100000
    file.write("int␣comp(int␣x)␣{\n␣␣␣␣if␣(x␣==␣0)␣{\n␣␣␣␣␣␣␣␣return␣0;\n␣␣␣␣}␣")
    for i in range(1,num-1):
        s = f"else␣if␣(x␣==␣{i})␣{{\n␣␣␣␣␣␣␣␣return␣{i};\n␣␣␣␣}}␣"
        file.write(s)
    file.write(f"else␣{{\n␣␣␣␣␣␣␣␣return␣{num-1};\n␣␣␣␣}}\n}}")
    file.write(f"\n\nint␣main(void)␣{{\n␣␣␣␣for␣(int␣i␣=␣0;␣i␣<␣{num};␣i++)␣{{␣␣␣␣␣␣␣␣\n␣␣␣␣␣␣␣␣comp(i);\n␣␣␣␣}}\n␣␣␣␣return␣0;\n}}\n")
```

switch 文 10 万個

```python
with open("./switch-case.c", mode='w') as file:
    num = 100000
    file.write("int␣comp(int␣x)␣{\n␣␣␣␣switch␣(x)␣{\n")
    for i in range(num):
        s = f"␣␣␣␣␣␣␣␣case␣{i}:␣return␣{i};\n"
        file.write(s)
    file.write("␣␣␣␣}\n}")
    file.write(f"\n\nint␣main(void)␣{{\n␣␣␣␣for␣(int␣i␣=␣0;␣i␣<␣{num};␣i++)␣{{\n␣␣␣␣␣␣␣␣comp(i);\n␣␣␣␣}}\n␣␣␣␣return␣0;\n}}\n")
```

これらを実行し、C ソースコードを生成する：

```
$ python gen-if.py
$ python gen-switch.py
```

以下が生成されたソースコードの一部始終：

if 文 10 万個のソースコード

```
int comp(int x) {
    if (x == 0) {
        return 0;
    } else if (x == 1) {
        return 1;
    } else if (x == 2) {
        return 2;
    } else if (x == 3) {
    } else if (x == 99997) {
        return 99997;
    } else if (x == 99998) {
        return 99998;
    } else {
        return 99999;
    }
}

int main(void) {
    for (int i = 0; i < 100000; i++) {
        comp(i);
    }
    return 0;
}
```

switch 文 10 万個のソースコード

```
int comp(int x) {
    switch (x) {
        case 0: return 0;
        case 1: return 1;
        case 2: return 2;
        case 3: return 3;
        case 4: return 4;
        case 5: return 5;
        case 99995: return 99995;
        case 99996: return 99996;
        case 99997: return 99997;
        case 99998: return 99998;
        case 99999: return 99999;
    }
}
```

```
100005
100006 int main(void) {
100007     for (int i = 0; i < 100000; i++) {
100008         comp(i);
100009     }
100010     return 0;
100011 }
```

```
1 // 検証に影響する最適化を無効にしてコンパイルする
2 $ gcc -O0 ifs.c -o ifs
3 $ gcc -O0 switch-case.c -o switch-case
```

生成物をタイマに通して実行する：

```
1  $ time ./ifs
2
3  real    0m59.045s
4  user    0m58.444s
5  sys     0m0.001s
6
7  $ time ./switch-case
8
9  real    0m0.006s
10 user    0m0.004s
11 sys     0m0.002s
```

実行結果から switch 文の方が if 文の約 9840 倍も高速であると分かる。

では、なぜ switch 文の方が速いのか。それは、比較演算の数にある。

if 文は記述した数と同じ数の比較演算が使われている。一方、switch 文は比較に関する演算は一度しか行われない。

それぞれのバイナリを Intel 記法 [1] で逆アセンブルする：

```
1 $ objdump -Mintel -d ifs
2 $ objdump -Mintel -d switch-case
```

if 文 10 万個の comp 関数の逆アセンブル (抜粋)

```
    97  0000000000401106 <comp>:
    98    401106:   55                    push   rbp
    99    401107:   48 89 e5              mov    rbp,rsp
   100    40110a:   89 7d fc              mov    DWORD PTR [rbp-0x4
   ],edi
   101    40110d:   83 7d fc 00           cmp    DWORD PTR [rbp-0x4
   ],0x0
```

```
   102    401111:   75 0a                   jne    40111d <comp+0x17>
   103    401113:   b8 00 00 00 00          mov    eax,0x0
   104    401118:   e9 2a fc 1c 00          jmp    5d0d47 <comp+0
   x1cfc41>
   105    40111d:   83 7d fc 01             cmp    DWORD PTR [rbp-0x4
   ],0x1
   106    401121:   75 0a                   jne    40112d <comp+0x27>
   107    401123:   b8 01 00 00 00          mov    eax,0x1
   108    401128:   e9 1a fc 1c 00          jmp    5d0d47 <comp+0
   x1cfc41>
   109    40112d:   83 7d fc 02             cmp    DWORD PTR [rbp-0x4
   ],0x2
   110    401131:   75 0a                   jne    40113d <comp+0x37>
   111    401133:   b8 02 00 00 00          mov    eax,0x2
   112    401138:   e9 0a fc 1c 00          jmp    5d0d47 <comp+0
   x1cfc41>
400089    5d0d22:   81 7d fc 9d 86 01 00    cmp    DWORD PTR [rbp-0x4
   ],0x1869d
400090    5d0d29:   75 07                   jne    5d0d32 <comp+0
   x1cfc2c>
400091    5d0d2b:   b8 9d 86 01 00          mov    eax,0x1869d
400092    5d0d30:   eb 15                   jmp    5d0d47 <comp+0
   x1cfc41>
400093    5d0d32:   81 7d fc 9e 86 01 00    cmp    DWORD PTR [rbp-0x4
   ],0x1869e
400094    5d0d39:   75 07                   jne    5d0d42 <comp+0
   x1cfc3c>
400095    5d0d3b:   b8 9e 86 01 00          mov    eax,0x1869e
400096    5d0d40:   eb 05                   jmp    5d0d47 <comp+0
   x1cfc41>
400097    5d0d42:   b8 9f 86 01 00          mov    eax,0x1869f
400098    5d0d47:   5d                      pop    rbp
400099    5d0d48:   31 ff                   xor    edi,edi
400100    5d0d4a:   c3                      ret
```

switch 文 10 万個の comp 関数の逆アセンブル (抜粋)

```
    97  0000000000401106 <comp>:
    98    401106:   55                      push   rbp
    99    401107:   48 89 e5                mov    rbp,rsp
   100    40110a:   89 7d fc                mov    DWORD PTR [rbp-0x4
   ],edi
   101    40110d:   81 7d fc 9f 86 01 00    cmp    DWORD PTR [rbp-0x4
   ],0x1869f
```

```
   102    401114:   0f 87 2a 42 0f 00       ja     4f5344 <comp+0
   xf423e>
   103    40111a:   8b 45 fc               mov    eax,DWORD PTR [rbp
   -0x4]
   104    40111d:   48 8d 14 85 00 00 00    lea    rdx,[rax*4+0x0]
   105    401124:   00
   106    401125:   48 8d 05 d8 4e 0f 00    lea    rax,[rip+0xf4ed8]
         # 4f6004 <_IO_stdin_used+0x4>
   107    40112c:   8b 04 02               mov    eax,DWORD PTR [rdx+
   rax*1]
   108    40112f:   48 98                  cdqe
   109    401131:   48 8d 15 cc 4e 0f 00    lea    rdx,[rip+0xf4ecc]
         # 4f6004 <_IO_stdin_used+0x4>
   110    401138:   48 01 d0               add    rax,rdx
   111    40113b:   ff e0                  jmp    rax
   112    40113d:   b8 00 00 00 00         mov    eax,0x0
   113    401142:   e9 fd 41 0f 00         jmp    4f5344 <comp+0
   xf423e>
   114    401147:   b8 01 00 00 00         mov    eax,0x1
   115    40114c:   e9 f3 41 0f 00         jmp    4f5344 <comp+0
   xf423e>
   116    401151:   b8 02 00 00 00         mov    eax,0x2
   117    401156:   e9 e9 41 0f 00         jmp    4f5344 <comp+0
   xf423e>
200106    4f532f:   b8 9d 86 01 00         mov    eax,0x1869d
200107    4f5334:   eb 0e                  jmp    4f5344 <comp+0
   xf423e>
200108    4f5336:   b8 9e 86 01 00         mov    eax,0x1869e
200109    4f533b:   eb 07                  jmp    4f5344 <comp+0
   xf423e>
200110    4f533d:   b8 9f 86 01 00         mov    eax,0x1869f
200111    4f5342:   eb 00                  jmp    4f5344 <comp+0
   xf423e>
200112    4f5344:   5d                     pop    rbp
200113    4f5345:   31 d2                  xor    edx,edx
200114    4f5347:   31 ff                  xor    edi,edi
200115    4f5349:   c3                     ret
200116
```

`if` 文では `cmp`、`jne`、`mov`、`jmp` が並んでおり、上から順に値を比較していき、等価が確認されたらアドレス `0x5d0d47` に飛び関数から抜け出る。この時、値が大きいと関数の深部まで進めないと等価が確認されないので時間がかかる。言い換えれば比較回数は引数の値に比例するといえる ($O(n)$)。

一方 switch 文では、cmp と jne のような比較命令は関数の始め以外全く見あたらない。

ここで、関数の始めの 10 数個の命令を見ると、4 行目では 16 進数 0x1869f(99999) とスタック上にロードされた引数を比較している。次の ja 命令では前行の結果に応じて関数の終わり付近のアドレス 0x4f5344 へ飛ぶ。この命令は「〇〇以上であるか」を検証するので、この場合は等価検証する値の範囲である 0〜99999 に引数が含まれているかを検証し、そうでなければその時点で関数から抜け出る処理を行っている。

13 行目まではアドレスに関する算術を行っている。rdx レジスタには引数の 4 倍の値を代入し、rax レジスタには 0x4f6004 というマジックナンバーをロードする。このマジックナンバーは整数型配列のアドレスとなっている。この配列は更にはコンパイル時に計算されたマジックナンバーがリトルエンディアンで格納されてる。

次に 32 ビットの eax レジスタに rdx レジスタと rax レジスタの和が示めすアドレスが指している値を代入している。つまり、引数をインデックスとし、それに 32 ビット整数型のサイズである 4 バイトを掛け、配列のアドレスに足すことで配列にアクセスしている。例として引数が 2 の時、アドレスは 0x4f6004 + 0x4 * 0x2 = 0x4f600c となり、そこから 4 バイト分を読むと 4db1f0ff となり、リトルエンディアンとして変換すると 0xfff0b14d となる。

cdqe 命令は eax レジスタをサイズが倍の rax レジスタに符号拡張している。この時 rax レジスタには負の値が入っている。引数が 2 の時、0xfff0b14d を符号拡張すると 0xfffffffffff0b14d となり、この数の 2 の補数で負の数と解釈すると -0xf4eb3 となる。

rdx レジスタに 0x4f6004 を再びロードし、rax レジスタに rdx レジスタの値を加算する。引数が 2 の場合、-0xf4eb3 + 0x4f6004 = 0x401151 となる。

この後 rax レジスタに代入されているアドレスに飛ぶが、この時のアドレスは case ラベルで指定した処理に対応している。

これはハッシュマップに似たもので、どんな値でも処理速度は変化することはない ($O(1)$)。

この魔法のような演算のおかげで、switch 文は比較せずとも分岐を行うことができ、switch 文で扱える値の型が整数型なのも納得いくだろう。

結局 if 文と switch 文どちらを使うべきかは分岐の数と実行環境による。

分岐の数が少なく、処理速度はあまり求められない場合は if 文、膨大な分岐を必要とし、かつ処理速度に制限がある場合は switch 文が有用である。

参考文献

[1]Intel Corporation. *Intel 64 and IA-32 Architectures Software Developer's Manual*. 03/2025. URL: https://software.intel.com/en-us/download/intel-64-and-ia-32-architectures-sdm-combined-volumes-1-2a-2b-2c-2d-3a-3b-3c-3d-and-4 (visited on 06/24/2025).

[2]cppreference. *switch statement*. 01/2018. URL: https://en.cppreference.com/w/c/language/switch.html (visited on 06/19/2025).

Last Compiled(UN*X Time): 1750768541