【初心者向け】超速でプログラミングが上達する方法【10年先まで戦う】

Taiyo Project

【初心者向け】超速でプログラミングが上達する方法【10年先まで戦う】

2015年10月3日 [開発者コラム]

実際の業務で使えるレベルまで上達できる方法をお教えします

この話を始める前にまず大前提として、私は過去8年間中小のソフトウェア会社に勤務し、業務として様々なプログラムを書いてきました。

そして今では会社を辞め、Taiyo Projectという名義で個人でスマートフォンアプリ・ゲームの制作を行うという、ちょっとだけ実力のあるプログラマーです。(きっと多分。)

そんな私が、これから初心者を脱し中級者を目指そうとするプログラマーの方々のために、10年先まで戦えるプログラミング上達の方法をお教えしたいと思います。

プログラミング初心者に送る、超速でプログラミングが上達する方法とは？

それでは早速、超速でプログラミングが上達する方法を1つずつ解説していきましょう。

プログラミングが上達する方法その1 入門書を3冊ぐらい読破する、もちろん写経アリで

まず、本当にガチで初心者の方でしたら、入門書を読破するところから始めましょう。

この時、できることなら「同じ言語の入門書で別の著者のもの」を2～3冊用意してください。

理由としては次のようなものが挙げられます。

1. 初級者レベルの内容を反復練習することで、プログラミングという行為に自然と慣れることができる。
2. 著者が違うと、同じ内容を異なる言い回しで表現していたりするので、それらを読み解くことでより理解が深まる。
3. 初心者用のカリキュラムなら挫折しても頑張れば解決できるため、適度なストレスと成功体験を得ることができる。
4. てゆーか、そもそも入門書1冊読破した程度じゃプログラマーとして使いものになる訳がない。

まぁ、4番目の理由が一番大きい(笑)のですが、とにかく初心者だからこそ自分のレベルでできることは全てマスターしておくことが大事です。

もちろんこの時、入門書にあるサンプルコードは全てキーボードで自分で打ち込み、プログラムとして動作させることを徹底しましょう。(いわゆる写経ってやつですね。)

これはその言語の基本的な文法を覚える(と言うより感じる)ことができ、また打ち込みミスによるエラーが出た時に、本当に簡単にですが自分で書いたソースコードを読みエラー箇所を探すというデバッグという行為を体験することができます。

こういった経験があるか無いかが、後々の10年20年に繋がっていくのです。

プログラミングが上達する方法その2 コーディングのアンチパターンを実感する

次に重要なのが、ソースコードを書く上でやるべきではないパターンを知り、それを実感することです。

どういうことかと言いますと、例えば以下は多くの初心者プログラマーがやりがちなミスの一例です。

- メソッド(関数)が60行以上に渡って記述されている。
- コメントが無い、またはコメントがあっても意味が分からない。
- メソッド名や変数名に意味のある単語が使われていない。
- まるでコピペしたかのような同じ記述が、コード内に何度も出現する。

いくつか挙げましたが、要は初心者プログラマーはソースコードを「短く」そして「分かりやすく」書くことができないのです。

そしてそれ故に初心者は、ちょっと複雑なプログラムを書こうとするとすぐに自分が何を書いているのか理解できなくなり、そこで挫折してしまいます。

逆に言えば、ソースコードを短く分かりやすく書くことができれば脱初心者と言えるのですが、例えも無しにそれらを理解することは難しいですよね。

ですので、ここはちょっと脱線して私がおすすめするホームページをご覧いただくことにしましょう。

[>> Cプログラミング診断室]

> このページは私がプログラマー1年目の駆け出し時代の時に何度も反復して読み返していたページです。
>
> 主に取り上げているのはC言語でそれもそこそこ古い時代のものですが、書いてある内容は現代においても非常に役に立つものばかりです。
>
> 特にコーディングのアンチパターンを知るという点では、私はこのページより優れた書を見たことがありません。
>
> このページに書かれている内容の7～8割ぐらいが理解できるようになれば、おそらくもう誰もあなたのことを初心者プログラマーだとは思わないでしょうね。

プログラミングが上達する方法その3 ミニアプリを作ってみる

上述のアンチパターンを知るのと同時に、簡単なミニアプリを作ってプログラミングの練習をしましょう。

ここでおすすめなのは、次のようなプログラムを書くことです。

- FizzBuzz
  →基本的な条件分岐の使い方。
- バブルソート
  →int配列の並べ替え。
- 既存のコマンドを自作する
  →例えばcatコマンド(ファイルの内容を標準出力(コマンドプロンプトなど)に表示する)を自作してみる。
  →もちろん、コマンドオプションにもできる限り対応する。
- バイナリダンプ
  →ファイルの内容を1バイト毎に16進数で表示する。
  →やはりコマンドオプションを付与することで、1行8バイト/16バイト/32バイトを切り替えたり、ASCII文字の表示/非表示を切り替えたりなど。
- その他諸々・・・

まあ何と言うか、条件分岐のように基本的なところから、プログラマー業界ではいわゆる「車輪の再発明」と呼ばれるところまでを、ここではあえて行います。

そうすることで自分で1からロジックを考える能力や、知らないことを自力で調べる能力などが付くと私は思っています。

> プログラマーとしての業務の経験が無い方は、きっと「catコマンド」とか「バイナリダンプ」とかは知らないですよね。
>
> 知らないなら調べましょう。Google先生に尋ねるのです。
>
> 私もプログラミング歴は10年近くになりますが、それでも調べるという行為は毎日のように行なっています。

私もプログラミング歴は10年近くになりますが、それでも調べるという行為は毎日のように行なっています。

知りたいことを調べられるというのも、プログラマーに必要な能力の1つなんです。本当に。(その次に必要なのはペイントを使いこなす能力、笑)

プログラミングが上達する方法その4 再帰処理をマスターする

上で挙げたようなシンプルなプログラムが書けるようになったら、次は再帰処理と呼ばれるロジックを書けるようになりましょう。

再帰処理を語る上でよく引き合いに出されるのが「正の整数nの階乗」というやつです。

これは、正の整数nから1つずつ小さい整数を1まで順に掛け算するというもので、次のような式で表されます。

n! = n * (n – 1) * … * 3 * 2 * 1

例えば、nが5であった時、nの階乗は「5 * 4 * 3 * 2 * 1 = 120」となりますね。

これをプログラムで表すと次のようになります。

このプログラムのfactorial関数の処理の内容を、頭の中で整理しつつ追ってみてください。

```c
#include <stdio.h>

/**
 * 階乗を計算する関数
 */
int factorial(int n) {
    if (n == 0) {
        // nが0の時は1を返す
        return 1;
    } else {
        // nが0でない時、factorial関数内でさらにfactorialを呼び出す
        return n * factorial(n - 1);
    }
}

/**
 * メイン関数
 */
int main() {
    // 5の階乗を求める
    int answer = 0;
    answer = factorial(5);
    printf("5の階乗は%dです。\n", answer);
    return 0;
}
```

※上記はC言語ベース、動作は未確認です。ご注意ください。
※想定では標準出力に「5の階乗は120です。」と表示されます。

プログラミング初心者の方でしたら、この階乗をどうやって実装したでしょうか？

おそらく上記のようにはならず、for文またはwhile文を駆使して実装したのではないでしょうか？

もちろんそれでも実装できますし、階乗程度のロジックであればその方が簡単かもしれません。

しかし、さらに複雑なロジックを実装しようと思った時、再帰処理を理解しているということは大きなメリットに繋がります。

なぜなら、再帰というのはループ処理と条件分岐処理がとても複雑に合わさった芸術とも言えるロジックだからです。

そういった考え方を自分の頭の中で整理しさらに実装までできることができれば、おそらく向こう10年プログラミングのロジックで悩まされることは無いでしょう。

なお、再帰処理の実装の練習には次のようなプログラムを組むと良いでしょう。

- 二分探索ロジック

→代表的な配列の探索ロジックです。練習にはぴったり。

- 数独解答プログラム
  →こちらは応用編、初心者らしく総当りでやればOK。ナンプレの応募もこれで楽勝！(笑)

> ちなみに、再帰処理より頭の中がこんがらがるロジックは実際の業務でも滅多にお目にかかりません。
>
> そのためか、目の前の仕事を終わらせることだけしかしてこなかったプログラマーにこういった複雑な処理を渡すと、面白いぐらいにぴたっと手が止まります。
>
> ぶっちゃけた話、再帰処理を正しく理解・実装できるだけで、今現在IT業界で働くプログラマーと名乗る人間の約50%ぐらいを一気にごぼう抜きできるでしょうね。
>
> 情けない話ではありますが、事実です。

プログラミングが上達する方法その5 色々作って遊んでみる

再帰処理まで理解できるようになれば、世の中が大分違って見えるようになっているはずです。

ですので、そろそろここら辺で色々遊んでみましょう。

例えば、私が研修生時代に作ったプログラムには次のようなものがあります。

- コンピューター対戦型オセロ
  →メモリの使い方が下手くそだったこともあり、AIの実装がいまいちでした。
  →ダブルバッファリングという概念を知らなかったため、画面更新時の描画のチラつきも酷かったです。
- IPメッセンジャー
  →同一サブネット内のユーザーとテキストでチャットができるアプリ。ネットワークプログラミングの練習で作ってみました。
  →キープアライブのためにブロードキャストでゴミパケットを投げ散らかす、ちょっと迷惑なロジックを組んでしまった記憶があります。

他にも何か作った気がしますが、さすがに10年近く前のことなのであまり思い出せませんね。

実用性の有無は置いといて、こういった多少手の込んだアプリを1人で作れるようになれば、もうあなたは立派なプログラマーですよ。

しかし、あまり上達し過ぎるのも問題かも・・・？

ちなみに私は全くの未経験でこの業界に入ったのですが、最初の会社の新人研修では与えられた課題の他に上に書いたようなことを黙々とやっていました。

週5日、1日8時間、それを4ヶ月間毎日ですね。

たまに気分転換のためにC++の教本(約600ページ)を買ってみたりして、「1日150ページだー！」とか言って狂ったように写経しまくったり・・・、いや自分どんだけ暇だったんだってぐらい本当に基礎的な訓練を毎日続けました。

その結果、業界8年目にしてあまりの仕事の速さに会社に居場所が無くなり、結局こうやって1人自宅でプログラミングすることに・・・。

あれ、おかしいな？思っていたのと違う(ﾟдﾟ)

寂しいことに、上達し過ぎるというのも色々と問題が多い業界かもしれません。

話は逸れましたが、とにかく、この記事がこれからプログラミングを勉強しようと思っている方やプログラミング中級者を目指そうと思っている方のためになれば嬉しいです。

ぜひがんばってくださいね！ヽ(ﾟ∀ﾟ)ﾉ ﾊﾟｯ☆

関連記事

- UnityとCocos2d-xを比較 開発環境で割と本気で悩んだ話
- スマホアプリの広告の入れ方を変えたらクリック率が1.5倍になった話
- 創作でお金を稼ぐことは果たして悪なのか？
- スマホアプリのマネタイズの方法で開発者として個人的に貫きたい信念みたいなものを語ろうと思う

コメントを残す

名前 (必須)

メールアドレス (公開されません) (必須)

ウェブサイト

コメントを送信