計算機言語 II 第 15 回 補足

http://www.math.u-ryukyu.ac.jp/~suga/gengo/2018-2/15.pdf

1 授業では解説できなかった予約語 (キーワード)

これまでの授業で, C の文法の主要な部分の解説はしました. 授業では述べなかった, 予約語の意味を解説しておきます.

現在では使う意味がほぼ無くなった予約語

register 変数の型に対する修飾語. register(レジスタ) とは, CPU 内にあるデータを保存する場所のことで, このデータに対して演算が実行される. CPU 内にあることからメモリアクセスが不要で, その分処理が速くなるが, register 変数に対しては, (値がメモリ内に無いので) アドレス演算子が作用できない.

C の開発当初は, コンピュータの仕組みも単純で, register 変数を利用することによりより速く動作するプログラムを書くことが出来た. 現在では CPU の複雑化とコンパイラの性能向上により, 人間が高速化を考えるよりコンパイラに高速化処理を任せる (最適化処理をさせる) 方がより速いプログラムが書けるので, この修飾語を使うことは無い.

auto 変数に対する修飾語. 自動変数の意味で, 通常の変数宣言では何も修飾語をつけなければ自動変数になる. その意味で, 使われることはほとんどない.

使う頻度が少ない予約語

volatile volatile とは「揮発性の」とか「変化しやすい」という意味. プログラムの外部から値が変更される可能性のある変数に用いる変数型の修飾語.

講義の中で述べたようなプログラムでは, プログラムの外から実行中のプログラムのデータの値を変更することはありません. このような事は, セキュリティホールに繋がりますので, 通常は OS がそのようなことを拒絶します.

しかし, コンピュータの周辺機器を操作するプログラムでは, コンピュータとその周辺機器との通信で, 様々な制御をします. つまり, 周辺機器とのやり取りで, プログラム内の変数が周辺機器によって書き換えられます. そのような変数に対する修飾語です. この修飾語を用いることで, コンパイラは, この変数に対する最適化を避け, 周辺機器とのやり取りが間違いなく実行できるようにします.

C は OS の開発言語ですから, 周辺機器の操作のプログラムを書くことも多く, そのためにこのような修飾語があります.

goto プログラム内にラベルを定義しておき, そのラベルの場所に処理を飛ばす命令です.

Dijkstra(ダイクストラ) が 1970 年頃に「構造化プログラミング」という概念を提唱しました. それは,

この講義で述べたように, プログラムの処理を (C だと関数に) 分割してプログラムを書いていくというものです. それまでは, プログラムで起こる様々な条件分岐を GOTO 文を書いて行なっていました. その結果, GOTO が複雑に絡み合ったプログラム (スパゲッティプログラムと言われる) ものが多く作成され, プログラムを追うのが大変になりました (特に修正が). そこで,「GOTO を使わずにプログラムをすべし」という標語のもと, 構造化プログラミングというものが登場しました. C は構造化プログラミングのためのプログラミング言語です.

goto が全く不要かというとそうでもなく, 何重にもなった繰り返し処理の中の部分でエラーが起こって処理を中止するときなどでは, goto で一気にエラー処理にジャンプすることもできるので,「なんとかは使いよう」の類のキーワードです.

その他のキーワード

extern　変数に対する修飾語で,「外部変数」という意味. extern があるとその変数の宣言はされるが, 定義 (その変数を保存するための領域確保) はされない. 別のファイルで本来の定義が書かれている.

continue　case 文のところで break というキーワードを解説したが, break は, case 文や loop を抜けるためのキーワード. それに対し, continue は, その処理をスキップして loop の次のステップに進めるための命令. つまり, continue は loop から抜けずに, すぐに loop の次のステップに処理を移すときに用いる.

enum　列挙型の宣言. 例えば,

```
enum {Sunday, Monday, Tuesday, Wednesday, Thursday, Fridey, Saturday}
```

のようにすると, 識別子 (名前), Sunday, Monday, ... , Saturday に対して, 0, 1, ..., 6 が代入される. 詳しくは, 講義で用いた教科書 p. 220 を参照してください.

static

static キーワードは, 変数の修飾語として用いると,「静的変数」の宣言になりました.

これを関数の修飾語としても用いることが出来ます. 関数の修飾語として static を用いると, 他のソースからは参照できない関数になります.

行列や連立一次方程式を解くプログラムを書くときに, 奥村さんが書かれた「matutil.c」の関数を利用しました. このときに分割コンパイルをして, 別のソースから matutil.c の関数を呼び出しました. 適切なプロトタイプ宣言を書いておくと, 別のソースにある関数を利用してプログラムを作り上げることが出来ます.

関数に static キーワードが付けられていると, その関数に対しては, 上のことが出来なくなります. これは, 同じ名前を持った関数の衝突に対する対処法です. 例えば, 値を入れ替える swap(x, y) という関数が, あるファイルでは, double 型の変数の入れ替えになっており, あるファイルでは int 型の変数の入れ替えになっているということが大規模なプログラムでは起こり得ます. このような場合に static キーワードを付けて, これらの関数は, 他のファイルからはアクセスできなくすることで, 同一関数の名前の衝突を防ぎます.

講義で解説しましたが, プログラムでは「識別子 (名前)」を利用して値を参照します (関数呼び出しも同じです). ソースコード上で同じ識別子を持つものがあった時に, それをどう区別するか? という問題がコンパイル時に起こります. 変数に関しては, 変数の通用範囲 (scope, スコープ) を文法上限定することで対処しています. しかし, C では, 関数に関しては文法的には, 本質的な対処をしていません. そこで, この static キーワードで, ファイル単位で名前の衝突 (重複) を避けるようにしています.

2 さらに進んで

2.1 アルゴリズムとデータ構造

この講義では, 時間の都合上, アルゴリズムとデータ構造についての話をほとんどすることが出来ませんでした. データ構造については, 木構造 (2 分木, B-tree 等) や, スタック, キュー (queue, 待ち行列と訳す) など様々な構造が考えられ, それぞれに対して, それを操作するアルゴリズムが開発されています.

C を用いてこれらの実装を書いた本として, 講義中に挙げた奥村先生の本があります. それ以外にも,

> C で学ぶデータ構造とプログラム, Leendert Ammeraal 著, 小山裕徳訳, オーム社 (前回紹介した)
> 定本 C プログラマのためのアルゴリズムとデータ構造, 近藤嘉雪著 SOFTBANK BOOKS

あたりが良書だと思います. これら以外にも古典的な名著として (アルゴリズムの表記は Pascal (Algol 系) だが, 一通りの C の知識があれば読める),

> データ構造とアルゴリズム, A.V. エイホ, J.D. ウルマン, J.E. ホップクロフト著, 大野義夫訳
> アルゴリズム + データ構造 = プログラム N. Wirth 著, 片山卓也訳, 日本コンピュータ協会

があります. 残念ながら, 上にあげた本は全て絶版中で書店では手に入りませんので, 図書館や古本屋を探してください. 以前にも述べましたが,「やさしい」の類の言葉が表題にある本は,「重要だけど難しい内容を省く」という操作をしてありますので, 別の本で勉強するにしても, そのような題名の本は避けた方が無駄になりません.

プログラミングの世界でも,「万能のアルゴリズムは存在しない」ということが成立しています. 講義で, 整列法として quick sort のライブラリ関数の使い方を紹介しました. しかし, データの性質によっては, quick sort よりも速い整列法が使える事もあります.

また,「空間と時間のトレードオフ」と呼ばれる問題があります. すなわち, 速いアルゴリズムは, 一般的にメモリを多量に消費するという問題です. 色々なアルゴリズムを勉強して, 問題に対して最適な方法が選べるようになるのが理想ですので, 興味のある人は, 定評のある書籍を手にとって, 実際にプログラムを作ってみてください.

2.2 開発環境

この講義では, 簡単なライブラリの使い方と make コマンドの導入だけを講義しました. Linux (Unix 系) でも, これら以外にも様々な開発のための補助環境が存在します. それらを利用して,「効率よくコードを書く」というのも重要な事ですが, たくさんありすぎてそれらを解説することはできません. これらもネットや書籍を参照してください. man コマンドだけでも色々なことがわかります.

Linux(Unix) 環境以外では, その開発環境も変わりますが, それらは, 実際にそれを手に入れて動かさないと分かりません. (特に IDE, Integrated Developing Environment, 統合開発環境はそうです.)

C コンパイラーに対するコマンドオプションは, man コマンドを診て下さい.

quick sort の時に述べましたが, 上の様々なアルゴリズムは, すでにライブラリ関数として実装されていることも, 多くあります. C では, 標準的に備わっているライブラリ関数にも規格がありますので, これもネット, 書籍, man コマンド等で調べてみてください.

授業でやったような, コマンドラインから実行するようなプログラムを書く際には, C はそこそこ便利ですが, Window System を用いた GUI(Graphical User Interface) を利用するプログラムでは, C 単独で出来ることは素朴すぎて, GUI 周りのライブラリ関数を構築してからプログラムをするのが通常です. このような開発環境も, 様々なものがあり, 多くはネットから入手できます.

今では, ネットに沢山の C のソースコードがあります. 最善の勉強方法は,「他人が書いたソースを読んで理解する」です. 特に Open Source Software(ソースコードを公開してネットで協力して開発していくソフトウェア, Linux はその一例) には, 質の良いソースが多くありますので, それらのプログラムのソースを読むことは, 良い勉強になります.

2.3 別のプログラミング言語

C は現在でも多くのソフトウェア開発に利用されていますが, 言語仕様の曖昧なところ (char 型や int 型の範囲が実装依存で規格では定まっていないなど) があるので, もっと仕様のはっきりした言語を使う事も多くあります.

それらの中でも, C++, Java, C# などは, C の後継の言語とも言え, C とほぼ同じ記法でプログラムができます. これまでの授業を理解していれば, これらの言語に移行するのは, それほど難しいことではありません.

これら以外のプログラミング言語においても, 基本的に「プログラム = アルゴリズムの実装」であり, 制御構造やデータ型定義の記述が変わるだけなので, 今までの学習は無駄ではありません.

最近のプロブラミング言語は,「オブジェクト指向」言語となっていますが, 「オブジェクト = アルゴリズム + データ構造」が一つの考え方になっていることは, 知っておいて下さい. 講義でオブジェクト指向言語を取り上げないのは, データ構造とアルゴリズムの実装を並行して講義する必要があり, 時間が足りなくなるからです. まずは,「アルゴリズムを実装する」を勉強し, 複雑な処理が必要となった時点で「データ構造をうまく作る」という風に学習した方が, 初学者にはわかりやすいと, 私は考えています.

プログラミングも数学と同じで, それほど易しいものではありません. ものすごい才能がない限り, ひとつひとつ積み上げて学習する以外に, 習得する方法はありません.