レポート

システムソフトウェア特論 課題#11

担当教員 久野靖

提出日 2020 年 2 月 17 日

電気通信大学
情報理工学研究科
情報・ネットワーク工学専攻

学籍番号 1931010

伊勢大平

1　課題

選んだ演習問題は 1 の a，b，c である．
第 2 節 (課題の再掲) で，課題を要約して説明する．

2　課題の再掲

2 つの例題をそのまま動かす．2 つの例題を動かしたら，次のことを行う．

a. 1 番目の例題で，演算を足し算以外のものにして動かす．(足し算以外の演算として，引き算を動かすこととした)

b. 2 番目の例題で，演算を「配列要素の合計」にして動かす．

c. 2 番目の例題を改造して「配列を全部クリアする」,「配列の要素を全部 1 つずつ前に動かす（先頭にあったものは末尾に移す)」を作る．

1 番目の例題のソースコードは，次のリスト 1(sam1.c) とリスト 2(sam1sub1.s) である．

リスト 1: 1 番目の例題の C ソースコード (sam1.c)

```
1 #include <stdio.h>
2 int sub(int a, int b);
3 int main(void) {
4   printf("%d\n", sub(3, 9)); return 0;
5 }
```

リスト 2: 1 番目の例題のアセンブリのソースコード (sam1sub1.s)

```
1         .text
2         .globl sub
3         .type sub, @function
4 sub: pushq %rbp
5         movq %rsp, %rbp
6         movl %edi, %eax
7         addl %esi, %eax
8         leave
9         ret
```

2 番目の例題のソースコードは，次のリスト 3(sam2.c) とリスト 4(sam2sub1.s) である．

リスト 3: 2 番目の例題の C ソースコード (sam2.c)

```
1 #include <stdio.h>
2 int sub(int n, int a[]);
3 int b[] = { 5, 1, 6, 8, 2, 7, 4, 3 };
4 int main(void) {
5   printf("%d\n", sub(8, b)); return 0;
6 }
```

リスト 4: 2 番目の例題のアセンブリのソースコード (sam2sub1.s)

```
1         .text
2         .globl sub
3         .type sub, @function
4 sub: pushq %rbp
```

```
5         movq %rsp, %rbp
6         movl 0(%rsi), %eax
7  .L1: dec %edi
8         cltq
9         jl .L2
10        movl 0(%rsi,%rdi,4), %edx
11        cmpl %edx, %eax
12        jge .L1
13        movl %edx, %eax
14        jmp .L1
15 .L2: leave
16        ret
```

3 方針

例題のソースコードの，sub の定義を変更する．

演習 c では，sub の定義の変更がうまくできたか確認できるように C のソースコードも変更する．

4 成果物

4.1 プログラム

引き算のプログラム (演習 a) はリスト 2 のように書いた.

リスト 5: 引き算のソースコード (sam1sub2.s)

```
        .text
        .globl sub
        .type sub, @function
sub: pushq %rbp
        movq %rsp, %rbp
        movl %edi, %eax
        subl %esi, %eax
        leave
        ret
```

配列要素の合計のプログラム (演習 b) はリスト 6 のように書いた.

リスト 6: 配列要素の合計のソースコード (sam2sub2.s)

```
        .text
        .globl sub
        .type sub, @function
sub: pushq %rbp
        movq %rsp, %rbp
        movl $0, %eax
.L1: dec %edi
        cltq
        jl .L2
        movl 0(%rsi,%rdi,4), %edx
        addl %edx, %eax
        jmp .L1
.L2: leave
        ret
```

演習 c の C のソースコードはリスト 7 のように書いた.

リスト 7: 演習 c の C ソースコード (sam3.c)

```
#include <stdio.h>
int sub(int n, int a[]);
int b[] = { 5, 1, 6, 8, 2, 7, 4, 3 };
int main(void) {
  int i;
  for(i=0;i<8;i++){
    printf("b[%d]=%d; ", i,b[i]);
  }
  printf("\n");
  sub(8,b);
  for(i=0;i<8;i++){
    printf("b[%d]=%d; ", i,b[i]);
  }
  printf("\n");
  return 0;
}
```

配列を全部クリアするプログラム (演習 c) はリスト 8 のように書いた.

リスト 8: 配列を全部クリアするソースコード (sam3sub1.s)

```
        .text
```

```
2         .globl sub
3         .type sub, @function
4  sub: pushq %rbp
5         movq %rsp, %rbp
6  .L1: dec %edi
7         cltq
8         jl .L2
9         movl $0, 0(%rsi,%rdi,4)
10        jmp .L1
11 .L2: leave
12        ret
```

配列の要素を全部 1 つずつ前に動かす (先頭にあったものは末尾に移す) プログラム (演習 c) はリスト 9 のように書いた.

リスト 9: 配列の要素を全部 1 つずつ前に動かすソースコード (sam3sub2.s)

```
1         .text
2         .globl sub
3         .type sub, @function
4  sub: pushq %rbp
5         movq %rsp, %rbp
6         movl 0(%rsi), %eax
7  .L1: dec %edi
8         cltq
9         jl .L2
10        movl 0(%rsi,%rdi,4), %edx
11        movl %eax, 0(%rsi,%rdi,4)
12        movl %edx, %eax
13        jmp .L1
14 .L2: leave
15        ret
```

4.2 プログラムの説明

リスト 5(引き算のソースコード) について説明する．リスト 2(1 番目の例題の足し算のプログラム) の 7 行目の命令 addl を subl に変えただけである.

リスト 6(配列要素の合計のソースコード) について説明する．レジスタ%rdi を配列の添字として用いる．レジスタ%rsi には配列の先頭を指すアドレスが格納されている．まず 6 行目で 0 をレジスタ%eax に格納している．7 行目から 12 行目までで，%rdi に配列の要素数を入れた状態から，%rdi の値が 0 になるまで,「%rdi = %rdi - 1，%eax = 配列の%rdi 番目 + %eax」を繰り返すようにする．これにより，返値となるレジスタ%eax に配列要素の合計が格納される.

リスト 7(演習 c の C ソースコード) について説明する．3 行目のように定義された配列をまずそのまま表示し，改行してから sub 関数を実行した後の配列を表示するようにした．これにより，sub 関数を実行したことで配列がどのように変化したかを見ることができる.

リスト 8(配列を全部クリアするソースコード) について説明する．レジスタ%rdi を配列の添字として用いる．レジスタ%rsi には配列の先頭を指すアドレスが格納されている．6 行目から 10 行目までで，%rdi に配列の要素数を入れた状態から，%rdi の値が 0 になるまで,「%rdi = %rdi - 1，配列の%rdi 番目 = 0」を繰り返すようにする.

リスト 9(配列の要素を全部 1 つずつ前に動かすソースコード) について説明する．レジスタ%rdi を配列の添字として用いる．レジスタ%rsi には配列の先頭を指すアドレスが格

納されている．まず 6 行目で配列の先頭の要素をレジスタ%eax に格納している．7 行目から 13 行目までで，%rdi に配列の要素数を入れた状態から，%rdi の値が 0 になるまで，「%rdi = %rdi - 1，%edx = 配列の%rdi 番目，配列の%rdi 番目 = %eax，%eax = %edx」を繰り返すようにする．

4.3 実行例

リスト 10 は 1 番目の例題および演習 a の実行結果である．1 行目から 3 行目までで 1 番目の例題の実行結果を示し，4 行目から 6 行目までで演習 a の実行結果を示している．1 番目の例題の実行結果では 3+9=12 を，演習 a の実行結果では 3-9=-6 を出力しているため，正しく動作していると分かる．

リスト 10: 1 番目の例題および演習 a の実行結果

```
1 $ gcc sam1.c sam1sub1.s
2 $ ./a.out
3 12
4 $ gcc sam1.c sam1sub2.s
5 $ ./a.out
6 -6
```

リスト 11 は 2 番目の例題および演習 b の実行結果である．1 行目から 3 行目までで 2 番目の例題の実行結果を示し，4 行目から 6 行目までで演習 b の実行結果を示している．2 番目の例題の実行結果では配列の最大値 8 を，演習 b の実行結果では配列要素の合計 5+1+6+8+2+7+4+3=36 を出力しているため，正しく動作していると分かる．

リスト 11: 2 番目の例題および演習 b の実行結果

```
1 $ gcc sam2.c sam2sub1.s
2 $ ./a.out
3 8
4 $ gcc sam2.c sam2sub2.s
5 $ ./a.out
6 36
```

リスト 12 は演習 c の 2 つの実行結果である．1 行目から 4 行目までは「配列を全部クリアする」プログラムの実行結果を示している．3 行目は sub 関数の実行前の配列を出力したものである．4 行目は sub 関数の実行後の配列を出力したものである．sub 関数の実行後は配列の要素が全て 0 になっているため，正しく動作していると分かる．5 行目から 8 行目までは「配列の要素を全部 1 つずつ前に動かす (先頭にあったものは末尾に移す)」プログラムの実行結果を示している．7 行目は sub 関数の実行前の配列を出力したものである．8 行目は sub 関数の実行後の配列を出力したものである．sub 関数の実行後は配列の要素がちゃんと移動しているため，正しく動作していると分かる．

リスト 12: 演習 c の 2 つの実行結果

```
1 $ gcc sam3.c sam3sub1.s
2 $ ./a.out
3 b[0]=5; b[1]=1; b[2]=6; b[3]=8; b[4]=2; b[5]=7; b[6]=4; b[7]=3;
4 b[0]=0; b[1]=0; b[2]=0; b[3]=0; b[4]=0; b[5]=0; b[6]=0; b[7]=0;
5 $ gcc sam3.c sam3sub2.s
```

```
$ ./a.out
b[0]=5; b[1]=1; b[2]=6; b[3]=8; b[4]=2; b[5]=7; b[6]=4; b[7]=3;
b[0]=1; b[1]=6; b[2]=8; b[3]=2; b[4]=7; b[5]=4; b[6]=3; b[7]=5;
```

5　考察

今回，演習 c の「配列の要素を全部 1 つずつ前に動かす」という処理は，ループで 1 つずつ動かすことで実装した．これは配列の要素数を n とすると $O(n)$ である．しかし，配列の先頭を指すポインタを一つ後ろにずらし，先頭の要素だけ末尾にくっつければ，$O(1)$ で動かすことができたかもしれないと考察した．レポートを書いているときに気づいて試してみることができなかったが，今後，実際に実装できるか試してみたいと考えた．

6　アンケートの解答

Q1 強い型の言語と弱い型の言語のどちらが好みですか．またそれはなぜ．

A1 強い型の言語の方が好きです．弱い型だと型宣言がない分コードが書きやすくなっても，読み返すときに読みづらくなったり，型が分からなくなったりします．変数にはなるべくその意味が分かりやすくなるように名前をつけますが，型が明記されている方がより意味が明確になり，読みやすくなると思います．エラーが出て書き直すことや，しばらく時間が経ってから変更することを想定すれば，やはり型が明記されているなるべく読みやすい記述の方が良いと思います．

Q2 記号表と型検査の実装について学んでみて，どのように思いましたか．

A2 ローカル変数は定義された場所によってその変数の使える範囲が決まりますが，それがどういう原理で実現されているのか，記号表を学ぶことで知ることができました．

また，型検査について学んでみて，抽象構文木で実現できると知り，とてもシンプルだと思いました．

Q3 リフレクション（課題をやってみて気づいたこと），感想，要望など．

A3 main 関数から呼び出される関数を入力としてアセンブリ言語を出力する仕組みを学びましたが，main 関数はどうやって処理するのかも知りたいと思いました．長い間 C 言語を使っていますが，main 関数はどこから呼び出されて，何に対して「return 0;」しているのがまだ分かっていないので，ちゃんと勉強していきたいと思いました．

また，アセンブリ言語はどのようにして設計するのか興味を持ちました．