岐阜工業高等専門学校

第 3 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 (15(2 年生))

令和 7 年 04 月 30 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 (GCC) 14.2.1 20250207`
- C コンパイラフラグ: `-Wall <ソースコード名> -o <プログラム名>`

2　今回の構文

2.1　`scanf` 関数

`scanf` 関数は標準入力から文字列を読み取り、指定された書式に沿って解釈し、解釈結果の値を指定された場所に保存する。ここでいう場所とは変数のアドレスのことである。書式は `printf` 関数と同じものである。[1]

`scanf` 関数

```
1 scanf("<書式>", <変数1へのアドレス>, <変数2へのアドレス>, ...);
```

コラム：標準入出力関数の戻り値

標準入出力関数には `void` 型を返す関数はほとんど定義されていない。

表 1：主な標準入出力関数の戻り値 [4]

関数	型	概要
`scanf` 系	`int`	正常に読み込まれた変数の数、0 または `EOF` 定数はエラーとなる
`printf` 系	`int`	バッファーやストリームに書き込まれた文字の数、負の値はエラーとなる
`setbuf`, `rewind`, `clearerr`, `perror`	`void`	これらの関数のみ値を返さない、エラーになり得る処理ではないから

3　演習 1 ー 5

プロンプトから読み込んだ整数値に 13 を加えた値を表示するプログラム。

3.1　コードリスティング

演習 1 ー 5

```c
#include <stdio.h>

int main(void) {
    int x = 0;

    printf("Input integer: x = ");
    scanf("%d", &x);

    printf("Result: x + 13 = %d\n", x + 13);

    return 0;
}
```

3.2　実行結果

```
information-processing-1_3rd-class/programs/prog1 on  main [!?] via C v14.2.1-gcc
❯ gcc -Wall main.c -o main

information-processing-1_3rd-class/programs/prog1 on  main [!?] via C v14.2.1-gcc
❯ ./main
Input integer: x = 4
Result: x + 13 = 17

information-processing-1_3rd-class/programs/prog1 on  main [!?] via C v14.2.1-gcc took 4s
❯ ./main
Input integer: x = 67
Result: x + 13 = 80

information-processing-1_3rd-class/programs/prog1 on  main [!?] via C v14.2.1-gcc took 4s
❯
```

4　演習 1 ー 6

プロンプトから読み込んだ整数値から 7 を減じた値を表示するプログラム。

4.1　コードリスティング

演習 1 ー 6

```c
#include <stdio.h>

int main(void) {
    int x = 0;

    printf("Input integer: x = ");
    scanf("%d", &x);

    printf("Result: x - 7 = %d\n", x - 7);

    return 0;
}
```

4.2　実行結果

```
information-processing-1_3rd-class/programs/prog2 on  main [!?] via C v14.2.1-gcc
❯ gcc -Wall main.c -o main

information-processing-1_3rd-class/programs/prog2 on  main [!?] via C v14.2.1-gcc
❯ ./main
Input integer: x = 9
Result: x - 7 = 2

information-processing-1_3rd-class/programs/prog2 on  main [!?] via C v14.2.1-gcc took 3s
❯ ./main
Input integer: x = 67
Result: x - 7 = 60

information-processing-1_3rd-class/programs/prog2 on  main [!?] via C v14.2.1-gcc took 2s
❯
```

5　演習 1 ー 8

プロンプトから読み込んだ 2 つの整数値の積を表示するプログラム。

5.1　コードリスティング

演習 1 ー 8

```c
#include <stdio.h>

int main(void) {
    int n1;
    int n2;

    printf("Input␣two␣integers.\n");
    printf("Integer␣n1:␣");
    scanf("%d", &n1);
    printf("Integer␣n2:␣");
    scanf("%d", &n2);

    printf("The␣product␣of␣two␣integers␣is␣%d.", n1 * n2);

    return 0;
}
```

5.2　実行結果

```
information-processing-1_3rd-class/programs/prog3 on  main [!?] via C v14.2.1-gcc
❯ gcc -Wall main.c -o main

information-processing-1_3rd-class/programs/prog3 on  main [!?] via C v14.2.1-gcc
❯ ./main
Input two integers.
Integer n1: 3
Integer n2: 4
The product of two integers is 12.
information-processing-1_3rd-class/programs/prog3 on  main [!?] via C v14.2.1-gcc took 3s
❯ ./main
Input two integers.
Integer n1: 15
Integer n2: 8
The product of two integers is 120.
information-processing-1_3rd-class/programs/prog3 on  main [!?] via C v14.2.1-gcc took 10s
❯
```

6　演習 1 ー 9

プロンプトから読み込んだ 3 つの整数値の和を表示するプログラム。

6.1　コードリスティング

演習 1 ー 9

```c
#include <stdio.h>

int main(void) {
    int n1;
    int n2;
    int n3;

    printf("Input three integers.\n");
    printf("Integer n1: ");
    scanf("%d", &n1);
    printf("Integer n2: ");
    scanf("%d", &n2);
    printf("Integer n3: ");
    scanf("%d", &n3);

    printf("The sum of three integers is %d.", n1 + n2 + n3);

    return 0;
}
```

6.2　実行結果

```
information-processing-1_3rd-class/programs/prog4 on  main [!?] via C v14.2.1-gcc
❯ gcc -Wall main.c -o main

information-processing-1_3rd-class/programs/prog4 on  main [!?] via C v14.2.1-gcc
❯ ./main
Input three integers.
Integer n1: 3
Integer n2: 4
Integer n3: 5
The sum of three integers is 12.
information-processing-1_3rd-class/programs/prog4 on  main [!?] via C v14.2.1-gcc took 5s
❯ ./main
Input three integers.
Integer n1: 30
Integer n2: 15
Integer n3: 55
The sum of three integers is 100.
information-processing-1_3rd-class/programs/prog4 on  main [!?] via C v14.2.1-gcc took 9s
❯
```

7　考察：List 1-11

List 1-11 にて、入力に 3.14 や 0.5 などの小数を入力すると出力はどうなるか。

7.1　コードリスティング

考察：List 1-11

```c
#include <stdio.h>

int main(void) {
    int no;

    printf("Input an integer: ");
    scanf("%d", &no);

    printf("You inputed %d.\n", no);

    return 0;
}
```

7.2　実行結果

```
information-processing-1_3rd-class/programs/prog5 on  main [!?] via C v14.2.1-gcc
❯ gcc -Wall main.c -o main

information-processing-1_3rd-class/programs/prog5 on  main [!?] via C v14.2.1-gcc
❯ ./main
Input an integer: 3.14
You inputed 3.

information-processing-1_3rd-class/programs/prog5 on  main [!?] via C v14.2.1-gcc took 6s
❯ ./main
Input an integer: 4.8
You inputed 4.

information-processing-1_3rd-class/programs/prog5 on  main [!?] via C v14.2.1-gcc took 5s
❯ ./main
Input an integer: 0.12
You inputed 0.

information-processing-1_3rd-class/programs/prog5 on  main [!?] via C v14.2.1-gcc took 3s
❯
```

7.3　考察

入力した全ての小数が繰り下げられている。`%d` は整数しか表示できず、小数の場合は小数部を切り捨て、整数部のみ表示している。小数を表示したい場合は `printf` 関数と `scanf` 関数両方の書式を`%f` にする必要がある。

A　付録

A.1　Code Hardening - バッファーオーバーフロー対策 scanf 編

C 言語は比較的自由な言語だ。しかし自由には責任が伴う。

初学者からベテラン C プログラマーが誰でも 1 度はやらかしてしまう間違いとして文字列の読み込みによるバッファーオーバーフローがある。

バッファーオーバーフローは静的・動的に割り当てられたメモリー領域外に書き込むことで発生する。

今回の演習課題は整数値を読み込んだが、バッファーオーバフローは起こらないが、整数値オーバーフロー・アンダーフローという別の問題が起こる。しかし、このバグによる影響は比較的小さい。なぜなら、これらの整数は関数のスタック上の変数として決まったサイズでメモリーに割り当てられており、領域外への値の書き込みがないからである。

しかし、ユーザーが入力する文字列となると話が違ってくる。バッファーに過剰な量のデータが流入することバッファーオーバーフローが起こり、最悪の場合、バッファー外のデータを侵食・書き換えてしまい、プログラムが誤動作する。[3] 静的に割り当てられた変数のバッファーオーバーフローは対処が容易であるが、実行するまでサイズが不明な場合が多い動的に割り当てられた変数の対策はその限りではない。

文字列・配列への範囲外読み書きはユーザーやネットワーク要求からの入力を扱う際にはより注意する必要がある。開発者がすべてのユーザーが指示に従うと思い込むのは、はっきり言って**愚か**である。プログラムはすべてメモリ安全性を第一に考えて書かれるべきである。

実際、このような脆弱性がシステムの全権を取得できてしまう程の問題を簡単なエラーによって引き起こされる場合がある。例えば、「CVE-2021-3156」では Linux 等での非管理者ユーザーが管理者としてファイル編集できるようにする「sudoedit」コマンドでは、配列のサイズが 1 つずれただけでバッファーオーバーフローを発生させ、認証なしで管理者権限が付与されてしまうバグが存在した。[5]

Python や Java などの多くの高級言語は配列のサイズを超える場所への値の代入はエラーとなりコンパイル時やプログラム実行中に停止するように設計されている。しかし C 言語はそのような設計はプログラム実行中には施されていない。`-Wall` フラグを使用しても警告を出すだけで止めることはない。[1)]

高級言語の文字列は本体の文字列と共にその長さが記録されているが、C 言語はその長さは記録されず、代わりに終端文字が文字列の終わりを示めす。`scanf` や `gets` は終端文字を検出するまで読み込みを続ける。裏を返せば、終端文字を見つける前にバッファーが溢れているときでも、範囲外への書き込みを続けてしまうということにも繋がる。

次の例を考える：企業のシステムにて、ユーザーがパスワード (文字列) を入力し、正しければ社員データベースを操作するサブルーチーンに変移する。パスワードの入力を司る処理とパスワードを照合する処理は個別にサブルーチーンがあるものとする。ユーザーが入力したパスワードは動的に確保した 64 バイトのバッファー領域に書き込まれる。

1)　gcc など最近のコンパイラでコンパイルされたプログラムはスタックカナリアと呼ばれる一昔の炭鉱夫が一酸化炭素検出のために使われた鳥のカナリアの様にバッファーオーバーフローなどによって関数スタックが破壊された際にプログラムを強制終了させるコードを追加する機能が常に有効化されている物が存在する。なおこの機能はヒープ (動的確保されたメモリ領域) には適応されない。[2]

対策されていないシステムの例

```c
#include <stdio.h>
#include <stdlib.h>

#define PASSLEN 65

void databaseManagement();

int checkPassword(char* passwd) {
    char passwd[PASSLEN];

    scanf("%s", passwd);

    return isCorrectPassword(passwd);
}

int main(void) {
    printf("Database Login\nPassword: ");

    if (checkPassword())
        databaseManagement();

    return 0;
}
```

この時、システムを悪用したいと企てるハッカーが 65 バイト以上の文字列を渡し、うまくバッファーオーバーフローを引き起こさせると、パスワードの入力を司るサブルーチーンの戻りアドレスが書き換えられ、照合サブルーチーンを介さずに直接データベース管理サブルーチーンを実行させる。

scanf 関数での対策はいたってシンプルである：書式を"%<バッファーサイズ - 1>s"に変更するだけである。[2] これによって、読み出す文字数を制限し、バッファーオーバーフローを防ぐことができる。

もう一つの方法は C11 規格から追加された scanf_s 関数を代わりに使用することである。scanf_s 関数はオリジナルの scanf 関数に新たにバッファーのサイズを受け付ける引数を最後尾に追加し、エラーチェックをより厳密にしたもので、これにより前述の方法と併用しつつ、明示的にバッファーサイズを指定することができる。[1]

scanf 関数の対策例

```c
// scanf("%s", passwd);
scanf("64%s", passwd);
passwd[PASSLEN-1] = '\0'; // 終端文字の存在を保証する
// C11規格以降のみ
scanf_s("64%s", passwd, PASSLEN);
```

2)　この-1 で終端文字が入るスペースを残す。

References

[1]Eendy et al. *scanf, fscanf, sscanf, scanf_s, fscanf_s, sscanf_s*. 07/2022. URL: https://en.cppreference.com/w/c/io/fscanf (visited on 04/24/2025).

[2]Low Level. *what ever happend to buffer overflows?* 03/2023. URL: https://www.youtube.com/watch?v=z6gdQt8mjn4 (visited on 04/27/2025).

[3]Low Level. *why do hackers love strings?* 11/2022. URL: https://www.youtube.com/watch?v=fjMrDDj47E8 (visited on 04/27/2025).

[4]Space Mission. *Standard library header <stdio.h>*. 02/2025. URL: https://en.cppreference.com/w/c/header/stdio (visited on 04/27/2025).

[5]CVE Program. *CVE Record: CVE-2021-3156*. CVE Program. 09/2024. URL: https://www.cve.org/CVERecord?id=CVE-2021-3156 (visited on 04/30/2025).

Last Compiled(UN*X Time): 1745945094