プログラミング演習 第 1 回
はじめての C

2000 年 4 月 6 日

目 次

1 この講義では **1**

2 C 言語の欠点 **1**

3 参考文献について **2**

4 単位について **3**

5 授業を受ける際の注意 **4**

6 Hello, world **4**

6.1 ソースファイル hello.c の作成 4

6.2 実行ファイルを作成 5

6.3 Hello, world プログラムの説明 5

7 cc の動作と用語説明等 **6**

8 レポート問題 **7**

9 レポートの書き方 **7**

10 おまけ **8**

10.1 コンピュータはどう計算しているのか 8

10.2 プログラミング言語は, なぜ必要なのか 8

10.3 プログラミング言語の種類 8

情報処理演習の講義で Pascal に触れました.
ここでは, もう一つの言語 C を用いたプログラミングの入門 (の入門？) の講義をします.
プログラミングの歴史とかプログラミング言語の種類とかについての話は後記のおまけを参照して下さい.

1 この講義では

C 言語を用いたプログラミングの入門を, 講義します. 計算機を実際に動かしてプログラムを作成してもらいます. 内容は, 主に C 言語の文法の話です. 外国語と同じで, 文法だけではプログラムは書けないのですが, 他のことに触れる時間的余裕は, 残念ながらあまりありません. 各自補うようにしてください.
何故 C 言語をするかは次の理由からです.

1. 最近の流行である.
   書店のコンピュータプログラミングのコーナーにいきますと, C 言語の本が多数並んでいます. また, Netnews でも C 言語に関する newsgroup がありいろいろな議論がかわされています. この様に, 勉強するにあたって色々な情報が身近にあるというのは都合がいいことです.

2. UNIX との親和性が高い
   この講義では, UNIX を用いてプログラミングの実習をします. C 言語はその UNIX の開発言語です. 従って, UNIX には C 言語でのプログラミングを補助する色々なツールがそろっています. (補助する道具については, どの程度授業で述べられるかは, わかりませんが...)

2 C 言語の欠点

C 言語は, UNIX の開発に際して assembly 言語 (意味は後記のおまけ参照。以下同様) の代わりとなるべく作られました. 従って, 比較的低級な言語です. と言うことは, computer の Hardware の簡単な仕組を理解していることが暗黙裏に要求されたりすることがあります. また, 初期の UNIX 開発のターゲットマシンだった Dec PDP-11 の機械語命令の影響を受けていまして, 必ずしも汎用性のある言語仕様だとは言えない部分があります.
さらに, 1980 年代には C 言語に様々な方言が存在しました. これは, 初期の C 言語の言語仕様がきっちりしていなかったことが原因なのですが, これを何とかしようということで, ANSI で標準化作業がなされました. (これに対して例えば Pascal は, 開発者が言語仕様をきちんと決めています.) そして出来たのが ANSI-C です. しかし, この標準化作業がいい格好で終わったとは言えない部分が多くあります. ということで, C 言語は初心者がいきなりはじめるには不都合なところが残念ながら多くあります.
また, 初心者にとっては C 言語の言語仕様の小ささ (機能が少ない) も欠点となります. プログラミング言語としては, 言語仕様の小ささというのは大した問題ではないのですが, 初心者には取っ付きにくいものになります. 例えば, コンピュータのプログラムを書く上で, 入出力というのは基本的に必要なものなのですが, C 言語そのものには, 入出力の仕様がありません. C 言語では, 多くの処理をあらかじめ作られたライブラリというプログラムに任せます. ということは, このライブラリ[1]の使い方をある程度勉強する必要があります.

[1] 文字, 文字列, 入出力, 数値関数, 日付の変更, 動的な記憶領域確保など

3　参考文献について

はじめのほうは, C 言語に対する参考文献で後半は, それ以外のものを挙げました.

[1] は, C 言語のバイブルです. C でのプログラミングを真面目にしたいのでしたら, 必ず買うようにしてください. なお, 最近日本語訳を修正した版が出版されています. この事実からわかるように, この本の訳は必ずしも良いとは言えず, 明らかな誤訳が多数含まれています. そういう意味では, 原書を手に入れたほうがよいかもしれません.

[9],[10] は一人でＣ言語勉強するのに最適な本であると思います. これらを読んだあと, [1] を読むことを奨めます.

[2], [3] は, プログラムを書く上での心構えを主に書いた本です. プロフェッショナルを目指すなら, 一読すべきでしょう.

[5], [4] は, C 言語を用いた数値計算法の本です. [5] には, 英語版ですが Pascal を同じ内容の本があります. [4] には, プログラムのソースファイルが入ったフロッピーディスクがついてきます.

[6] は数学者が書いた本です. そういう意味では, 取っ付きやすいかもしれません. ただそこに書いてあるプログラムのプログラミングスタイルに少々問題があります. [7] は, [6] の結び目理論にかかわる部分を, 追加して際構成した本です。

UNIX のシステムコールに関連したプログラミングの本として [8], [12] を挙げておきます. この二つは共に古いので, 取り敢えず挙げただけです.

[13] は UNIX のネットワークプログラミングの本です.

[14], [15], [16] はプログラミング全般にわたる古典的名著です.

[14] は, わかりやすいプログラミングの心得を書いたものです. 処理系には ratfor と呼ばれる独自の言語を用いて書いてあります. ratfor は皆さんが今使っている機械にもあったと思います. なお, 日本語訳はありませんが, これとまったく同じ内容を Pascal で書いた本が, 同じ著者で出版されています.

[15] は, データ構造とプログラムの関係を書いた本です. 処理系は Pascal を使っています. プログラムというのは, しょせんデータの処理方法の記述ですが, そのデータ構造をうまく解析して与えないと, よいプログラムが書けないと言う視点でプログラミングを解説しています.

[16] は, プログラムに現われる数学的構造を面白く書いた本です. 第 2 巻までは, 日本語訳があります. 著者は, この本の出来上がりが気に入らないといって TEX を開発したのは有名です.

皆さんの使っている機械では, C 言語以外でもコンパイラ系で, Fortran, Pascal, インタプリタ系で, Awk, Perl が動きます. Fortran 以外について [17], [18], [19] に各々の処理系の原典を挙げておきます.

世の中には, プログラミングの本も C 言語の本も山のように出版されています. この中で, 良い本を捜すのは大変です. 伝え聞く限りでは, 多くの本が出版されている割には良書と呼ばれるものは少ないようです. また, 翻訳本にはひどい誤訳もあるそうです. この様な情報は, たまに Net News などでも議論になっているようですから, Net News の C 関係の news group を定期的に読むことをお勧めします.

参考文献

[1] カーニハン・リッチー著, The C Programming Language,
日本語訳 石田晴久訳, プログラミング言語 C 第 2 版, 共立出版.

[2] S. Oualline 著 岩谷宏訳, Practical C Programming 現実的な C プログラミング, ソフトバンク.

[3] 藤原博文著, C プログラミング診断室, 技術評論社.

[4] L. Baker 著 吉田弘一郎訳, C 言語数学関数ハンドブック, 技術評論社.

[5] W. H. Press 他著 丹慶勝市他訳, Numerical recipies in C, 技術評論社.

[6] 落合豊行著, C 言語による数学解析, 近代科学社 1988 年.

[7] 落合豊行, 山田修司, 豊田英美子著, コンピュータによる結び目理論入門, 牧野書店 1996 年.

[8] Curry 著 アスキー編集部訳, UNIX C プログラミング, アスキー 1991 年.

[9] 内田智史著, C言語によるプログラミング [基礎編], [応用編], オーム社 1991 年.

[10] 結城 浩著, C言語プログラミングレッスン [入門編], [文法編], ソフトバンク 1994/1995 年.

[11] エイチアイ著, 実用 UNIX C/C++言語ハンドブック, ナツメ社 1999 年.

[12] Rochkind 著 福崎訳, UNIX システムコールプログラミング, アスキー 1987 年.

[13] W. R. Stevens 著 篠田訳, Unix ネットワークプログラミング, トッパン 1992 年.

[14] カーニハン・プローガー著 木村 泉訳, ソフトウェア作法, 共立出版 1981 年.

[15] N. Wirth 著 片山卓也訳, アルゴリズム + データ構造 = プログラム日本コンピュータ協会 1979 年.

[16] Donald E. Knuth, The art of computer programming, vol 1 - vol 3 Addison-Wesley (1 巻と 2 巻は日本語訳有).

[17] K. イェンゼン・N. ヴィルト著 原田賢一訳, Pascal 第 4 版 培風館.

[18] A. W. エイホ, B. W. カーニハン, P. J. ワインバーガー著足立高徳訳,
プログラミング言語, AWK アジソン ウェスレイ, トッパン 1988 年.

[19] L. Wall, R. L. Schwartz 著 近藤嘉雪訳, Perl プログラミング, ソフトバンク 1993 年.

4 単位について

単位は 出席と e-mail によるレポートで判定します. レポート問題は, 配るテキストに書いてあります (後ろ参照). 送り先は,

osaka-report@se.ritsumei.ac.jp

去年も何度も注意を致しましたが, 基本的に間違った場所に提出したレポートは評価いたしません.

5　授業を受ける際の注意

授業中，以下のことは禁止します．

Netscape, Internet Explore 等の WWW ブラウザの使用

各種 game 等の娯楽一般

暇な人は Web で遊んだりせずに，周囲の人々にアドバイスを送ったりして，受講者全員のレベルアップに協力して下さい．よろしくお願いします．

ちょっとしたタイプミスで次に進めなくなることがあります．TA(ティーチングアシスタント) の学生に聞くだけでなく，うまく動かないときは，隣の人に見て貰うようにしましょう．ちょっとみて貰うだけで，解決する場合もあります．

6　Hello, world

6.1　ソースファイル hello.c の作成

まずは C 言語を使った最も簡単なプログラミングを作ってみましょう．次のプログラムは，C 言語の入門では必ず一番最初に書くものです．

画面に Hello, world と表示するプログラム．ファイル名 hello.c

Mule を用いて，hello.c という名前のファイルに次の内容を入力し，セーブしてください．/* と*/ で囲まれた部分はタイプしなくて良いです．

```
-----この下から-----
/* Hello, World を出力するプログラム */

#include <stdio.h>

main()
{
        printf(" Hello, world\n");
}
-----この上まで-----
```

/* と*/ で囲まれた部分はコメント文と呼び，プログラムの説明などを書きます．この部分はなくてもプログラムの実行には関係しません．

授業中はタイプの時間の節約のため，コメント文を書く必要はありませんが，レポート課題ではきちんとコメントをタイプするようにして下さい．

注意 コメント文が必要な理由の一つは, 基本的にプログラムは他人も見ることを前提に書かれるべきだからです. それ故, 誰が見てもわかるようにコメントを入れておくとそのプログラムの改良などの作業がしやすいわけで, 例の 2000 年問題はその点修正が苦労したそうです. 構造化プログラミングの思想にのっとって, 誰にもわかりやすい,「数学屋のような癖」のないプログラムを書くようにしましょう.

6.2 実行ファイルを作成

ソースファイル hello.c から実行ファイル a.out を作成します．

(1) `cc  hello.c` とタイプします．
3 - 5 秒後に再びプロンプトが現われます．a.out という新しいファイルが出来ていることを ls コマンドで確認しましょう．

(2) このファイルは，実行可能ファイルです．`ls -l a.out` で，ファイルモードが実行可能 (x) になっていることを確認して下さい．(普通はしなくて良い)

(3) `./a.out` と入力します．

上の操作をソースファイルをコンパイルして実行するといいます．

コマンド cc がうまく実行できなかった人は，次をチェックしてください．

1. ファイル名は，hello.c となっているか．
2. コマンド又はファイル名を正しくタイプしたか ?
3. hello.c の中味の括弧や quotation は，正しく入力されているか ?
4. #include の # は行の先頭にあるか ?
5. printf 文の最後にセミコロン ; がついているか ?

メモ

UNIX のコマンドを忘れた人がいるかも知れませんが, UNIX 内で作成したファイルにはいくつか情報がついてまわります. 例えば, 今作成した実行ファイルを`ls - l a.out` でみると,

-rwxr-x— 1 osaka teacher 5844 Mar 31 11:52 a.out

ここで重要なのは最初の１０項目です. 一番目はファイルの種類を表しています. "-"は通常ファイル、"d"はディレクトリを意味します. 残り, ９項目は, ３項目づつ所有者, グループ, その他のユーザに対する許可情報を与えています. 今の場合, 所有者に, r(読みとり), w(書き込み), x(実行) 許可が与えられていることがわかります. 先に述べたことは, ここでもし, 所有者情報で`x` が`-` であったら, `chmod u+x a.out` として許可を与えなければなりません. ここの, `u+x` は、`u`(所有者) に`+x`(実行許可を与える) ということです. これらのことは UNIX の基本でありますので, 時間があるときにでも本などに目を通して下さい. 例えば,「たのしい　UNIX」(坂本文著、アスキー出版社).

6.3 Hello, world プログラムの説明

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

#で始まるこの部分は，C プリプロセッサ (preprocessor) cpp への命令です．(とりあえずは，おまじないと思っていて下さい．)

```
main()
```

main と言う関数の始まりを示します．C 言語では，必ず main と言う名前の関数が必要で，ここからプログラムの処理がはじまるようになっています．() の中には，この関数のパラメータ (通常プログラミングとか計算機の分野では，引数という．) が入ります．

```
{
        printf("Hello, world\n");
}
```

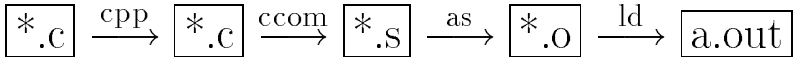
ここが，このプログラムの本体です．C 言語では，一連の処理は中括弧 { と　} で挟まれた部分に書きます．中央の

```
        printf("Hello, world\n");
```

が，このプログラムがする全てです．printf は，標準入出力ライブラリにある関数の名前です． 2 重引用符 (") で囲まれた文字列を標準出力 (ディスプレイ) に表示します．最後の\n は改行を表わします．

7　cc の動作と用語説明等

コマンド cc は一つのプログラムではなく，少なくとも次のプログラム (プロセス) を作り出します．下の図の四角く囲まれた部分に書かれているのは，その時に使うファイル名です．



cpp C preprosessor: C compiler (ccom) が処理をする前に，文字列の処理の部分のほとんどを受け持って，事前に処理をするものだと思ってください．

ccom C compiler : C 言語の文法で記述された処理内容を，assembly 言語に変換するプログラム．

as Assembler : assembly 言語で書かれた内容を，機械語 (すなわち 2 進法で記述されもの) に変換するプログラム．

ld Link editor : リンカともいう．上の hello.c ではコンパイルした結果は，Hello, world\n という文字の並びを関数 printf に受け渡すという命令だけが，作り出されます． printf の実体は，ライブラリとしてコンピュータのファイル

```
/usr/lib/libc.a
```

に入っています．ld は，hello.c のコンパイルしてできた結果と上のファイルにある printf を結び付けて (リンクするという)，さらにシステムが，実行時に要求する情報をつけて実行可能ファイルを作り上げます．

Library プログラムを組むと分かりますが，画面の入出力などはどのプログラムでも同じ様なことをします．これをいちいちゼロからプログラムしたのでは，無駄が多いのです．そこで，よく使う処理はあらかじめ 2 進法レベルのものを作成，まとめたのが必ずあります．これがライブラリ (library) です．ライブラリは，compiler(実は，リンカ) に指示することによって，自分のプログラムにくっつけることが出来ます．また，C 言語では標準ライブラリというのが ANSI により定義されており，多くの処理はこれを使って実現します．

ファイル名 C 言語のプログラミングにおいてファイル名は，次の規則にしたがって，名前をつけます．

1. *.c :C 言語のプログラムを記述したファイル．
2. *.h :ヘッダ (header) ファイル．定数とか関数の定義を記述する．
3. *.s :assembly 言語のプログラムを記述したファイル．

4. *.o :オブジェクト (object) ファイル．2 進法で記述した機械語が入っている．
5. a.out :特別なオプションを指定せずにコンパイルしたときに出来る実行可能ファイル．

上のファイルのうちいくつかは，hello.c をコンパイルするときには現われませんでしたが，それは単に目に見えないだけで，コンピュータの内部ではちゃんと作られています．

8 レポート問題

問題 1 次のように表示するプログラムを作成せよ.

----------ここからはじまる--------

プログラミングはおもしろい.

```
Programing is interesting.
```

----------ここでおわり----------

二つの文の間に空行を入れていることに注意.

要望のある方は, 上のプログラムの下に授業でやって欲しい内容など書いていただければ幸いです. レジュメを作成する際みなさんの意見を参考にしていきたいと思います. (但し, 私の能力の範囲でおこないます)

9 レポートの書き方

レポートは特に定めない限り, 次のように書く.

1. 名前の書き方: 1 行目にコメントとし, 次のように書く.

   /* 学籍番号　名前*/

2. 説明をつける: 使用している変数や処理に, 今回 (あるいは今後) のレジュメのプログラムを参考にしてコメント文を入れて下さい.
3. 締切: 授業のあった翌週の水曜日まで.
4. 提出先: osaka-report@se.ritsumei.ac.jp
5. **Subject**: メールのタイトルは次のようにして下さい. 第 n 回の問題 m 番目の場合

   creportno.n-m

   今回は`creportno.1-1` です.

10　おまけ

10.1　コンピュータはどう計算しているのか

今のコンピュータは, 電気で仕事をしています. 数字は電気を用いて表されています. 例えば, ある基準電位から見て電圧がある時に 1, ないときに 0 というふうに決めてあります. このようにして, 電気を用いて 0 と 1 が表現できます. 従って, 2 進数を用いれば電気を用いて数が表現できることがわかります.

今のコンピュータの内部では, 全てのものが 2 進数で表現されています. また, 上のような電気の 0 と 1 の世界にうまくスウィッチを組み合わせると 2 進数での加法とか, 乗法をする電気回路が作れます. この回路をうまく利用して, コンピュータは計算をしたり文書処理をしたりしているわけです. また, この電気回路の利用の仕方の指定を変えれば, 一つの計算機が, いろいろな用途に使えます. この利用の仕方を変えることが, プログラミングなのです. そして, 利用の仕方を変えるのを記述する言語がプログラミング言語で, 利用の仕方の記述の事をプログラムといいます.

電子式として最初に活躍したコンピュータ ENIAC では[2], 配線を付け変えるという作業を人間がして, 利用の仕方の変更をしました. しかしこの方法では, ちょっとした計算にも膨大な人手がかかる事は明らかです.

現在のコンピュータのハードウェア (電気回路) には, この利用の仕方を与える幾つかの命令を, 用意するようにしました (von Neumann のアイデア). この命令を使って, コンピュータの利用の仕方を制御しております. そして, この命令も 0 と 1 の並びで記述されます. この 0 と 1 の並びで記述される命令もプログラミング言語で, 機械語といいます.

10.2　プログラミング言語は, なぜ必要なのか

さて, コンピュータにとっては便利な 2 進法も人間にとっては不便なことこの上ありません. 0 と 1 の並びでそこに何が記述してあるかを読み取るのは, 簡単な場合以外はまず不可能です.

そこで, 人間が用いる言葉に近い記述でプログラミングができるように工夫がなされました. そうして生まれたのが, プログラミング言語です. プログラミング言語というのは, コンピュータに対する命令を記述するための, ある人工的な文法をもった言語です. より広くはその言語の処理系 (その言語を用いて, コンピュータを働かせる仕組全体) のことも指します. コンピュータが, 文法に従って書かれた記述を自分の本来の言葉である, 機械語に翻訳し (実行し) ます. つまり, 人間にとって面倒なことをコンピュータにやらせるようにしたのです.

プログラミング言語を用いたプログラミングとは, 処理内容をその言語の文法にしたがって記述することです.

さらに処理内容の記述する時にも, コンピュータを利用するようになりました. プログラミングというのは, 処理内容を記述した文書を作ることですが, この文書を作るという処理にもコンピュータを使うわけです. この事が一般的になったのは, 意外にも比較的最近のことです.

10.3　プログラミング言語の種類

von Neumann 型のコンピュータが生みだされて以来約 40 年たちますが, その間に様々なプログラミング言語が開発されました.

まず, プログラミング言語はプログラムを処理する形態で, 次の二つに大きく分類されます.

Interpreter(通訳系と訳す事がある)

処理を記述したものから, 一部分のまとまりのある処理内容を機械語に変換しては実行し, それが終

[2] 実はイギリスで開発されたコロッサスが一番最初らしい. 一松　信著「暗号の数理」(Blue Backs) 参照

わると次の部分に移る, ということを繰り返して, 処理をします. つまり, 同時通訳の人が外国語から日本語に通訳するように, プログラミング言語の内容をその場で機械語に通訳して, 計算機に実行させているのです. awk は interpreter です.

Compiler(翻訳系と訳す事がある)
: 処理の記述を一度に全部機械語に翻訳してしまい, その後にそれを実行します. コンピュータは, 機械語に翻訳された内容を読みこみ, それを実行します. この授業では, compiler を取り上げます.

また, プログラミング言語は, その文法が違えば異なる言語です. 文法のなかで, 良く知られているものを挙げると次のようなものがあります. 注意して欲しいのは, 同じ文法を持つ言語でも compiler だったり interpreter だったりすることがあることです.

assembly 言語 プログラミング言語を作るに当って, 最も単純な発想はコンピュータに対する命令である 2 進法の並びを, 一対一で人間が理解できる言葉に対応させることです. この対応で一つの言語が作れます. それが, assembly 言語です. assembly 言語の処理系のことを assembler といいます. この言語を作るのは簡単ですが, 機械語で出来ることがあまりにも単純な事しかできないため, この言語を使うプログラミングは人間にとってはやさしいことではありません. 例えば, 画面に 1 を表示させるのも, assembly 言語を使うと結構大変な仕事になります. また, 機械語と一対一に対応しているということから, コンピュータが違えば, assembly 言語も違うということに注意してください. すなわち, assembly 言語という名前の全てのコンピュータに共通の言語があるわけではありません.

assembly 言語のようにコンピュータに密着した言語を, 低級言語といいます. それに対し, より人間に近い言語を高級言語といいます. 高級言語と呼ばれるものの特徴は, コンピュータが違っても (原理上は) まったく同じ記述でプログラミングが出来るということです.

Fortran Formula transformation の略です. 最も早い時期に作られた, 高級言語です. 主に科学技術用数値計算をするためのプログラミング言語です. 古くからあるために, 数値処理のためのパッケージとかが豊富で, 現在でも用いられています. ただ, プログラミング理論とかが無い時代の産物なので, 最近の理論から見ると良いプログラミング言語とはいえないようです. 実際問題として, 特に Fortran 用のパッケージを使う必要性がないのなら, もう勉強する必要もないと思います

COBOL Fortran と同時期に作られた高級言語ですが, こちらはもっぱら事務処理用言語として開発されました. 時々マスコミを賑わす 2000 年問題の多くは, COBOL を用いて作られたプログラムで発生します.

ALGOL Fortran, COBOL と同時期に開発された言語です. こちらの方は, 数学者が中心となって作ったといわれるだけあって理論的にすっきりした言語だそうです. ただ, あまり普及しませんでした. その理由はよくわかりませんが, 想像するに, あまりにも言語仕様が巨大になったことと, 標準での入出力の定義が, 与えられなかった事だろうと思います.

Pascal スイスの N. Wirth という人が, 教育用目的に ALGOL を元に言語仕様を小さくし, さらに標準の入出力を定義して開発しました. 1980 年代には結構流行し, 現在でも比較的使われています. 現時点でも, プログラミング入門用言語としては, 良いものだと思います. ただ, 元の言語仕様では大きなプログラムを書くにはちょっと不便です. なお, パスカルから派生した言語として modula-2 と Ada があります. またその孫ともいえる, Oberon という処理系もあります. Oberon は, IBM-PC 互換機とか Macintosh では無料で処理系が ETH(何の略かは忘れました) から入手できます.

LISP List Processor の略です. プログラム言語としては, かなりの歴史を持っています. やたら括弧をつかってプログラムをするのが, 特徴です. これと親戚の言語として, Scheme というのもあります. 人工知能や Expert System の分野では, 良く用いられます. 例えば数学では, 数式処理系 (数式のまま微積分をしたり, 行列計算をしたりする処理系) のひとつ Reduce は, LISP をつかって開発されています. 前期の授業で取り上げた emacs の拡張機能も, LISP の親戚である Emacs LISP で作られます. 以前は方言の多い言語でしたが, 現在では, Common LISP と言う規格が作られました. ただ, Common LISP の言語仕様もかなり大きなものになっています. Unix 用では, 京都大学で開発された KCL (Kyoto Common LISP) が, 有名です.

Prolog 日本のどこかの省庁の次世代コンピュータの開発言語として採用されました. 三段論法をそのままプログラムしていくような言語です.

C 1970 年代に Unix のシステム記述用言語として開発されました. OS の開発にはそれまでは, assembly 言語を用いていましたから, そこに高級言語を用いたということで, これは画期的なことでした. しかし, そもそもは assembly 言語の代わりとして作られたという経緯があるため, 比較的低級な言語になっています.

SmallTalk 1970 年代に, 新しいプログラミング言語として Xerox の研究所で開発されました. オブジェクト指向という新しいプログラミング規範 (paradigm) を導入した言語です. もともとは, 言語だけでなく OS も一体となった環境でしたが, 残念ながらその環境はあまり世に受け入れられませんでした. (値段の問題だと思います) OS の環境のほうは, 良く似た考え方が Macintosh に採用され最近の主流となっています. オブジェクト指向プログラミングの考え方も, C, Pascal, Oberon LISP などに導入されて, Objective-C, C++, CLOS 等の言語となっています. 当時の Xerox では, 現在の主流となっている技術, 例えばワークステーションの概念とか, EtherNet によるネットワークがすでに考えられており, Xerox なしでは現在のコンピュータはありえないのですが, 当の Xerox がそこから余り利益を得ていないのは, 歴史の皮肉です.

BASIC アメリカで, プログラミング入門用として開発されました. (開発者たちの名前は忘れました) 多くのパソコンに入っていたため, 一躍有名になりました. 本来の BASIC の言語仕様は, 割ときちんとしているのですが, パソコンに移植されたものが言語仕様の一部分だったため, 「良い」言語としては, あまり普及していません. (本来の言語仕様を True BASIC というそうです) 手軽ではありますが, パソコンに付いてくるのを使ってプログラミングの入門をするのは, 余りいいことだとは思いません. なお, 新指導要領では高校の数学で, この BASIC を使用するそうです.

Perl 最近少し流行している処理系です. これは, compiler ではなく interpreter です. だいたい $awk + shell + C$ の感じです. つまり, shell のように command を実行させることが出来, awk のように手軽な記述が可能で, C のように何でもできる言語です. (C の中で, command 実行はしない事はないが, 処理系に依存するので大変. shell は command 実行以外はほとんど出来ない. awk も文書処理以外の処理は, 余り得意でない.) UNIX のシステム管理者を目指すなら, 勉強して損は無い言語です.

Java WWW で閲覧するページにいろんな機能を付け加えるために開発された言語です. C 言語に似た構文とオブジェクト指向な環境が備わっています. 最近の WWW の流行にあわせて, これも流行しているようです.