プログラミング言語論
第２回

構造化プログラミング
処理の流れの制御

今週の内容

- 構造化プログラミングとは：
  - 展開的詳細化法と構造化定理
    - 構造化定理による処理フローの整理整頓
    - 関数を使って処理にまとまりを構築
    - 適切に処理や変数に名前やスコープを作ることで整理整頓
- 演習内容
  - 処理フローの制御

プログラミング言語の歴史

プログラミング言語の進化

低水準（低級）言語から高水準（高級）言語へ

- 機械語
  - 2進数で命令を表記
- アセンブラ言語（ニーモニック）
  - 機械語の数字をアルファベットなどの記号で表記
- 高級言語，高水準言語
  - 人間に理解しやすいように表記した言語
  - 知ってる言語は全部高級言語

※中間言語というものもある．ハードに依存しすぎず，コンパイルされた状態．
Ｊａｖａのバイトコードなど．ＬＬＶＭの発展により，再度注目されている．

古のプログラミング

- 入出力は紙テープ
- まともな入出力デバイスがない

- エディタなんて便利なものは無かった
- プログラムは手書きの時代！

1970年代のSFアニメ等に於いてはコンピュータが作動している場面のガジェットとしてオープンリールデータレコーダと共によく描かれていた。鑽孔テープのビット列をそのまま読み取って理解するのは、紙テープが記録媒体に使われていた時代のコンピュータ技師にとっては当たり前の技能であった。

https://ja.wikipedia.org/wiki/紙テープ

プログラミング言語の歴史
初期

1950年代半ば～1960年代半ば

アセンブラでの変数名の発明

- アセンブラ
- アドレス名に変数名をつけることで，プログラマが状態を把握しやすくする
- MOVE A [+120]
  - Aレジスタの値を120バイト後方のアドレスへコピー
- MOVE A data
  - 120バイト後方のアドレスをdataと名づけていた.

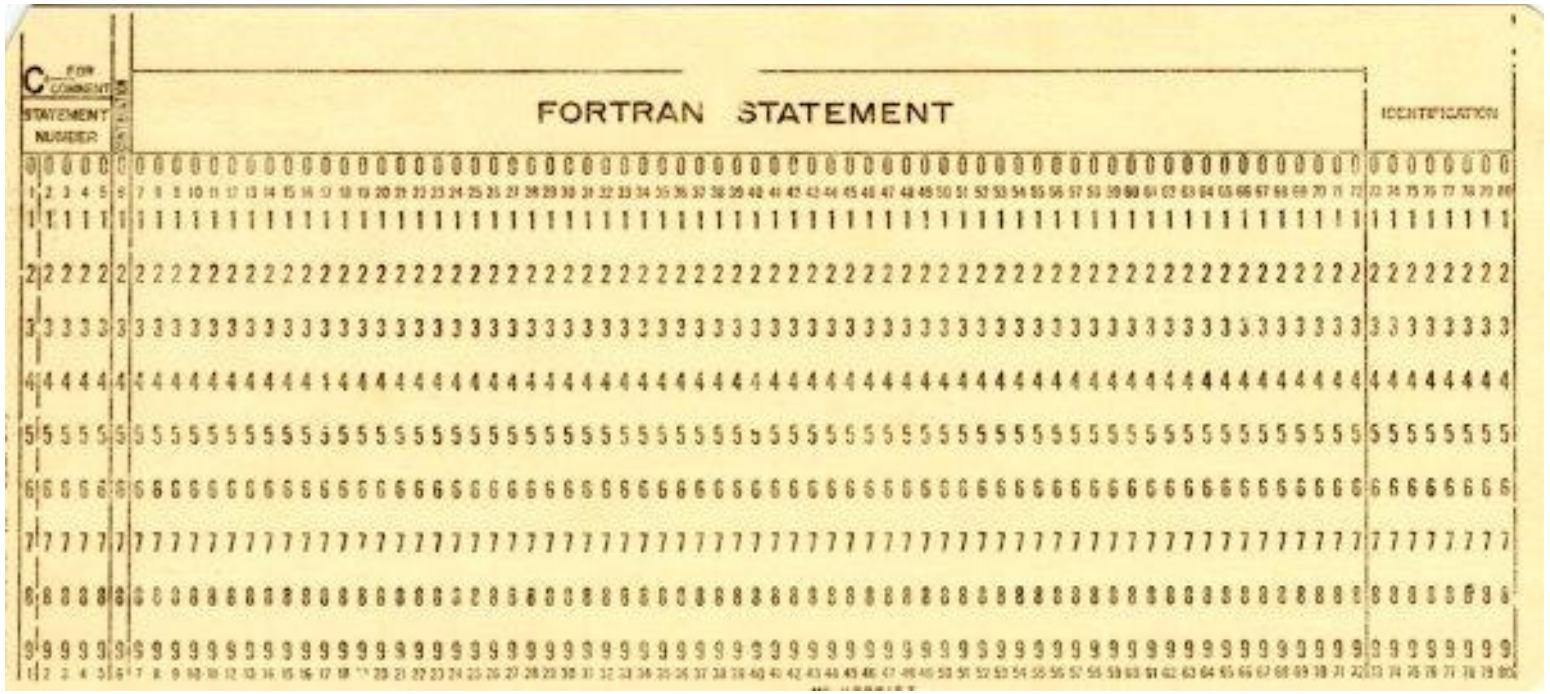
1957:FORTRAN (FORmula TRANslation)

- （事実上）初めて高水準言語.
  - 数式を翻訳するという意味
  - 「Xかける y たす z 」を「x*y+z」で記述
  - コンパイラの性能は高くなく，機械語は熟練したプログラマが直接入力したほうが速かった
  - 初期は制御文はIf, goto do(ループ)だけ
  - I,J,K,L,M,Nは強制的に全部int．それ以外はdouble．型宣言必要なし
- 科学技術用計算に向いた**手続き型プログラミング言語**
- 1957年：IBMのジョン・バッカスが考案（1954年開発開始）
- 現在でも使用されている言語
  - 普段使う？線形代数の計算はおそらくFortran
  - FORTRAN II,III.IV, 66/77，Fortran 90/95/2003/2008と進化
    - 78年以降はFortranと表記

手続き型プログラミング言語

- 「処理の流れ（手続き）」に重点を置く
- 命令（ステートメント）の列で構成
- 関数やサブルーチンで処理を分ける
- 代表的な言語
  - C, Pascal, Fortran, BASICなど
- 後に述べる構造化や非構造化の話との関係
  - 構造化しようがしまいが命令の羅列で動かくものは手続き型言語

FORTRANパンチカード



穴の開いている位置で英数字を表現（1

紙テープ
（連続）

1960:Lisp (LISt Processor)

- 1960年発表（1958年開発開始）
- 1950年代末～’60年代　米国マサチューセッツ工科大学のMcCarthy（マッカーシー）がリスト処理用の言語として開発
- 最古の**関数型言語**

  関数の組み合わせによってプログラミングを行う

  ｆ（ｘ），$g(x_1,x_2,\cdots)$
- ガベージコレクションの発明
- Common Lispとして現役．Emacsの拡張言語

ＬＩＳＰの例


```
(defun sum (n)
        (if (= n 1)
                1
                (+ n (sum (- n 1)))))
(sum 10)
```

1958:ALGOL（ALGOrithmic Language）

- 1958年開発（ALGOL58）
- 科学技術計算向けの言語
- 構造化プログラミングの概念の導入
  - **厳密な構文規約**を採用
  - プログラミング言語の構造として優れている
- 1960年　ALGOL60制定
  - 他のプログラミング言語（CやPASCALなど）に大きな影響を与える
  - 現在は滅亡
- 1968年　ALGOL68発表→言語規模の巨大化

ＡＬＧＯＬの例


```
begin
        integer i;
        real sum;
        sum := 0;
        for i := 1 step 1 until 10 do
                sum := sum + i;
        Print Real(sum)
end
```

プログラミング言語の歴史
中期

1960年代半ば～1970年代半ば

1971:Pascal

- 1971年　Wirth（ヴィルト）により発表
- ALGOL60の後継言語
- 特徴
  - 構造化プログラミングの実現、適切なデータ構造
  - 系統的な**プログラミング教育に最適**

    →　既存の言語とは異なる視点で開発
- 言語仕様の範囲を広げ過ぎない→言語処理系の効率化→ALGOL68の反省
- C言語などに影響
- 昔の教科書はPascal

Pascalの例


```
program test(input, output)
var i : integer,
        sum : real;
begin
        sum := 0;
        for i := 1 to 10 do
        begin
                sum := sum+i
        end;
        writeln(sum)
end.
```

1973:C言語

- 1973年　米国AT&Tのベル研究所　デニス・リッチーがUNIX開発用に設計
- 特徴
  - 汎用処理言語でありシステム記述言語でも汎用性の高い言語
  - システム記述はそれまではアセンブラ
  - **別名：高級アセンブラ**
  - 自由度，実行速度，コンパイル速度のために**安全性を無視**
    - Rustは安全性を最重視

９０年代：Cが分からないシステムエンジニアはいらないとまで

C言語の例


```
int main()
{
	int i;
	float sum = 0;
	for(i = 1; i <= 10; i++)
		sum += i;
	printf("%f¥n",sum);
}
```

構造化プログラミング

ソフトウェア危機

- 計算機の急激な発展に伴い，扱うシステムが複雑化し対応不可能に
- ソフトウェア危機（1968 NATOソフトウェア工学会議）
- プロジェクトの破綻
  - 予算超過
  - 期間の超過
  - 品質の低下
  - 仕様条件を満たさないソフトウェア
  - 管理不能状態のプロジェクト
  - 保守不能なコード

- 現代：デスマーチ（アンドリュー・ケーニッフィ，1995）
  - 長時間の残業や徹夜・休日出勤の常態化といった、プロジェクトメンバーに極端な負荷・過重労働を強い、通常の勤務状態では成功する可能性がとても低いプロジェクト、およびこれに参加させられている状況を主に指す.

ソフトウェア開発の最大の敵

- 全ての元凶は複雑性
  - 人間の脳はあまり複雑のものを扱うことが苦手
  - 複雑なものは，高い記憶力，理解力，集中力が要求される
    - 実働，1月１０００ステップと言われる？諸説ある.
      - ステップ：意味のある行数のこと
- 複雑化するソフトウェアには，整理整頓するため手段・プログラミング言語の進化が必要

プログラムが複雑になると？

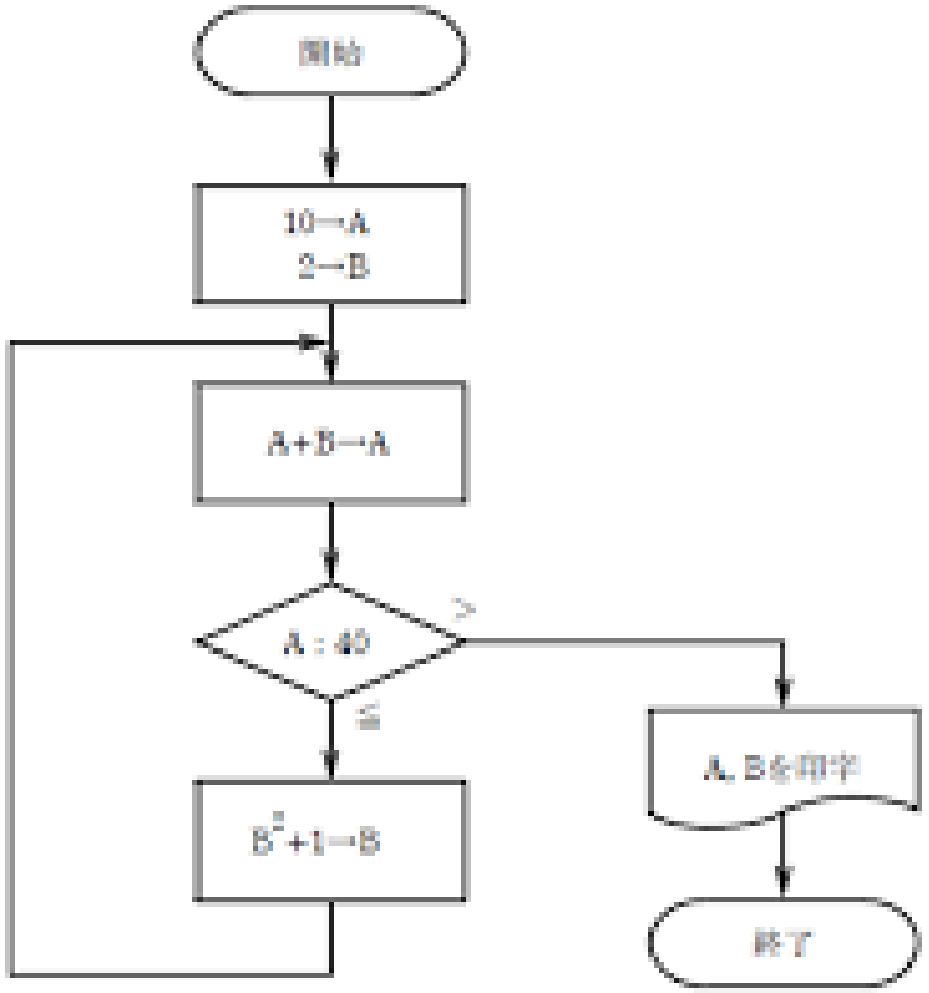
- バグ発生率が激増
- バグ修正のための工程も増加
- 仕様変更による修正も大変

動けば良い時代から，

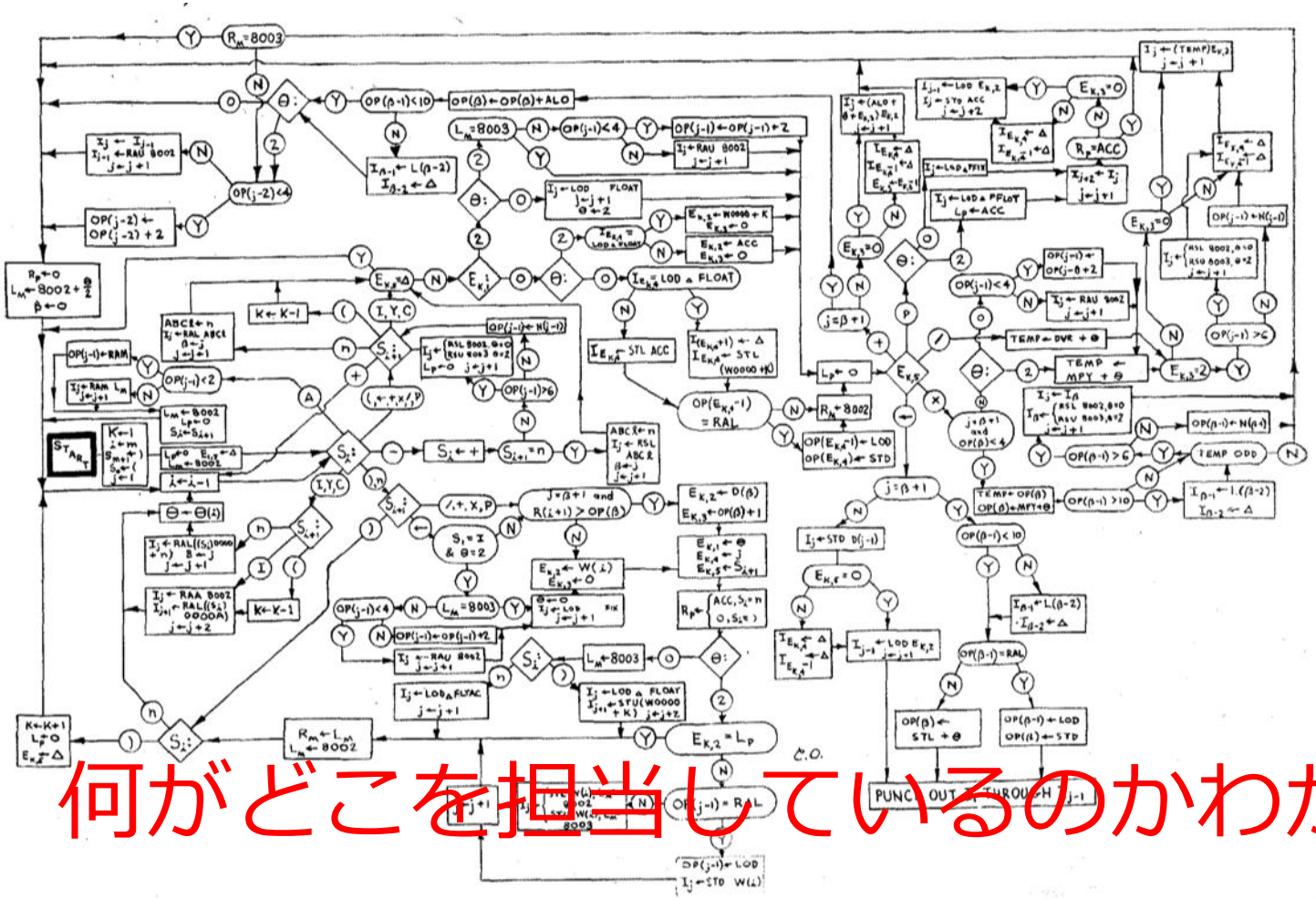
正しい，早い・速い，読みやすい，保守しやすい

コーディングが求められるように

昔はフローチャートで設計



昔はフローチャートで設計



何がどこを担当しているのかわかりにくい

Donald E. Knuth “RUNCIBLE?algebraic translation on a limited computer,” Communications of the ACM CACM Homepage archive　Volume 2 Issue 11, Nov. 1959 Pages 18-21

スパゲッティプログラムを書いてはイケナイ26

- プログラムのソースコードがそれを書いた本人にしか読めないことを指す。皿に盛られたスパゲッティのようにロジックが絡み合っていることから。

https://ja.wikipedia.org/wiki/スパゲティプログラム

原因：変なプログラムロジック、go to 文の濫用

Global 変数の濫用

構造化プログラミングの登場

- 大局的に検証し，全体から部分へ，部分から詳細へと構成する要素を順々に考えていく．
  - 段階的詳細化法
  - 構造化定理，Gotoレスプログラミング

段階的詳細化法

- Stepwise refinement（1971）
- Pascal開発者のニクラウス・ヴィルトを中心に提唱.

（順序的）プログラムは

1. データの入力
2. 操作
3. 結果の出力

の３つの作用に分解できる．すべてのプログラムを分解し，段階的に詳細化するとまとまりやすい．これは，関数化など普段から行ってること.

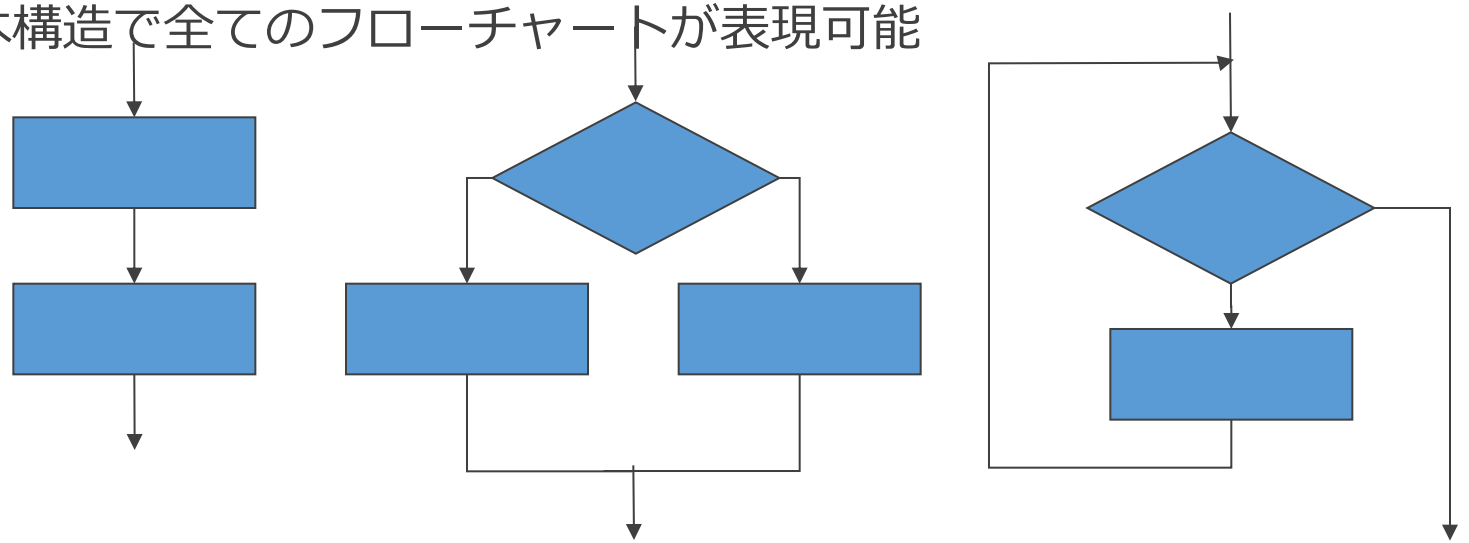
ダイクストラの主張

- エドガー・ダイクストラ
  - グラフ理論の最短経路問題におけるダイクストラ法の人
- Goto文は害悪だ！Goto文撲滅運動
  - E. Dijkstra. “Go To Statement Considered Harmful”, In *Communications of the ACM*, Vol. 11, Issue 3, ACM, New York, NY, USA, 1968, pp. 147-148.
  - あらゆる処理（アルゴリズム）は，Goto文無しに記述可能であることを証明されている
  - 当たり前のようにgoto文を使う時代

構造化定理

- 1966年：コラド・ベーム，ジュゼッペ・ヤコピーニ
  - この論文が出たころには構造化プログラミングなんていう単語もない．論文内でも構造化定理という単語は無い
- 逐次実行（順次実行，順次，順接，順構造）
- 分岐 if-then, if-then-else
- 繰り返し（反復，ループ）　while, repeat, for

の基本構造で全てのフローチャートが表現可能



- ただし入口，出口が１つのみをもつようなアルゴリズムに限る

構造化プログラミングとは

- プログラムを基本構造（逐次実行，分岐，ループ）＋アルファでつなげて表現すること

- いくつかの基本構造の意味のある塊をモジュールと呼ぶ
  - 別名：サブルーチン（FORTRAN），関数（C言語）
- プログラムはモジュールの組み合わせで表現可能

- プログラムの構造はモジュールを処理の順につなげて表現可能.
  個々のモジュールの詳細は内部に書けばよい

考え方（パラダイム）の変化

- プログラムとは1行１行命令を発行していくものである
  - 一直線の処理しか出来ない
  - 分岐するためにはジャンプ（gotoが必要）
  - プログラムは好きなようにフローチャートで設計
    - **非構造化プログラミング**
- 任意のフローチャートは1つの入り口，出口を持つ３つの記号で書け，そのように書けばフローチャートは分割できる
- 可読性・保守性の向上

制御構文

If文はなぜあるのか？

「無い場合と比べて可読性が向上するから」

- if以前
  - アセンブラにはif文がない
  - comp命令で比較はできる
  - 比較して，条件を満たしたらジャンプ命令(C言語だとgoto命令)
    - この命令は，EDSACの時代からあった

else，else if の登場

- if else, else if の関数は必須？
  - なくてもかける．しかし**可読性が大幅に劣化**
  - gotoとifだけで実現すると？

```
void iffunc(int x)
{
        if (x > 0)
        {
                printf("正の数¥n");
        }
        else if (x < 0)
        {
                printf("負の数¥n");
        }
        else
        {
                printf("ゼロ¥n");
        }
}
```

```
void noiffunc(int x)
{
        if (x <= 0) goto NOT_POSITIVE;
        printf("正の数¥n");
        goto END;
NOT_POSITIVE:
        if (x >= 0) goto NOT_NEGATIVE;
        printf("負の数¥n");
        goto END;
NOT_NEGATIVE:
        printf("ゼロ¥n");
END:
        return;
}
```

小テスト2

- Fizz Buzzを作れ.
  - コードを添付するかオンラインテキストにコードを貼り付けだけでOK. レポート等は必要ない.
- Fizz Buzzの詳細
  - 1からスタートして
    - 3で割り切れるときはFizz
    - 5で割り切れるときはBuzz
    - 3と5で割り切れるときはFizz Buzz
    - それ以外は数字

    を言うというゲーム. プログラミングの例題として有名
    - Ex
      - 1, 2, Fizz, 4, Buzz, Fizz, 7, 8, Fizz, Buzz, 11, Fizz, 13, 14, Fizz Buzz, 16, 17, Fizz, 19, Buzz, Fizz, 22, 23, Fizz, Buzz, 26, Fizz, 28, 29, Fizz Buzz, 31, 32, Fizz, 34, ...

レポート２：処理の流れの制御

下記の問いに答えよ．提出形式はソースコードと出力ファイルやコメントを書いたreadmeテキストファイルをzipでまとめてアップロードせよ.

- [課題2-1] for文と，if と gotoだけでFizz Buzzを作れ． else, else if, switch等を一切使ってはいけない.
  - ただし，言語はC言語(またはC++)を利用すること.
- [課題2-2] 「Go To Statement Considered Harmful」（goto文は害悪だ）と言われているが，そうではないものも存在する.
  変数i,j,kで作られる3重ループのそれぞれの変数が０～１０まで変化するとき，i,j,k=(7,7,7)を満たした場合にループを抜けるプログラムを，gotoを使った場合，breakを使った場合それぞれ書け
  - これは，多重ループの脱出はgotoの場合のほうがスムーズにいく場合の例である.

C or C++ を利用する

- CSEにSSHで接続
  - 方法はどこかに書いてあるはず
- Windows 環境
  - Windows Subsystem for Linux(WSL)を利用 (ほぼLinux互換環境)
- Mac環境
  - Xcode でCommand Line Tools をインストール(gcc が利用可能)
- その他
  - Google Cloud Platform (GCP), Amazon Web Service (AWS)でCPUを借りて（Linuxをインストール）そこでgcc を利用

まとめ

- 構造化プログラミング：プログラムを基本構造（逐次実行，分岐，ループ）＋アルファでつなげて表現すること
- 構造化定理: **逐次実行，分岐，繰り返し**の基本構造で全てのフローチャートが表現可能
- パラダイムの変化：動けば良い

  →**可読性・保守性の向上**
- 制御文が散らかるのを整理する
  - 変数が散らかるのは？
    - 命名規則やオブジェクト指向で解決