月刊マイコン別冊

X68000 活用研究III

X-BASIC 活用Q&A

塚越一雄 著

X-BASICを
完全理解するための
55の疑問を解説

SHARP

20Mバイトハードディスク搭載、
HDモデル登場。

X68000

PERSONAL WORKSTATION

ACE HD

■本体＋キーボードCZ-611C-GY(グレー)・-BK(ブラック)標準価格399,800円
■15型カラーディスプレイテレビ(ドットピッチ0.39mm)CZ-601D-GY(グレー)・-BK(ブラック)標準価格119,800円
■15型カラーディスプレイテレビ(ドットピッチ0.31mm)CZ-611D-GY(グレー)標準価格145,000円
■チルトスタンドCZ-6ST1-E(グレー)・-B(ブラック)標準価格5,800円

シャープ株式会社 ●お問い合わせは…シャープ㈱電子機器事業本部システム機器営業部〒545 大阪市阿倍野区長池町22番22号☎(06)621-1221(大代表)
電子機器事業本部テレビ事業部第4商品企画部〒162 東京都新宿区市谷八幡町8番地☎(03)260-1161(大代表)へ

ますます熱くなる。
クリエイティブワークステーションX68000。

●新たなゆとりが創造力を刺激する——。20Mバイトハードディスクを本体に内蔵したX68000 ACE HD の登場です。もちろん、X68000としての本質は変わるはずもなく、あのクリエイティブなX68000そのものです。といって、たとえ3.5インチのハードディスクとはいえ、それをスリムなマンハッタンシェイプの本体内に搭載するには、これまで以上の実装密度が要求されます。このハードディスクモデルには、集積度をさらに高めたカスタムICや、メモリとして1Mビットのダイナミック RAMが採用されていますが、これは、いわば過去1年間の成果というべきもので、ある意味では、ビジュアルシェルなどのソフトウェアに対してハードウェアのユーザーインターフェイスとも言えるでしょう。

●約110本、X68000のパフォーマンスにふさわしいさまざまなジャンルのソフトウェアがすでに流通。このマシンのソフト環境は着実な歩みを見せています。この間、ユーザー各位の熱烈なご支持とシステムハウス各位の開発ご努力に心からの感謝をささげるとともに、そうしたご厚意に対して、私たちは将来的な展望も含めて、でき得るかぎりのサポートをお約束するものです。

〈X68000ACE HD の主な特長〉●3.5インチ20Mバイトタイプのハードディスク(平均アクセスタイム80ms)を1基内蔵 ●実装密度を追求して信頼性を高めたマンハッタンシェイプ ●68000搭載 ●テキスト、グラフィック、スプライト、独立3画面設計、最大12Mバイトの大容量メモリ(標準1Mバイト) ●フレンドリーOS、Human 68k搭載 ●連文節変換、マルチフォントをサポートした強力日本語処理 ●1024×1024ドット(最大表示エリア768×512ドット)の実画面エリアを装備した高解像度表示能力 ●512×512ドット、65,536色同時発色 ●水平32、1画面128のパワフルなスプライト機能 ●オーバースキャン機能を採用した512×512ドットレベルのスーパーインポーズ ●テキストビットマップ方式採用 ●8重和音ステレオFM音源搭載 ●音声デジタイズ記録AD PCM※採用 ●マウス・トラックボール標準装備 ●5インチ1MバイトFDD2基搭載 ●「X-BASIC」、「辞書ディスク」と各種ユーティリティ、「日本語ワードプロセッサ」をバンドル

※Adaptive Differential PCM

豊富な周辺機器がクリエイティブワークをサポート。

●15型カラーディスプレイ	CU-15M1-E	標準価格 99,800円
●カラーイメージスキャナ※1	CZ-8NS1	標準価格188,000円
●カラーイメージユニット※2	CZ-6VT1	標準価格 69,800円
●カラービデオプリンタ	CZ-6PV1	標準価格198,000円
●24ピン漢字プリンタ(80桁)	CZ-8PK7	標準価格122,000円
●24ピン漢字プリンタ(136桁)	CZ-8PK8	標準価格152,000円
●24ピン漢字プリンタ(80桁)	CZ-8PK9	標準価格 89,800円
●熱転写カラー漢字プリンタ	CZ-8PC2	標準価格 69,800円
●ハードディスクユニット(20MB)	CZ-620H	標準価格178,000円
●モデムユニット※3	CZ-8TM2	標準価格 49,800円
●RS-232Cケーブル(平行接続型)	CZ-8LM1	標準価格 7,200円
●RS-232Cケーブル(クロス接続型)	CZ-8LM2	標準価格 7,200円
●拡張I/Oボックス(4スロット)	CZ-6EB1	標準価格 88,000円
●1MB増設RAMボード(内蔵用)	CZ-6BE1A	標準価格 38,000円
●2MB増設RAMボード※4	CZ-6BE2	標準価格 79,800円
●4MB増設RAMボード※4	CZ-6BE4	標準価格138,000円
●GP-IBボード	CZ-6BG1	標準価格 59,800円
●ユニバーサルI/Oボード	CZ-6BU1	標準価格 39,800円
●増設用RS-232Cボード(2チャンネル)	CZ-6BF1	標準価格 49,800円
●数値演算プロセッサボード	CZ-6BP1	標準価格 79,800円
●スキャナ用パラレルボード	CZ-6BN1	標準価格 29,800円
●システムラック	CZ-6SD1	標準価格 44,800円
●アンプ内蔵スピーカーシステム(2本1組)	AN-160SP	標準価格 59,800円
●ジョイカード	CZ-8NJ1	標準価格 1,700円

※1 使用に際しては、カラーイメージスキャナCZ-8NS1に同梱のRS-232Cケーブルで接続するか、より高速のパラレルデータ転送を行う場合、別売のスキャナ用パラレルボードCZ-6BN1で接続してください。※2 使用に際してはコンピュータ本体と専用15型カラーディスプレイテレビ(CZ-601D、CZ-611Dなど)が必要です。※3 モデムユニットCZ-8TM2に同梱のソフトはX1/X1 turboシリーズ用です。※4 使用に際しては、あらかじめ、別売の1MB増設RAMボードCZ-6BE1Aを増設してください。

アートツールと呼びたい「PRO-68K」シリーズソフト。

MUSIC PRO-68K CZ-213MS 標準価格 18,800円

メロディ譜、ピアノ譜、最大8パートのスコア(総譜)を自由なレイアウトで書き込んだ譜面を内蔵のFM音源で演奏できる楽譜ワープロ&演奏用ミュージックツール。

SOUND PRO-68K CZ-214MS 標準価格 15,800円

FM音源のパラメータを直接指定したり、エンベロープやビブラートを言葉による音のイメージ指定で思いどおりの音色が作成できるサウンドエディティングツール。

BUSINESS PRO-68K CZ-212BS 標準価格 68,000円

スプレッドシート、データベース、グラフ作成機能を緊密に一体化させた統合ビジネスツール。マウス対応のイージーオペレーション、最大16個のマルチウインドウが使えます。

C compiler PRO-68K CZ-211LS 標準価格 39,800円

Cコンパイラ、BASIC-Cコンパータ、アセンブラ、リンカ、デバッガ、アーカイバ、コンバータで構成。Human 68k上におけるプログラム開発を効率良くサポートします。

〈ゲームソフト〉●ツインビー CZ-217AS 標準価格7,800円
●アルカノイド CZ-222AS 標準価格7,800円

さらに洗練されて信頼性を高めた ハイコストパフォーマンスFDモデルX68000ACE

X68000 PERSONAL WORKSTATION ACE

■本体+キーボード
CZ-601C-GY(グレー)・-BK(ブラック)標準価格319,800円

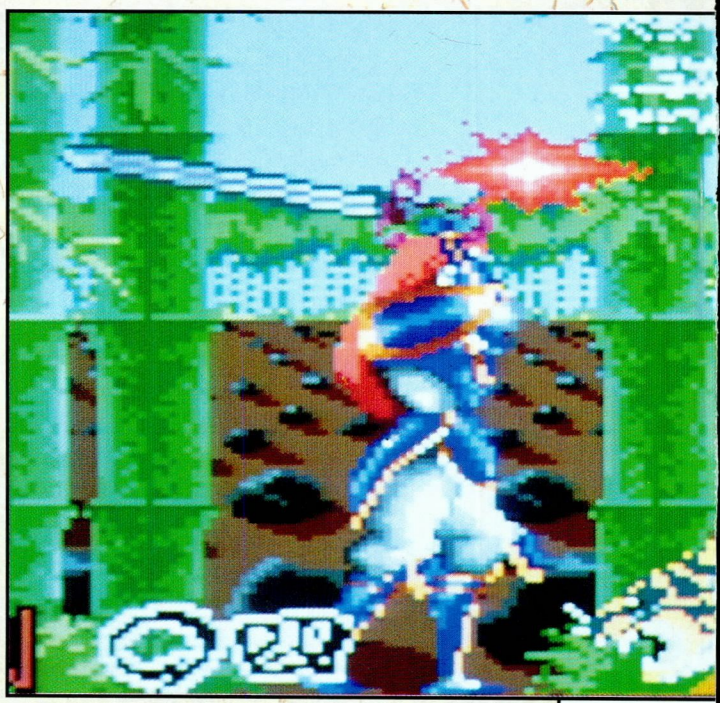
写真の15型カラーディスプレイテレビCZ-601D-GY(グレー)・-BK(ブラック)、チルトスタンドCZ-6ST1-E(グレー)・-B(ブラック)は別売です。

〈パソコン教室開催のお知らせ〉X68000、MZ-2861のパソコン教室を開催します。くわしくは、下記までお問い合せください。
札幌(011)642-8111・仙台(022)288-8705・東京(03)260-1161・横浜(045)201-6525・名古屋(052)332-2611・大阪(06)222-7655・神戸(078)291-8715・福岡(092)481-2860

資料請求券 X68000 活用研究III

超大型キャラクターが空を
魑魅魍魎が跋扈する鎌倉
ナムコの意欲作品をX6800

『源平討魔伝』ゲーム解説



未来永劫に続くかと思われた，源氏と平家の血みどろの抗争のため，国は乱れ，怨念は地に満ち，昼夜を分かたず死霊，生霊の跋扈する時代のお話です。

源の頼朝ひきいる源氏に滅ぼされた，平家の武者，平の景清が，天帝の命を受けた安駄婆により，地獄から甦るところから，ストーリーは，始まります。

平家の怨念を一身に封じ，景清は，鎌倉に居を構える頼朝を滅ぼさんと，国々を攻めのぼります。

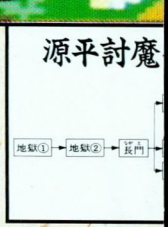
待受けるのは，義経，弁慶，琵琶法師など，おなじみのキャラクターに加え，餓鬼，鬼姫など多数の魔物たちです。魔力を秘めたこの者たちは，たとえ倒しても『これで勝ったと思うなよ……』等の不気味なつぶやきをもらしながら消え去りはするものの再び登場してきます。

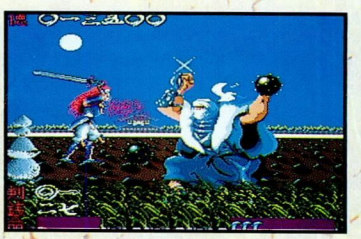
ゲームは，原則的に右方向へ進みます。出雲，京都などの国をクリアすると，次の国に進めます。国によって，画面は通常の「横モード」と，上から眺めた感じになる「平面モード」，それに超特大キャラクターが，画面を所せましと登場する「ビッグモード」の3種類に設定されています。地獄を含めると，47もの国(ステージ)があります。

源平討魔伝はステージマップ図のように，鎌倉時代の国々がステージになっています。ステージをクリアするには，魔物や義経，弁慶などの敵を剣で倒しつつ，一定の距離を進むとあらわれる鳥居に入らなければなりません。図からもわかるように，一つのステージに複数の鳥居が存在するところもあり，最終目的地の鎌倉までたどりつくには，いくつかのバリエーションも楽しめます。

鎌倉で待受けている頼朝を倒すには，ゲーム・スタート時に安駄婆が告げるように「八咫鏡」「草薙剣」「八坂瓊曲玉」の三種の神器が必要です。これ等の宝物は，いくつかの国のどこかに隠されており，ルートによっては鎌倉につくまでに手に入らない場合もあります。

主人公の景清のステータスは，攻撃力である「剣」，生命力を示す経本のロウソクで画面左下に表示されている「命」，そして命が無くなると落とされる黄泉の国で『地獄の沙汰も銭しだい』と蘇生の役にたつ「銭」の三

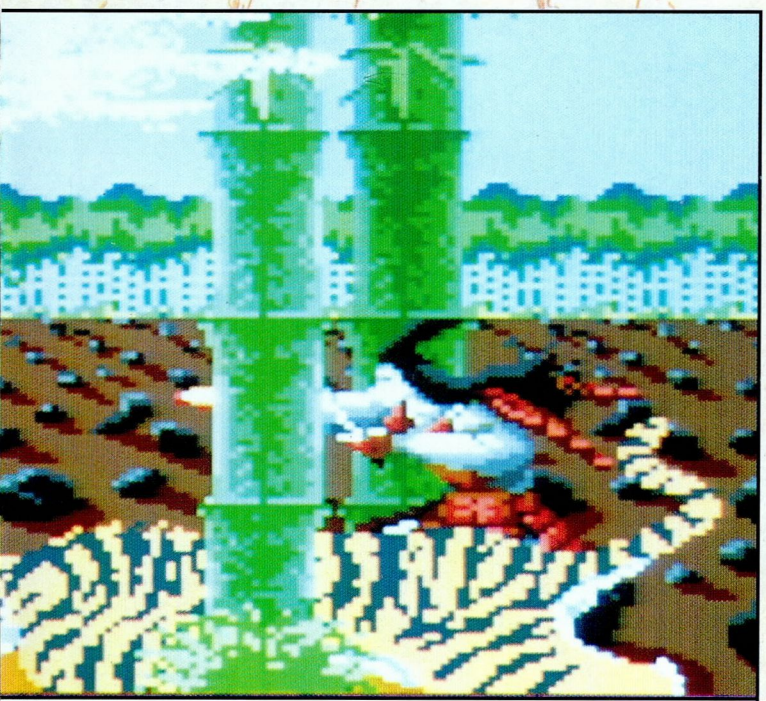




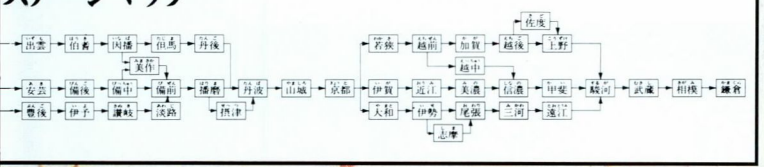
3モードに展開する,日本美あふれる画面

マイコンソフト
電波新聞社　出版販売部
〒141 東京都品川区東五反田1丁目11番15号
電話(03)445-6111

跳び宙を舞う
時代を舞台に繰広げられる
Oに再現!



ステージマップ



つがあります。これらは敵などを倒すと出現する，紫，緑，青，茶色の玉を集めることでアップさせることができます。また，ボーナス・ステージで大量に集めることも可能です。

源平討魔伝は，いわゆる〝アクション・ゲーム〟のジャンルに属するゲームですが，アイテムを集めたり，主人公の生命力や，武器（剣）の攻撃力を高めたりするＡＶＧやＲＰＧ的な要素も持っています。

このゲームの魅力は，様々なゲーム・システムの面白さにもありますが，１ドット単位で細かく，しかも緻密に設定された，ゲーム・バランスにあると言えるでしょう。

はじめてプレイする人にとっては，息もつかせぬスリルと迫力が，京都あたりまで辿りつけるようになった人には次々に解明できるようになるアイテムの持つ能力とその存在理由が，頼朝と対決できるほど成長した人にとっては小憎らしいまでに配慮された理不尽さのない位置・当り設定が，このゲームの奥深さを物語ってくれるでしょう。

そして，グラフィックスとキャラクター，効果音，肉声サンプリング，ＢＧＭが一体となって創りあげる平家復讐絵巻の世界にいつしか浸りきっているはずです。

個別の音楽作品としても十分に楽しめるMUSICのみの演奏モードや，キャラクター・テスト・モードなど，マニアの皆さんにも満足していただける機能も満載しました。

平家復讐絵巻

源平討魔伝

FOR X68000 絶賛発売中

源平討魔伝 for X68000 5インチディスク版 標準価格7,800円
源平討魔伝 for X68000 5インチディスク版 オリジナルジョイスティックXE-1PRO付 標準価格15,800円 限定発売
各パッケージには卓上のぼりが付いています。

マイコンソフ

DRAGON BUSTER

The soldier Clovis went to Mt.Dragon, running after Dragon. Dragon had taken Princess Celia as a hostage. But, monsters in the graveyard and ruins interrupted his way.

セリア王女がドラゴンにとらわれた！
王女の身とローレンス王国を救うために立ち上がったのは、
ドラゴンバスターとしての力を内に秘めたクロービスであった。
待ち受けるルームガーダーを倒し、アイテムを集めながら
最終ラウンドに潜むドラゴンを打ちやぶるのが使命だ。
ユメとロマンにあふれた冒険が今はじまる。

ドラゴンバスター

シャープX1 5インチFD版
標準価格6,200円

Presented by

DEMPA MICROCOMPUTER SOFTWARE

©株式会社ナムコ 発売元：電波新聞社 〒141 東京都品川区東五反田1-11-15 ☎03-445-6111

X68000 活用研究 III

X-BASIC 活用Q&A

塚越一雄著

X-BASIC活用Q&A

第1章 X-BASICのインストゥール

Q1 X-BASICの起動の方法は？ ……14
Q2 X-BASIC専用システムディスクの作り方は？ ……19
Q3 1行96文字モードでX-BASICを起動するには？ ……27
Q4 コンフィギュレーションファイルとは何か？ ……30
Q5 X-BASICの起動時オプションの使い方は？ ……35
Q6 X-BASICのプログラムをオートスタートさせるには？ ……37

第2章 OSとのインターフェース

Q7 X-BASICとプロセスとの関係は？ ……42
Q8 エラーコードを返すには？ ……49
Q9 ディレクトリを見るには？ ……54
Q10 カレントディレクトリ／カレントドライブを変更するには？ ……58
Q11 ファイルを削除／ファイル名を変更するには？ ……60
Q12 チャイルドプロセスを実行するには？ ……65
Q13 チャイルドプロセスとしてシェルを起動するには？ ……67

第3章 スクリーンエディタ

Q14 スクリーンエディタを利用するには？ ……70
Q15 テキストをセーブ／ロードするには？ ……74
Q16 行番号をカットしてテキストを表示・プリントするには？ ……76
Q17 文字列を検索するには？ ……79
Q18 まとまった行をまとめて編集するには？ ……81
Q19 1つの行を2行に分割したり，2つの行を1行に統合するには？ ……84
Q20 コントロールコードとは？ ……87
Q21 コントロールコードを利用した高度な編集を行うには？ ……90

第4章 X-BASICでつまずかないために

Q22 2つの命令実行形式とは？ ……96

Q23 命令の3形態とは？ ……99
Q24 関数の3形態とは？ ……103
Q25 プログラムを途中から実行するには？ ……108
Q26 なぜ変数宣言が必要なのか？ ……113
Q27 どのような場合に変数宣言を省略できるか？ ……116
Q28 文字型と整数型の使用上の違いは何か？ ……120
Q29 文字列型の使い方は？ ……123

第5章 構造化プログラミング

Q30 構造化プログラミングとは？ ……130
Q31 基本3構造とは何か？ ……134
Q32 多重分岐を美しく書くには？ ……143
Q33 X-BASICでサポートされている拡張繰り返し文は？ ……147
Q34 ユーザー関数の定義の仕方は？ ……149
Q35 なぜサブルーチンを使ってはいけないのか？ ……153
Q36 ローカル変数とは何か？ ……156

第6章 テキスト画面の制御

Q37 1行の表示文字数を変えるには？ ……164
Q38 X-68000のテキスト画面の表示可能行数は？ ……166
Q39 テキスト画面を32行×16行モードで使用するには？ ……167
Q40 スクロール範囲とは？ ……169
Q41 clsとchr$(12)の違いは？ ……171
Q42 画面を徐々に暗くするには？ ……174
Q43 カーソル位置を変更するには？ ……176
Q44 カーソルを消去するには？ ……178
Q45 現在のカーソル位置を知るには？ ……179
Q46 書式制御を使うには？ ……181
Q47 テキスト画面に色を付けるには？ ……183
Q48 RGB方式でカラーコードを作るには？ ……186

CONTENTS

第7章 ディスクアクセス

Q49 ディスクファイルにアクセスする手順は？ ……190
Q50 具体的なディスクアクセスの方法は？ ……193
Q51 error off は何のために使う？ ……197
Q52 "w"モードと"c"モードの違いは？ ……199
Q53 既存ファイルの保護を考慮するには？ ……202
Q54 fseek( )は，何に使う？ ……207
Q55 ディスクの残り容量を調べるには？ ……210

補 章 ファンクションキー活用テクニック

1．簡単なテクニック ……212
- ●ディスクのイジェクト
- ●シフトモードの内容を表示する

2．こんなテクニックもある ……212
- テクニック1　関連内容を SHIFT モードに設定する
- テクニック2　改行は@Mを指定する
- テクニック3　カーソル行をクリアするには@Eを指定する
- テクニック4　カーソル以後をクリアするには@Zを指定する
- テクニック5　自動的に挿入モードにするには，@Aを指定する
- テクニック6　カーソルを左に戻すには，@[を指定する

3．ファンクションキーの設定例 ……219

付 録 X-BASIC クイックリファレンス

1．X-BASIC の特色 ……224
2．起動法 ……224
3．オペレーション ……226
4．プログラム ……227
5．言語仕様 ……227

まえがき

栄光あるX-68000活用研究シリーズの第3巻をお届けします。

一昨年，私がこのシリーズの第1巻を執筆している頃は，まだX-68000が世に出る前のことでした。当時は，まだPC-9801全盛の真っ最中で，果たして新しい16ビットマシンが世に受け入れられるか不安がありました。しかし，X-68000が発売になるや，その危惧も危惧でしかないことがすぐにわかりました。やはりマニアが作ったマシンは強かった。X-68000は大好調。お陰様で私の第1巻が，イコールX-68000関連書籍の第1号になるという栄誉を受けることができたのです。書籍の方も好評で，増刷していただくことができました。また第1巻で呼びかけた

「もしご支持を頂けるなら，第2，第3のX-68000活用研究も夢ではありません。私以外の優秀な方が参加してくれるかもしれません」

もすぐに実現しました。優秀なX-68000ファンの手により，すばらしい第2巻の刊行に成功したからです。

本書は，その第2巻を受けたX-BASICのテクニカルQ&A集です。この形式を採用したのは，解説書では乗りにくいテーマを取り上げたかったためです。深遠なるX-BASICの思想を浮き彫りにしたかったからです。X-68000はマニアックなマシンなため，ともするとアマチュア的なプログラミングに陥りやすいという側面をもっています。それは，それでよいことです。しかし，ある程度プログラミングに慣れてきたら，もう少し科学的な側面からプログラミングを捉えなおしてみるのも有益なことです。構造化BASICとしてのX-BASICは，充分それに応えるだけの実力を持っています。X-BASICの持つ，その科学的な側面を言語の内面から解説してみたい——これが本書の狙いでした。少しでもその狙いに成功していれば幸いです。

さて，本書執筆中にもX-68000は，続々新しい動きを見せています。

C COMPILER PRO-68K

もその1つです。次回の私のX-68000活用研究は，おそらくC言語がテーマになると思います。よろしければご要望をお聞かせください。

最後になりましたが，本書執筆にあたりお世話になった電波新聞社書籍出版部，およびMULTIマイコン研究会の皆様にお礼申しあげます。

1988年2月16日

MULTIマイコン研究会　塚越一雄

第 1 章

X-BASICのインストゥール

Q 1　X-BASIC の起動の方法は？

Q 2　X-BASIC 専用システムディスクの作り方は？

Q 3　1行96文字モードで X-BASIC を起動するには？

Q 4　コンフィギュレーションファイルとは何か？

Q 5　X-BASIC の起動オプションの使い方は？

Q 6　X-BASIC のプログラムをオートスタートさせるには？

Q1 X-BASICの起動の方法は?

A1

X-BASICをマスターすることで，X-68000を自由にあやつることができるようになります。X-BASICは汎用的な言語ですので，ほとんどの応用プログラムを記述することが可能です。**ミュージックエディタ**や**グラフィックエディタ**，また**表計算プログラム**や**シミュレーションゲーム**さえも作成可能です。

X-BASICのプログラムは，最終的にはCのプログラムに置き換えられ，**機械語**に落とすことができます。ですからBASICとはいえ，かなり実用的なプログラムを開発することができます。

この素晴らしいX-BASICを使うには，X-BASICそのものを起動できなければ始まりません。X-BASICへの出発の第1歩は，それを起動することから始まります。

X-BASIC起動の3方法

X-BASICを起動する方法は，次の3つが考えられます。

起動の3方法

〈1〉ビジュアルシェルから起動する
〈2〉コマンドシェルから起動する
〈3〉X-BASIC専用のシステムディスクを作成し，X-BASICをオートスタートさせる

〈3〉のX-BASIC専用のシステムディスクを作成してしまうのが最も便利です。その方法は次のQ2で説明するとして，ここではその基本として，〈1〉，〈2〉による方法を説明しておきます**【第1-1図】**。

以下では，X-68000オリジナルのシステムディスクによる作業例を示します。

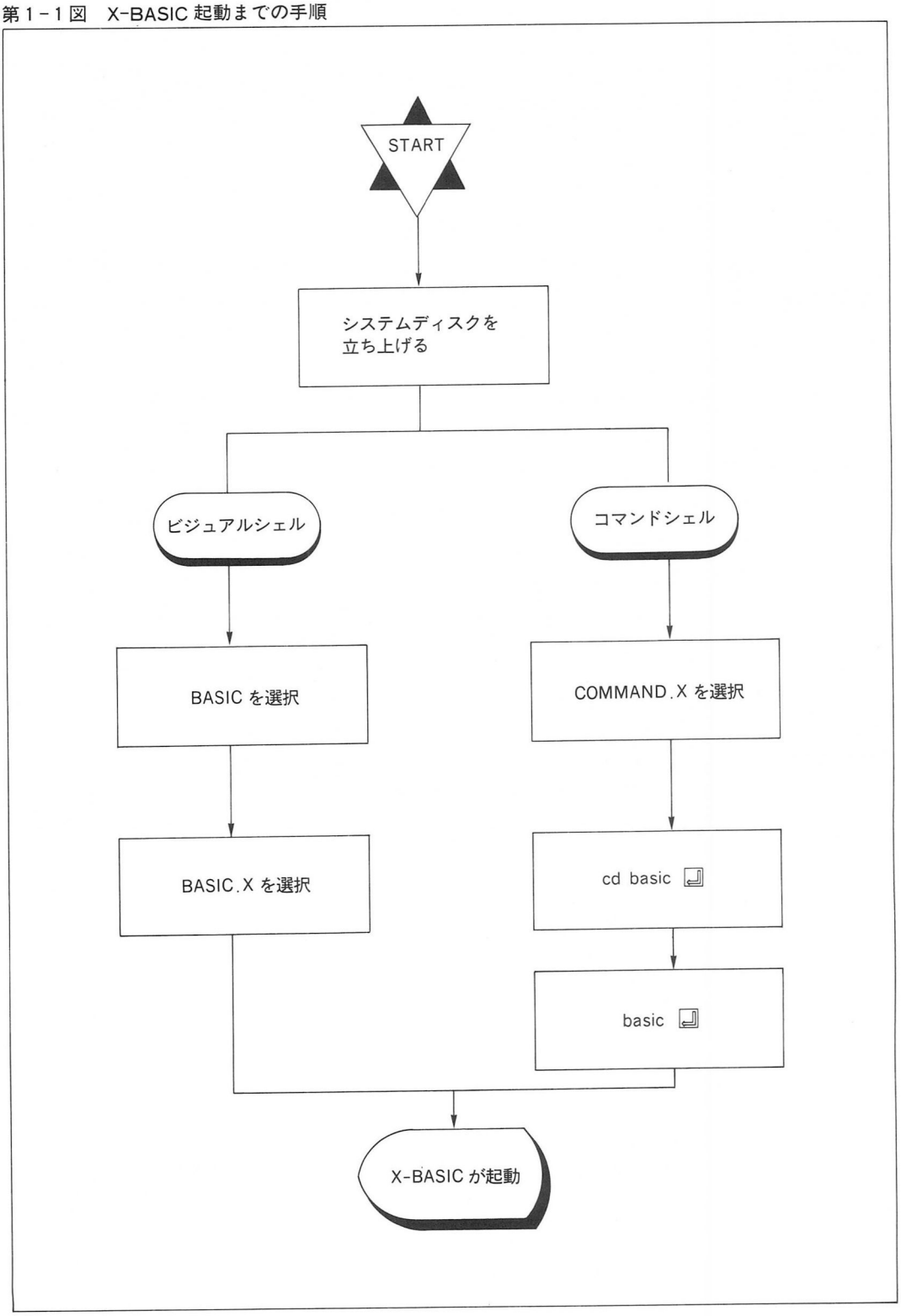
●ビジュアルシェルからX-BASICを起動する

手　順

ビジュアルシェルからX-BASICを起動する手順は，

フォルダー BASIC をオープンする
↓

第 1 - 1 図　X-BASIC 起動までの手順



BASIC.X をオープンする

の2段階を経て行います。それでは，その手順を具体的に説明しましょう。

※
ビジュアルシェル
アイコン
クリック
フォルダー
については，「X-68000活用研究」第1巻参照

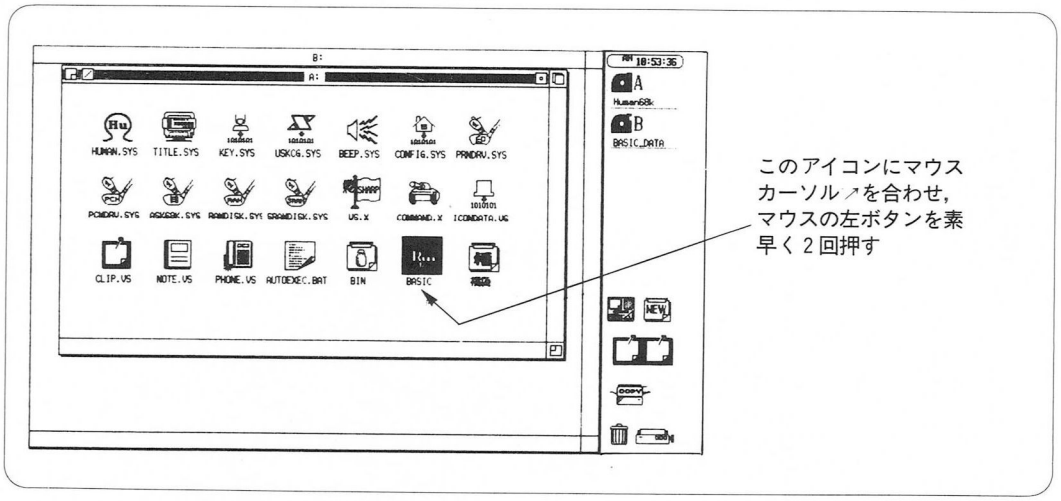
X-68000オリジナルのシステムディスクを起動しますと，最初に**ビジュアルシェル**が立ち上がります。たくさんの**アイコン**が表示されますので，その中から

BASIC

という名前のアイコンを選択します。すなわちマウスを動かし，マウスカーソル↗をBASICと表示されているアイコンに重ねます**【第1－2図】**。

そして，**左クリック**(左ボタンを素早く2回押す)します。すると，BASICのフォルダー（ウィンドウ）が現れます**【第1－3図】**。

第1－2図　ビジュアルシェルでBASICを選択



BASICのフォルダーには，BASIC関係のアイコンが納められています。その中の

BASIC．X

というアイコンを選択します。すなわちマウスを動かし，マウスカーソル↗をBASIC．Xと表示されているアイコンに重ねます。そして，左クリック(左ボタンを素早く2回押す)します。これで，X-BASICが起動します**【第1－4図】**。

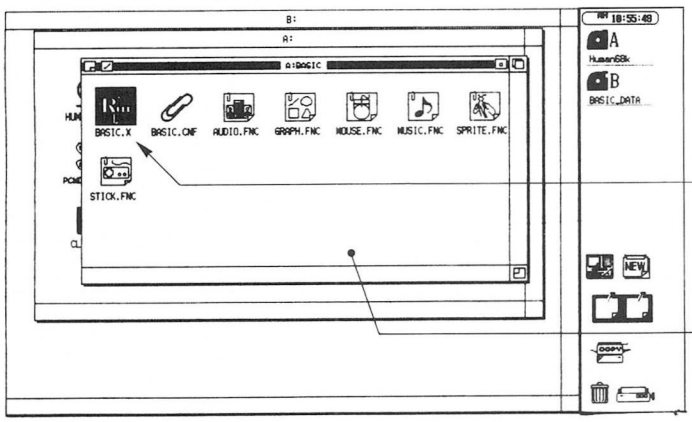
●コマンドシェルからX-BASICを起動する

X-68000オリジナルのシステムディスクを起動しますと，最初にビジュアルシェルが立ち上がります。たくさんのアイコンが表示されますので，その中から

COMMAND．X

を選択します。すなわちマウスを動かし，マウスカーソルをCOM-

第 1 - 3 図　BASIC のフォルダーで BASIC.X を選択



このアイコンにマウスカーソル↗を合わせ，マウスの左ボタンを素早く 2 回押す

これが BASIC フォルダー。BASIC 関係のアイコンが 8 個収められている

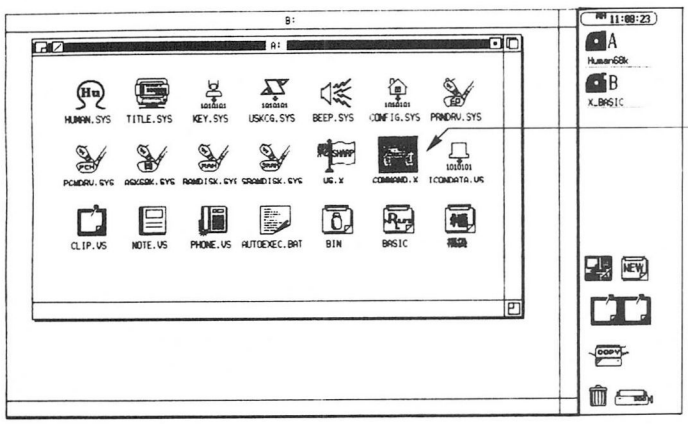
第 1 - 4 図　X-BASIC が起動

```
X-BASIC for X68000 version 1.00
Copyright 1987 SHARP/Hudson
 128 kbytes free
Ok
```

X-BASIC のオープニングメッセージ

ユーザーが使用できるフリーエリアが128K バイトあることを示す
この値は後で説明するコンフィギュレーションファイルの free で指定できる

第 1 - 5 図　COMMAND.X を選択



このアイコンにマウスカーソル↗を合わせ，左ボタンを 2 回押す

MAND.Xと表示されているアイコンに重ねます【第1－5図】。

そして，左クリック（左ボタンを素早く2回押す）しますと，コマンドシェルが起動します【第1－6図】。

第1－6図　コマンドシェルが立ち上がる

```
Command version 1.00
A>
```

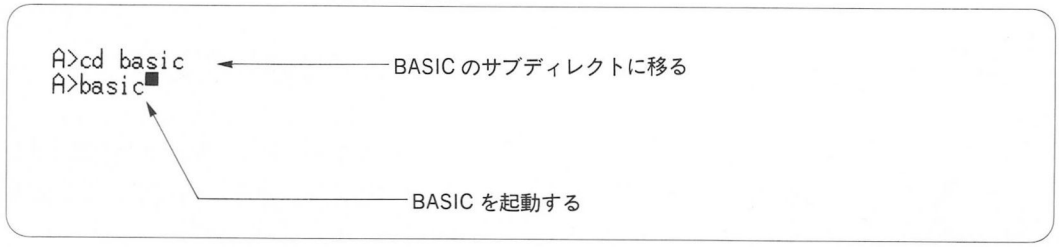
コマンドシェルが起動しましたら

cd　basic ↵

basic ↵

と入力します。これで，X-BASICが起動します【第1－7図】。

第1－7図　コマンドシェルでBASICを起動



X-BASICを終了させる

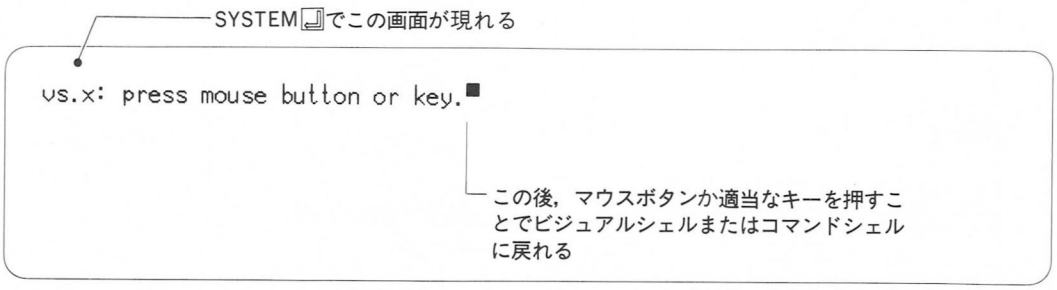
X-BASICを終了させる——すなわちX-BASICを抜け，元のビジュアルシェルやコマンドシェルに戻るには

system

system ↵

と入力します。すると，**第1－8図**のような画面が現れますので，マウスボタン（左でも右でも可）または適当なキーを押します。これで，X-BASICを起動した元のシェルに戻ることができます。

第1－8図　ビジュアルシェルに戻る



Q 2 X-BASIC専用システムディスクの作り方は？

A 2

これから X-BASIC を使用する機会が多くなるでしょう。そこで，

X-BASIC 専用システムディスク

を作成しておくことをお勧めします。１度これを作成しておきますと

メリット

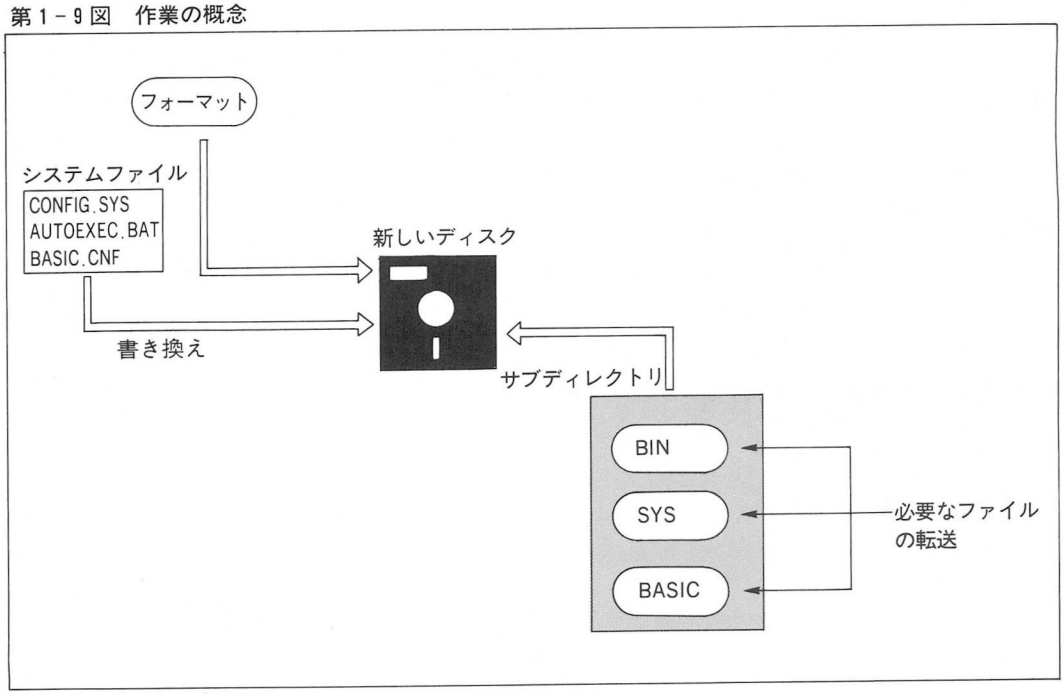
- X-BASIC がオートスタートする
- １枚のディスクに漢字入力機能を組み込める

といったことが可能になります。ですから残りのディスクは，データディスク専用として X-BASIC のプログラムをセーブするのに使うことができます。

専用システムディスク作成の工程

X-BASIC 専用システムディスクを作成する手順は，おおまかに**第１－９図**のようになります。以下，詳細にこの作業の方法を説明していくことにします。

第１－９図　作業の概念



●システムディスクの作成

作業の第1工程は，新しいシステムディスクを作成することです。すなわち

新しいディスクをフォーマット
↓
そこにシステムファイルを転送する

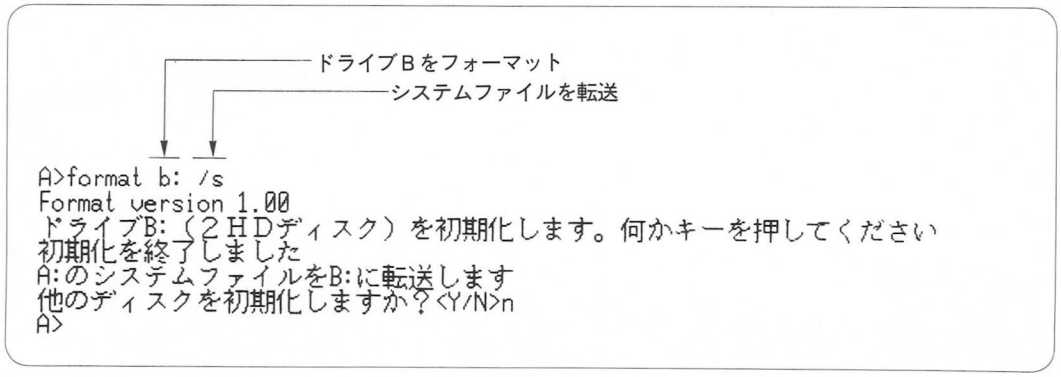
といったことを行います。この作業を1度に実行するには

ディスクの挿入

ドライブ0 —— **X-68000システムディスク**
ドライブ1 —— **新しいディスク**

を挿入し，formatコマンドを実行します。その処理の様子を**第1-10図**に示しておきますので，これを参考に作業を進めてください。

第1-10図　システムディスクを作成する



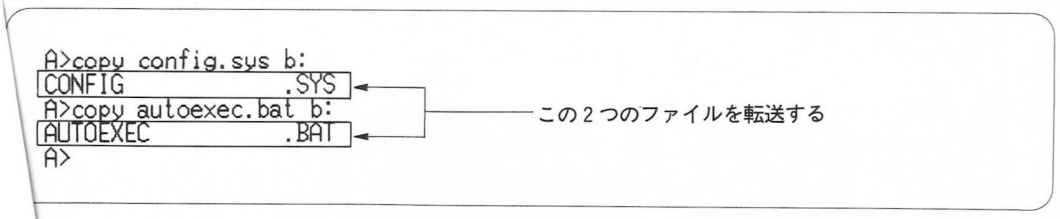
●システム初期化用ファイルの転送

次にシステムの初期化に必要な2つのファイル

CONFIG．SYS
AUTOEXEC．BAT

を新しいシステムディスクに転送します。その処理の様子を**第1-11図**に示しておきますので，これを参考に作業を進めてください。

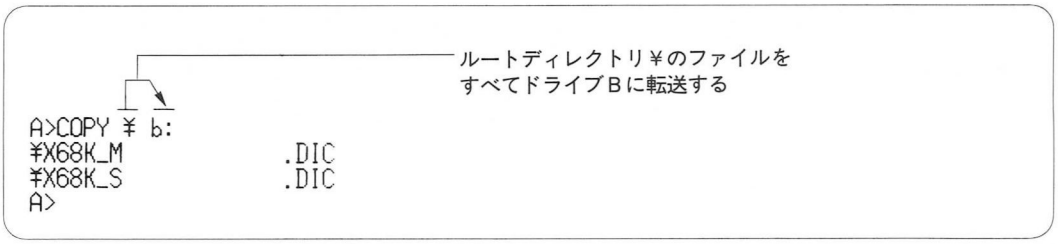
第1-11図　システム初期化用ファイルの転送



●辞書ファイルの転送

次に２つの**辞書ファイル**を転送します。ドライブ０のシステムディスクを取り出し，代わりに辞書ディスクを挿入します。そして，**第１-12図**を参考にファイルを転送します。なおすでに日本語ワードプロセッサを多用している方は，日本語ワードプロセッサに含まれている辞書ファイルを転送した方がよいかもしれません。辞書の学習や登録がかなり進んでいるからです。

第１-12図　辞書ディスクの転送



●サブディレクトリの作成

次に３つのサブディレクトリを作成し，ここに必要なファイルを転送します。

サブディレクトリ

BIN ―― 外部コマンドを転送する
SYS ―― SYS ファイルを転送する
BASIC ―― X-BASIC 関係のファイルを転送する

まずドライブ０をシステムディスクに戻します。そして，md コマンドを使用して３つのサブディレクトリを作成します**【第１-13図】**。

次に**外部コマンド**を転送します。本当はシステムディスクに含まれるすべての外部コマンドを転送したいところです。しかし，新しいディスクには辞書ファイルも同居させていますので，容量が足りなくなるでしょう。とりあえずは**第１-14図**に示したファイルだけを転送しておき，必要が起こったところで他のファイルも転送するとよいでしょう。

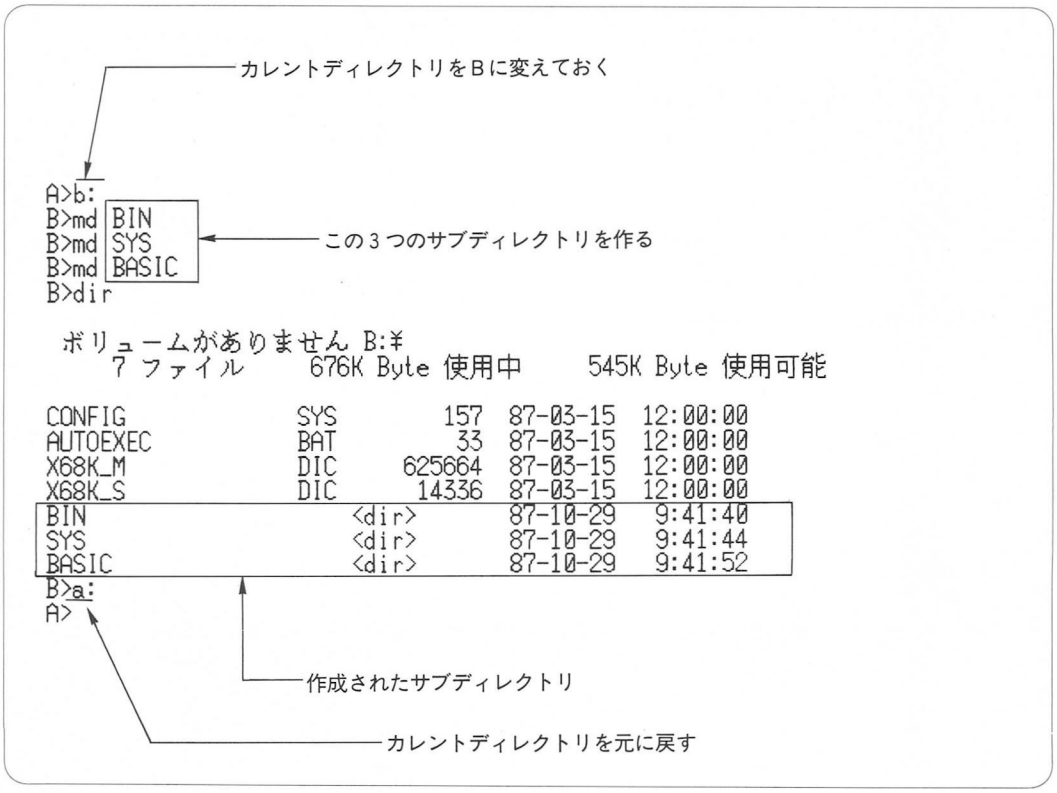
SYS ファイルの転送は**第１-15図**のように行います。

また BASIC 関係のファイルの転送は**第１-16図**のように行います。

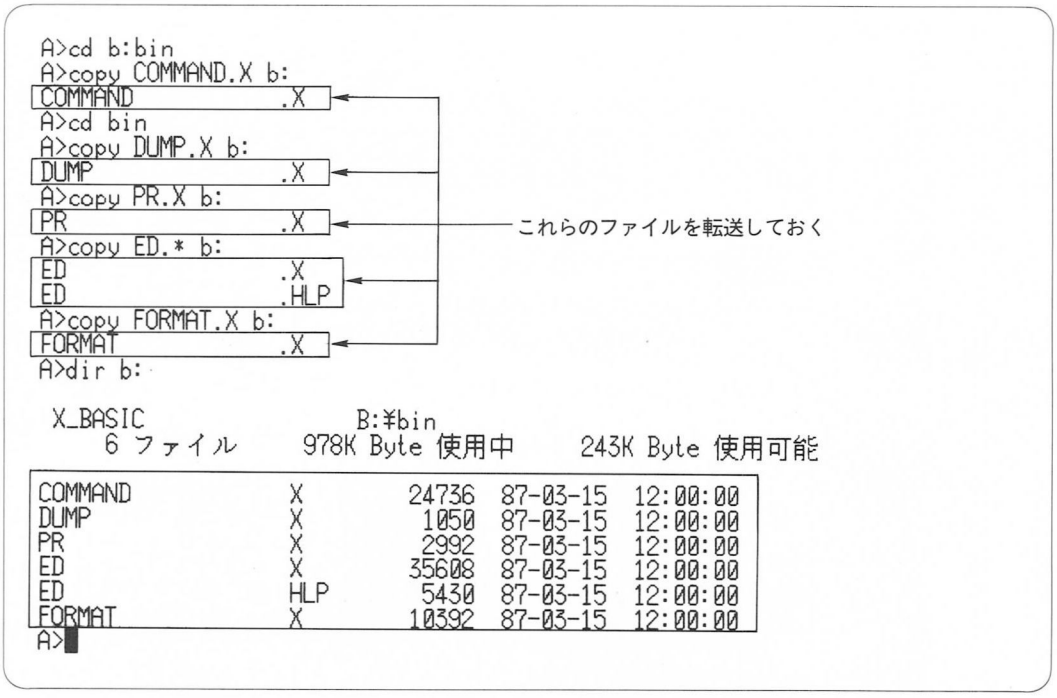
● CONFIG．SYS の書き換え

先に転送した CONFIG．SYS の内容を，新しいシステムディスクに合わせて書き換えます。もしスクリーンエディタ ED の使い方をご存じでし

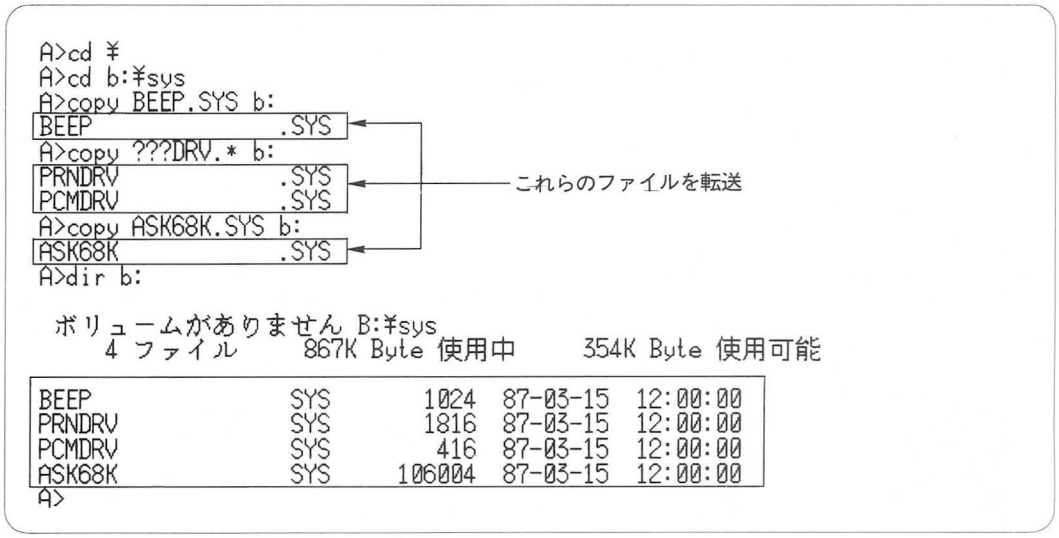
第1-13図　サブディレクトリを作る



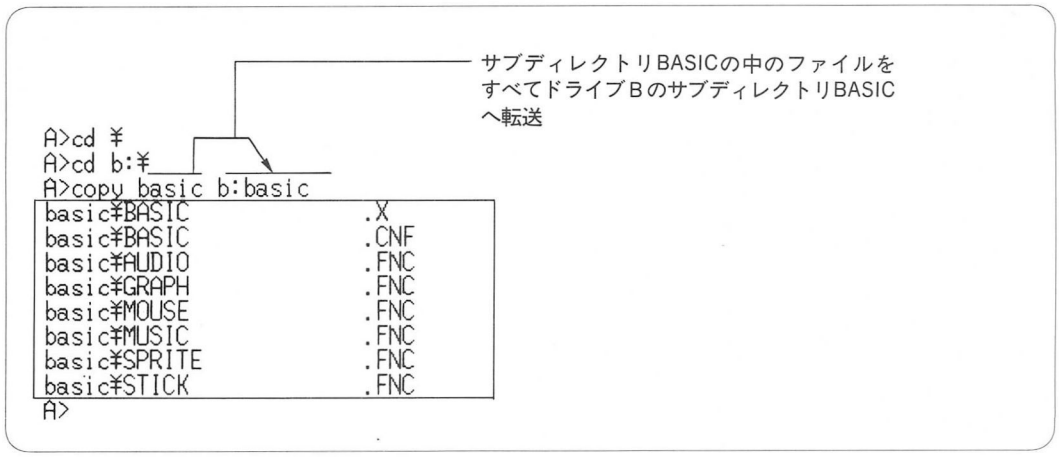
第1-14図　外部コマンドの転送



第 1 -15図　SYS ファイル（ドライバ）の転送



第 1 -16図　X-BASIC 関係のファイルの転送



たら，それを使用すると簡単です。ED の使い方がわからない方は，少々面倒ですが**第 1 -17図**の方法でも可能です。この時，気を付けなければならないのは

^I

の入力です。これは，**TAB キー**を押したものです。

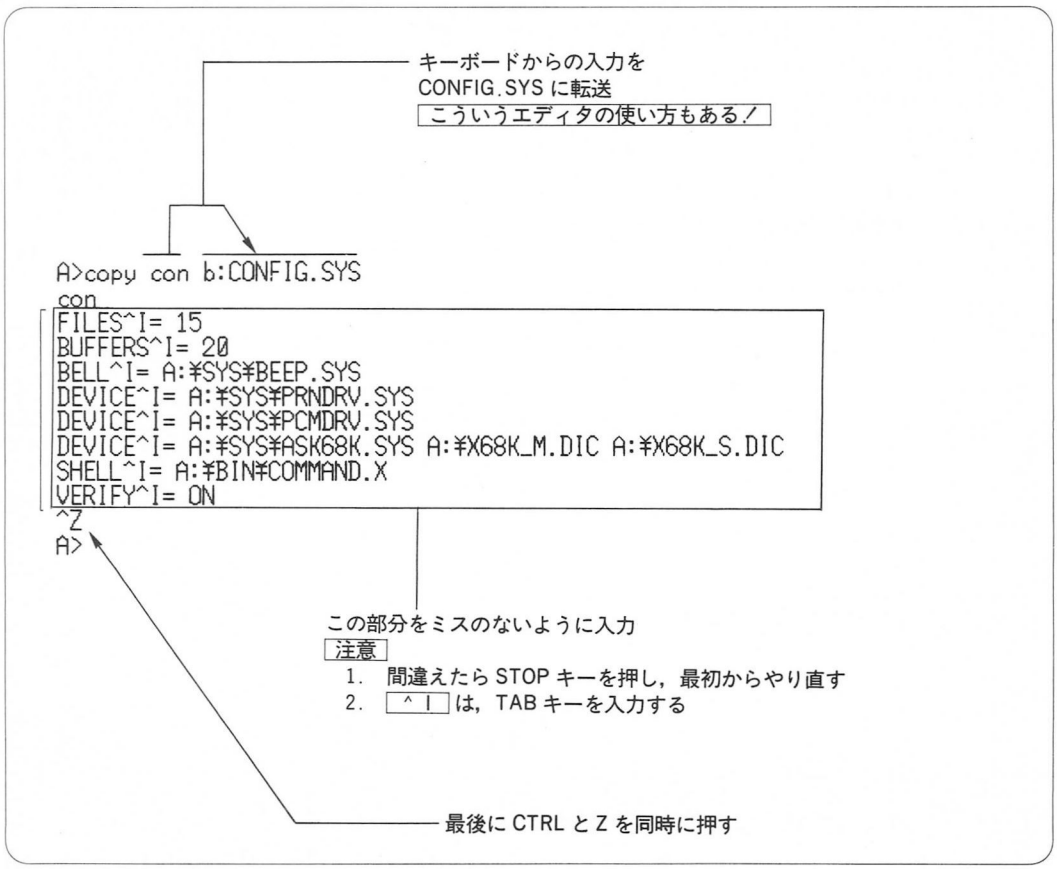
いずれの方法にしても**第 1 -18図**の内容を持った CONFIG.SYS が出来上がれば結構です。

● AUTOEXEC. BAT の書き換え

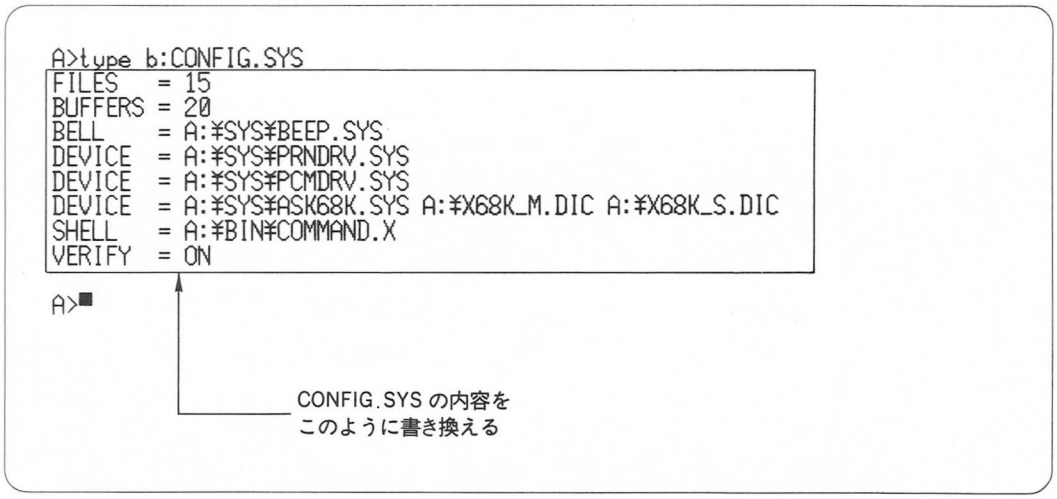
AUTOEXEC. BAT も新しいシステムディスクに合わせて書き換えます。といっても最後に

BASIC

第１-17図　CONFIG.SYS の書き換え



第１-18図　修正を受けた CONFIG.SYS



という１行を付け加えるだけです。ですから**第１-19図**を参考に作業を行うと楽です。こういう type コマンドやリダイレクトの使い方があることを覚えておくと便利でしょう。

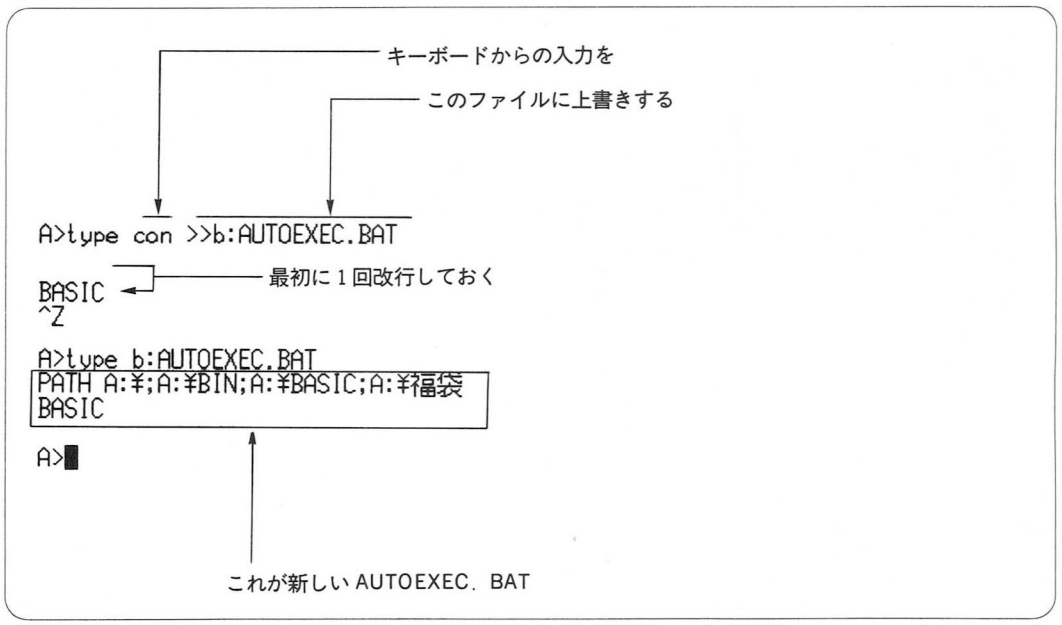
〈注意〉

BASICを入力する前に最初に↵で1行改行しておくことを忘れないでください。さもないと

……副袋BASIC

のように，BASICが前の行にくっついてしまいます。

第1-19図　AUTOEXEC.BATの書き換え

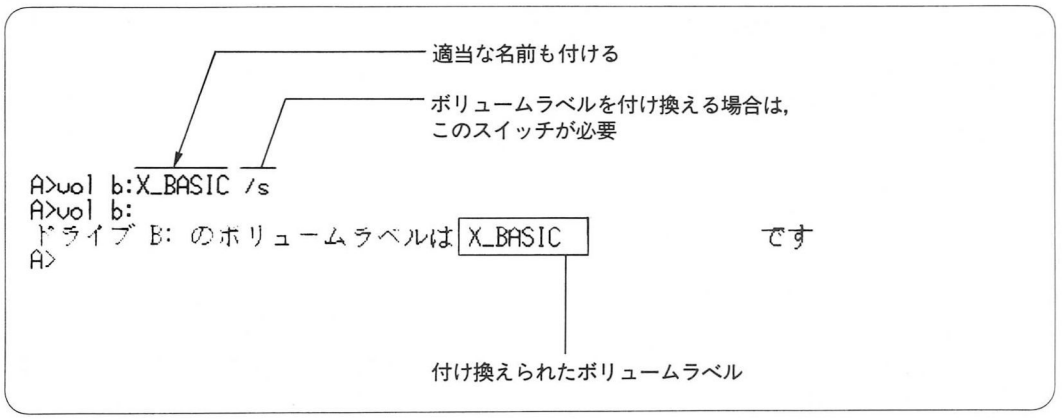


●ボリュームラベルを付ける

ボリュームラベル

以上で，作業はほとんど終わりです。ただし，新しいシステムディスクに**ボリュームラベル**を付けておくと便利です。**第1-20図**を参考に作業してみてください。

第1-20図　ボリュームラベルを付ける

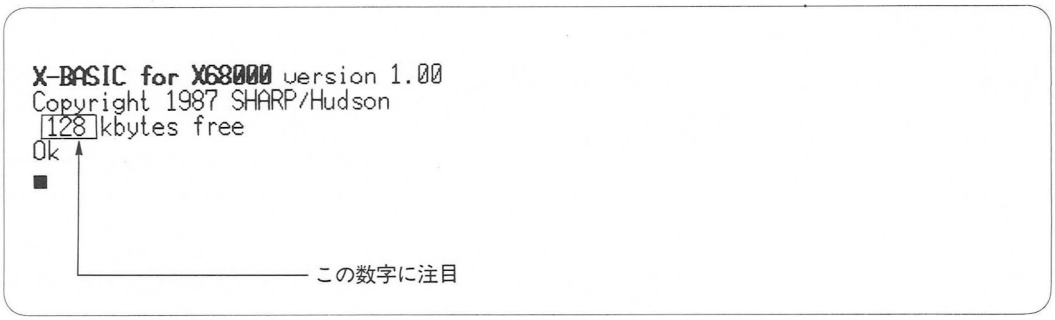


X-BASICをオートスタートさせる

できあがったX-BASIC専用のシステムディスクをドライブ0に挿入し，リセットしてみてください。**第1-21図**のようにX-BASICがオートスタートすれば成功です。Q 1の方法でX-BASICを起動するより，どんなにか便利であることがおわかりでしょう。

念のため，漢字入力が可能かをテストしてみてください。

第1-21図　X-BASICがオートスタート



Q 3 1行96文字モードで X-BASIC を起動するには？

A 3

なぜ1行64文字モードは不便か？

普通，X-BASIC を起動しますと

1行**64文字**モード

デフォルトのモード

で立ち上がります。Q 2で作成した X-BASIC 専用のシステムディスクで立ち上げても同じです。この方が文字が大きくて読みやすいのですが，不便なこともあります。たとえば files コマンドでファイルの一覧を表示させますと，1つのファイルが2行にまたがってしまい，読みにくいリストになってしまいます。

1行の文字数は，width コマンドで変えることができます。しかし，毎回 X-BASIC を起動する度に width を実行するのは面倒です。そこで，いきなり

1行**96文字**モード

で X-BASIC を起動する方法をご紹介しましょう。合わせて，X-BASIC のフリーエリアを大きくする方法も紹介しておきます。フリーエリアを大きくしておきますと，大きなプログラムを作成したり，大きな配列変数を取ることができるようになります。

〈補　注〉

files や width のコマンドについては，後の方のQで説明します。

X-BASIC の初期状態をコントロールするには？

起動時の X-BASIC の状態を変更するには，大きく次の3つの方法があります。

3つの方法

〈1〉コンフィギュレーションファイルを書き換える
〈2〉オプションを指定する
〈3〉自動実行ファイルを利用する

ここでは，〈1〉の方法で説明しておきます。なお**コンフィギュレーションファイル**そのものについては，次のQ 4で説明します。

フリーエリアの変更

コンフィギュレーションファイルとは，BASICを起動する際に参照されるテキストファイルで，X-BASICの機能や初期状態をいろいろコントロールするのに使われます。X-BASICを起動し，

load@ "basic¥BASIC．CNF"↵

と入力し，続いて

list↵

と入力してみてください（以下，**第1-22図**の作業例参照）。コンフィギュレーションファイルの内容が表示されます。その最初のところに

free

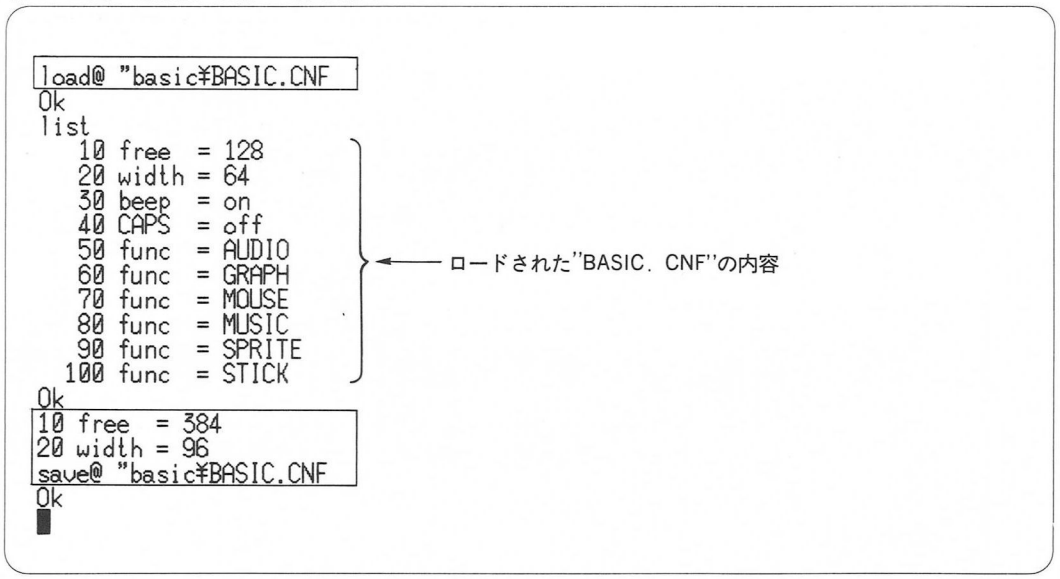
free＝128

というのがあります。これが先にちょっと触れましたX-BASICのフリーエリアを指定するものです。この場合，ユーザーが使用できるフリーエリアが128Kバイトあることを示しています。そこで，

10　free＝384↵

と入力してみてください。これで，フリーエリアを384Kバイトに変更することができます。

第1-22図　BASIC.CNFの書き換え



1行の文字数を変更

同様にして，2行目に

width

width＝64

というのがあります。これが，X-BASIC の初期画面を 1 行64文字モードにしている元凶です。これも

```
20　width＝96↵
```

と入力することで，書き換えてしまいます。最後に書き換えたコンフィギュレーションファイルの内容をディスクにセーブしておきます。それには

```
save@　"basic￥BASIC．CNF"↵
```

と入力します。

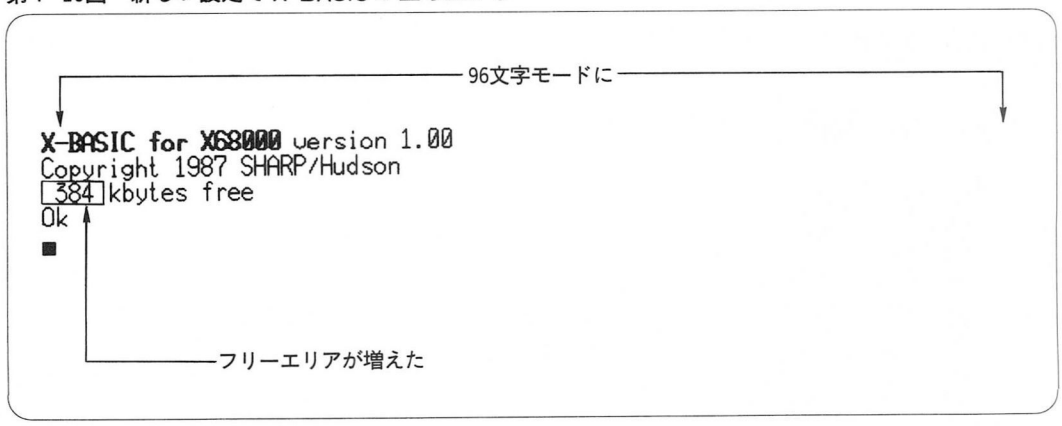
結果を見る

以上で，コンフィギュレーションファイルの書き換えはおしまいです。リセットしてみてください。今度は，1 行96文字モードで立ち上がるはずです。また，X-BASIC のオープニングメッセージの 3 行目が

```
384　kbytes　free
```

となっているでしょう。これが，フリーエリアの大きさを表しています(**第 1 -23図**)。Q 2 の**第 1 -21図**と比較してみてください。

第 1 -23図　新しい設定で X-BASIC が立ち上がる



Q 4 コンフィギュレーションファイルとは何か？

A 4

コンフィギュレーションファイルの中身を見るには？

コンフィギュレーションファイルは，Q 3でも軽く触れましたようにX-BASICを起動する際に参照される特別なテキストファイルのことです。X-BASICの機能や初期状態をいろいろコントロールするのに使われます。

コンフィギュレーションファイルのファイル名は，決まっていて

BASIC．CNF

BASIC．CNF

となっています（次のQ 5で説明しますが，違う名前を使用することもできます）。コンフィギュレーションファイルを実際に見てみるには，次のようにします。

コンフィギュレーションファイルの中身を見る

〈1〉X-BASICを起動する
〈2〉「system↵」と入力してX-BASICを終了させ，コマンドシェルに入る
〈3〉「cd　BASIC↵」で，サブディレクトリBASICに入る
〈4〉「type　BASIC．CNF↵」と入力する

以上の操作で，コンフィギュレーションファイルの中身を見ることができます**【第1-24図】**。

コンフィギュレーションファイルの書式

このようにコンフィギュレーションファイルは，いくつかの行で構成されています。また各行は

各行の構成

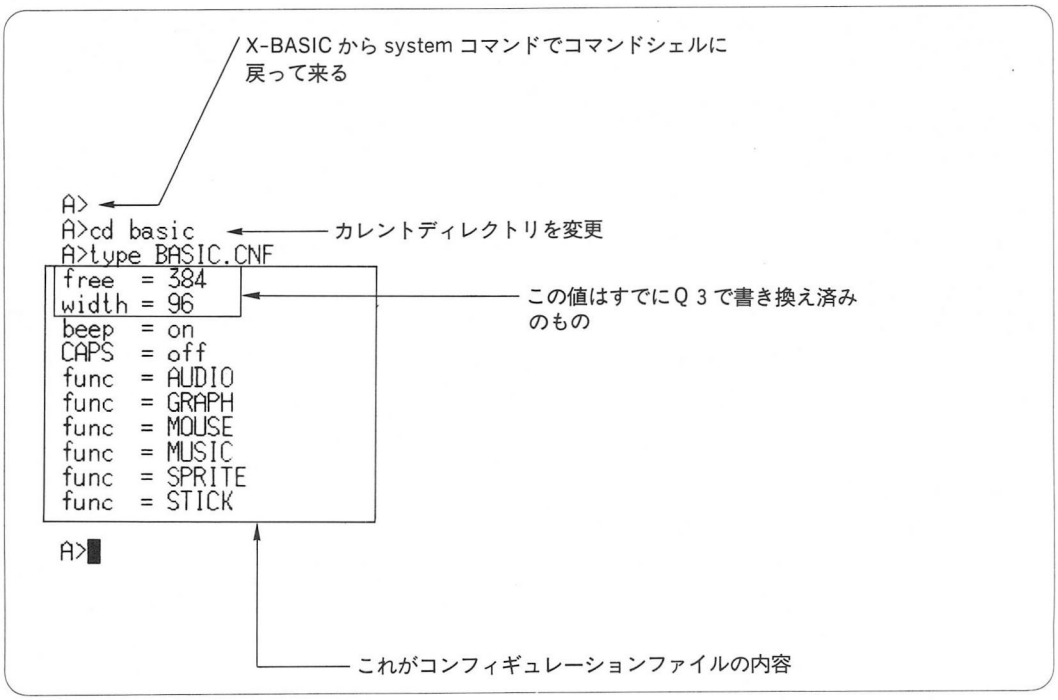
特別な単語＝その特別な単語に設定する内容

のような構成になっています。コンフィギュレーションファイルで使える特別な単語は，次のものがあります。

コンフィギュレーションファイル用コマンド

width
beep

第 1 -24図　コンフィギュレーションファイルの中身を見る



caps
free
func

このうち func は，X-BASIC で使用する外部関数のライブラリ名を指定するもので，その詳細は後の方で説明することにします。ここでは，残りの 4 つについて詳しく説明しておきます。

● width

X-BASIC が起動した時，1 行の文字数をいくつにするかを指定するのに使います。次のいずれかが，使用可能です。

width

width＝64… 1 行64文字モードにする時
width＝96… 1 行96文字モードにする時

コンフィギュレーションファイルで width を省略しますと，1 行**64文字モード**で X-BASIC が起動します。

● beep

これは，**ビープ音**を制御するのに使います。ビープ音は，通常，エラー発生時の警告音に使われ，次のいずれかで聞くことができます。

ビープ音

X-BASICでエラーが発生した時
beepという命令を使った時

ところが夜間等，あまりビープ音を出したくない時があります。そんな時はこのコンフィギュレーションファイルのbeepを使って，ビープ音を止めることができます。

beep

beep＝on…ビープ音を出す
beep＝off…ビープ音を止める

コンフィギュレーションファイルでbeepを省略しますと，ビープ音が聞こえる状態でX-BASICが起動します。

● caps

これは，文字どおり〈大文字を制御〉するもので，次の2通りの使い方があります。

caps

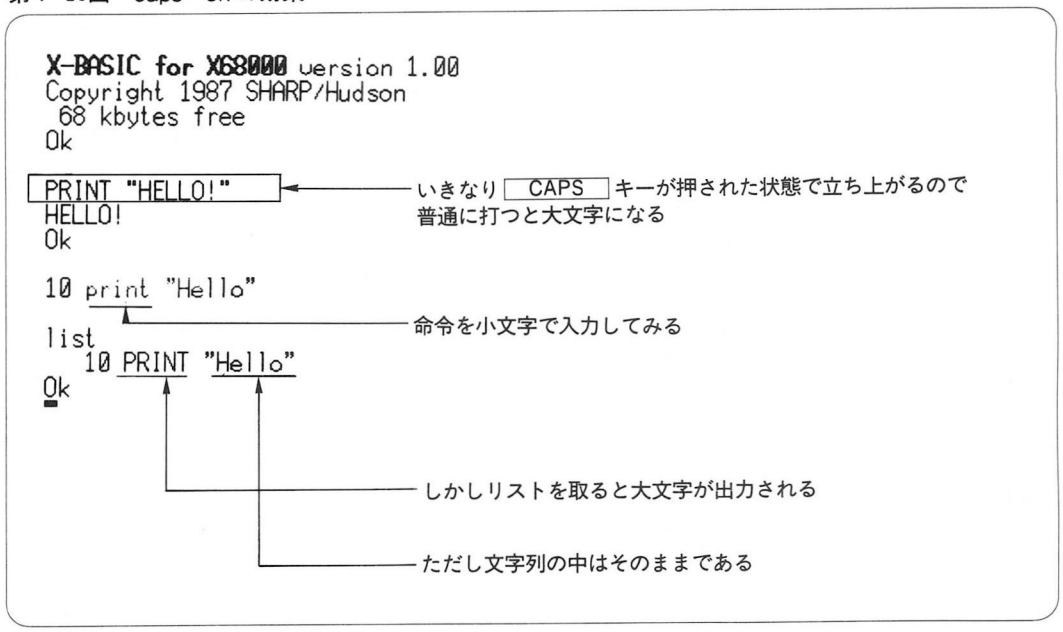
caps＝off
caps＝on

この〈大文字を制御〉には，次の2つの意味が含まれています。

① X-BASICが起動した時のCAPSキーの状態
②リスト出力時の命令語の表示

たとえばcapsをonにして，X-BASICを起動しますと，いきなり
CAPSキーが押された状態——赤いインジケータが点灯します
で起動します。それともう1つ，capsはX-BASICの**プログラムリスト**を制御します。X-BASICはわりと自由度が高く，プログラムリストの中で命令を大文字でも小文字でも表現することができます。昔のBASICは，命令を**大文字**で使うことが多かったので，小文字で入力した命令もリストを取ると大文字で出力されるようになっていました。capsをonにしておきますと，X-BASICでもそのようになります **【第1-25図】**。

第１-25図　caps＝on の効果



ただし，最近はＣ言語が主流になってきています。Ｃでは，命令を小文字で表現します。X-BASIC も最終的にはＣに変換することができますので，今後は X-BASIC でも小文字を使うのがよいと思います。caps を off にしておきますと，たとえ大文字で入力した命令でもリストを取ると**小文字**で出力されます。ちなみにコンフィギュレーションファイルで caps を省略しますと，caps が off の状態で X-BASIC が起動します。

● free

これは，X-BASIC で使用する**フリーエリア**の大きさを指定するものです。フリーエリアは，メモリ上に取られた X-BASIC 用の領域のことで，次の用途に使われます。

- X-BASIC のプログラムを置く領域
- 変数（単純変数や配列変数）の値を置く領域
- スタック領域

　繰り返し命令を実行中に繰り返しの状態を覚えておいたり，関数やサブルーチンの呼び出しでメインルーチンの位置を覚えておいたりの用途に使われる

ですからデータベースを扱う時のように，大きな配列を使用する場合は，フリーエリアを大きく取るとよいでしょう。その指定の仕方は，たとえばフリーエリアに100K バイトの大きさが取りたいなら

free

```
free＝100
```

のように記述します。フリーエリアの大きさとしては，

指定可能な値→　　**1～65535**（単位はKバイト）

までが指定可能です。ただし，当然ながら自分の機械のメモリの大きさにより上限の制限を受けます。

freeを省略しますと，**68Kバイト**の大きさが取られます。8ビット機の場合，システムを含めて最大で64Kバイトですから，これだけでも通常の用途なら十分です。ちなみにX-BASICが起動した段階で，X-BASICが

予約値→　　**96バイトの領域**

を確保してしまいますので，ユーザーが使えるのはfreeで指定した値よりも96バイト分少なくなります。

Q 5 X-BASIC の起動時オプションの使い方は？

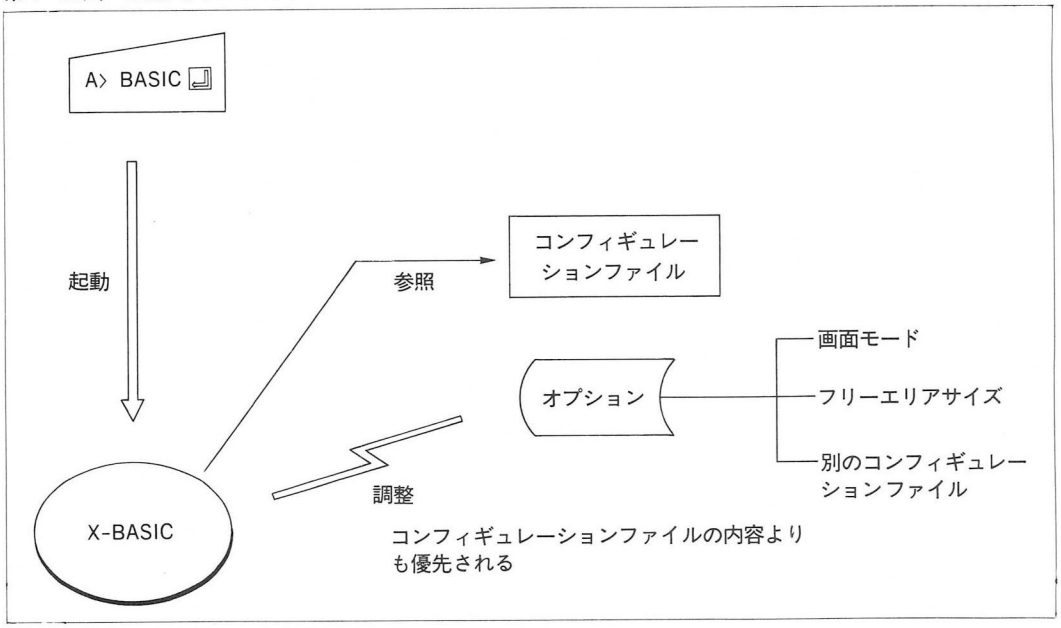
A 5

X-BASIC の起動時の状態は，Q 4 でも説明しましたようにコンフィギュレーションファイルの内容で制御することができます。しかし，時には beep をオフで立ち上げてみたいとかのように，**一時的**に作ってあるコンフィギュレーションファイルとは異なる設定で立ち上げたい場合もあります。そのようなちょっとした変更ならわざわざコンフィギュレーションファイルの内容を書き換えなくても

起動時オプション　　**起動時オプション**

を付けることにより，X-BASIC の環境を変えることができます【**第 1 -26 図**】。

第 1 -26図　起動時オプション



起動時オプションの書式

通常，コマンドシェルから X-BASIC を起動するには

BASIC

と入力します。しかし，その BASIC の後に／(**スラッシュ**)を付けることにより，いろいろなオプションを指定することができます。それが X-BASIC の起動時オプションです。

書　式

BASIC ／**オプション**↵

X-BASICで使える起動時オプションは，次の3つがあります。

オプションの種類

／c**「コンフィギュレーションファイル名」**
／f**「フリーエリアサイズ」**……1～65535のいずれか
／w**「1行の文字数」**……64または96

たとえば

BASIC ／w64↵

としてX-BASICを起動しますと，たとえコンフィギュレーションファイルの内容が96文字モードになっていようとも1行64文字モードで起動します。また，フリーエリアのサイズを一時的に変更したい場合は**fオプション**を使います。たとえば

BASIC／f64↵

とすれば，フリーエリアのサイズが64Kバイトで X-BASICが起動します。

複数のコンフィギュレーションファイルを使い分けるには？

cオプションは，複数のコンフィギュレーションファイルを使用する際に便利です。通常，コンフィギュレーションファイルのファイル名はBASIC．CNFに固定されています。しかし，／cオプションを使うことで別の名前のコンフィギュレーションファイルを使うことができます。ですから通常使う内容はBASIC．CNFに入れておき，夜だけ使用する一時的なコンフィギュレーションファイルとかは別の名前で（たとえばNIGHT．CNF）作成しておくとよいでしょう。

BASIC／cNIGHT．CNF↵

とすれば，NIGHT．CNFをコンフィギュレーションファイルとみなしてX-BASICを起動することができます。

その他の補足

起動時オプションが指定されますと，その指定がコンフィギュレーションファイルの内容よりも優先されます。また

BASIC／オプション／オプション…

のように複数のオプションを同時に使うこともできます。その際，オプションの順序は任意です。

Q 6 X-BASICのプログラムをオートスタートさせるには？

A 6

X-BASICで作成したプログラムを，X-BASICの起動時に

オートスタート……自動的に実行させる

ことができます。この機能を利用すると，たとえばX-BASICで作成した表計算ソフト等をオートスタートさせる（つまり表計算専用ディスクができあがる）ことができます。

オートスタート

プログラムを置くディレクトリに注意

X-BASICのプログラムをオートスタートさせるには，次の2つの方法があります。

●ファイル名を"AUTORUN．bas"にする

X-BASICのプログラムのファイル名を

AUTORUN．bas

にしておきますと，X-BASIC起動時にそのプログラムを自動的に実行してくれます。この時注意しなければならないことは，"AUTORUN．bas"を

BASIC.X…X-BASICの本体

と同じディレクトリに置かなければならないということです。さもないと無視されてしまいます。

ファンクションキーの設定等，X-BASICの起動時の環境を整えるためのプログラムを"AUTORUN．bas"に入れておくと便利です。

●起動時にファイル名を指定する

たとえば"TEMP．bas"というX-BASICのプログラムをオートスタートさせたいとしたら

```
BASIC  TEMP↵
         └──オートスタートさせるファイル名
              拡張子basは省略可能
```

のようにしてX-BASICを起動します。この時注意しなければならないことは，オートスタートさせるプログラムを**カレントディレクトリ**に置かなければならないということです。もしくは，**パス名**を付けて指定する必要

があります。さもないと無視されてしまいます。

3つの補足

3つほど，補足をしておきます。

1つは，オートスタートさせるファイル名と起動時オプションを同時に指定したい場合は，必ず**オプションの方を先**に指定しなければならないということです。ですからX-BASIC起動の正確な書式は

X-BASICの起動

BASIC ／オプション［ ／オプション…］　自動実行ファイル名

となります。

もう1つは，AUTORUN．basが存在し，かつオートスタートさせるべきファイル名も同時に指定された場合，どちらのファイルが優先されるかです。この場合は，AUTORUN．basではなく，**指定された方のファイル**のが優先され，自動実行されます。

そして，最後にオートスタートされたプログラムは，実行後に**自動的に消滅**します。実行後，そのプログラムを残すことはできません。

オートスタートの実例

それでは，オートスタートの実例を示しましょう。

まず最初に**第1-27図**のように2つのBASICのプログラムを作ります。それぞれは

AUTORUN．bas
　　"exec　AUTORUN．bas"と表示するプログラム
TEMP．bas
　　"exec　TEMP．bas"と表示するプログラム

のファイル名でセーブしておきます。なおこれらのファイルは，サブディレクトリbasicにセーブしています。そこにX-BASICの本体であるBASIC．Xがあるからです。次に

system ↵

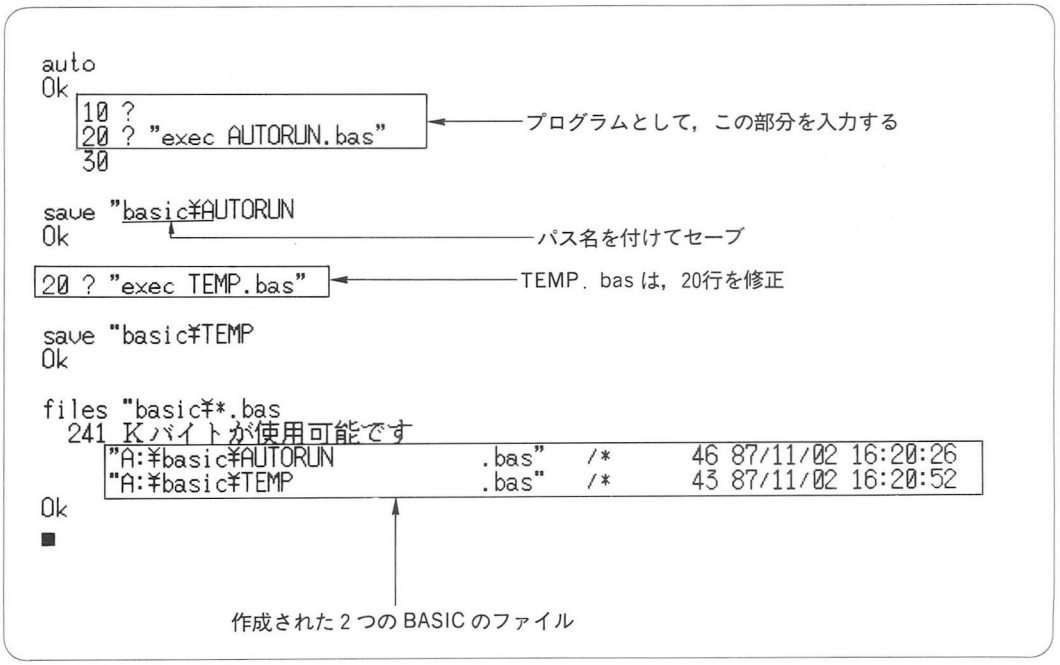
でコマンドシェルに戻ります。そして，

BASIC↵

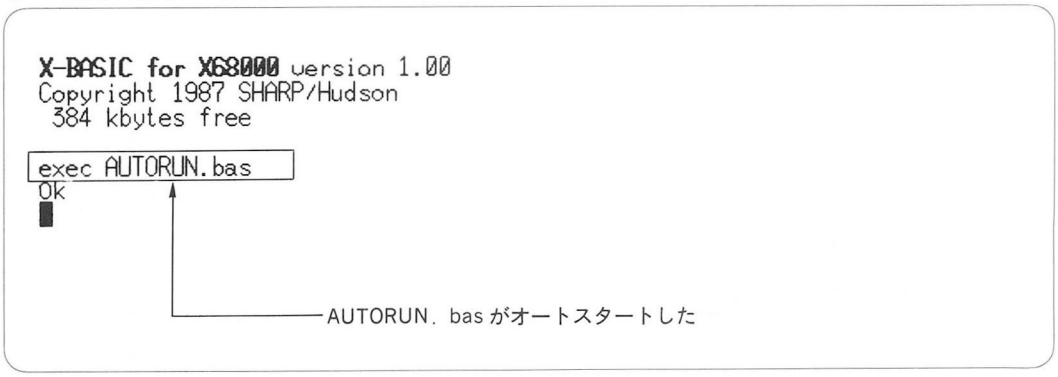
でX-BASICを起動します。すると，AUTORUN．basが自動的に実行されるのがわかります**【第1-28図】**。

再びコマンドシェルに戻り，今度は

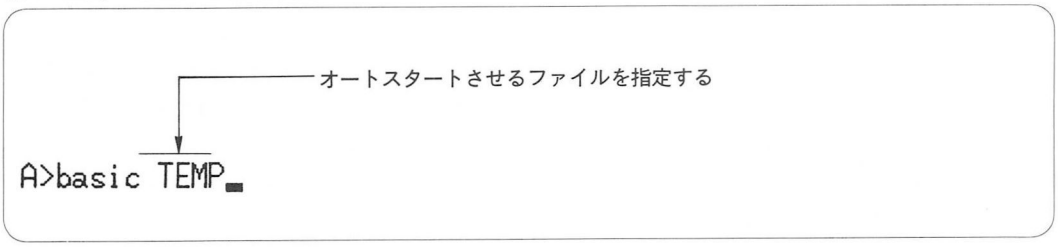
第 1 -27図　自動実行させるファイルを作る



第 1 -28図　AUTORUN.bas のオートスタート



第 1 -29図　TEMP.bas を指定

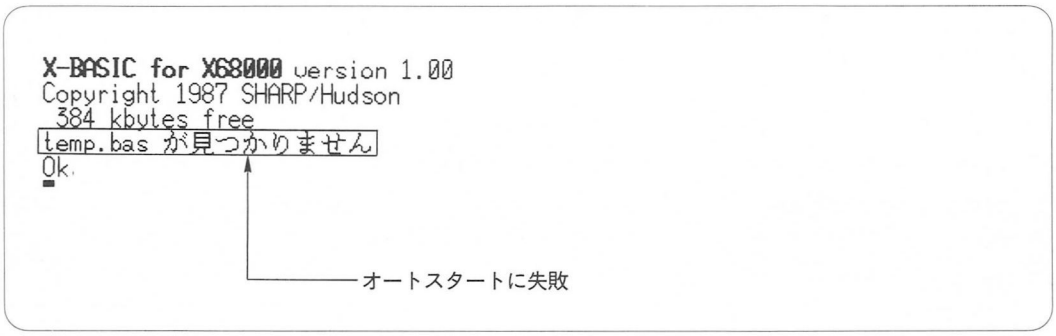


BASIC　TEMP↵

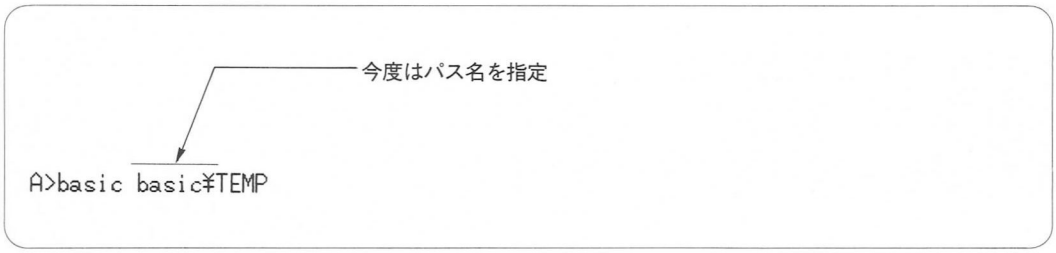
のように"TEMP．bas"を指定してX-BASICを起動してみます【第 1 -29図】。

しかし，残念ながらTEMP．basは**実行されていません**。これは，

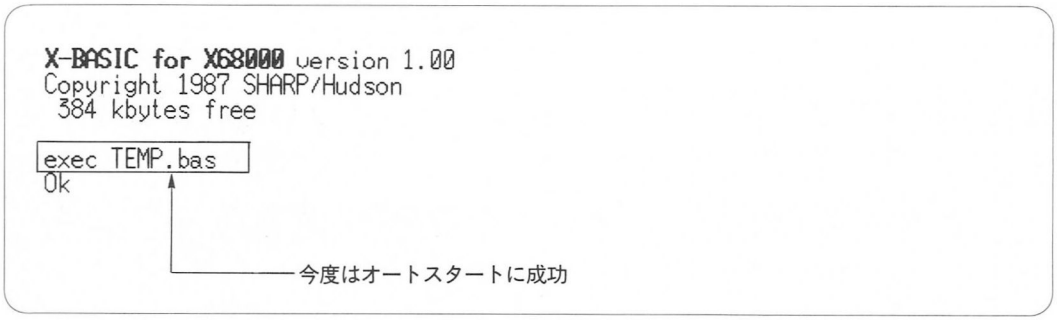
第1-30図　失　　敗！



第1-31図　パス名を付けて



第1-32図　成　　功！



TEMP．basがカレントディレクトリ（この場合は，ドライブ0のルートディレクトリ）ではなく，サブディレクトリbasicにあるからです**【第1-30図】**。

そこで，**第1-31図**のように**パス名**を付けてX-BASICを起動してみます。すると，今度はTEMP．basがオートスタートします**【第1-32図】**。

AUTORUN．basの方は無視されたことに注意してください。

第2章

OSとのインターフェース

Q7　X-BASICとプロセスの関係は？

Q8　エラーコードを返すには？

Q9　ディレクトリを見るには？

Q10　カレントディレクトリ／カレントドライブを変更するには？

Q11　ファイルを削除／ファイル名前を変更するには？

Q12　チャイルドプロセスを実行するには？

Q13　チャイルドプロセスとしてシェルを起動するには？

Q7 X-BASICとプロセスの関係は?

A7

第1章で，X-BASICを起動する際の注意点を詳細に見てきました。逆に，X-BASICからOS（コマンドシェルやビジュアルシェル）に戻る方法として

system

というコマンドがあります。また似た機能を持つステートメントに

exit（）

system
exit（）

があります。この両者は大変に似ていますが，微妙に異なるところがあります。本来，systemやexit（）は，**プロセス**についての命令です。そこで，

●プロセスから見たsystem，exit（）の機能
●両者の明確な相違

について，このQ7および次のQ8で具体的に説明しておくことにします。

プロセスとは何か?

X-68000のOSであるHuman68kは，本来は**シングルタスク**のOSですが，プログラムやメモリの管理の手法にマルチタスク的な面が見られます。すなわちプログラムの実行単位を

プロセス

プロセス

としてとらえ，1つのプロセスから別のプロセスを起動することができます。この時最初のプロセスを親プロセス，また新しく生成されたプロセスを子プロセスといいます**【第2-1図】**。

親プロセスと子プロセス

親プロセスが子プロセスを生成する時，

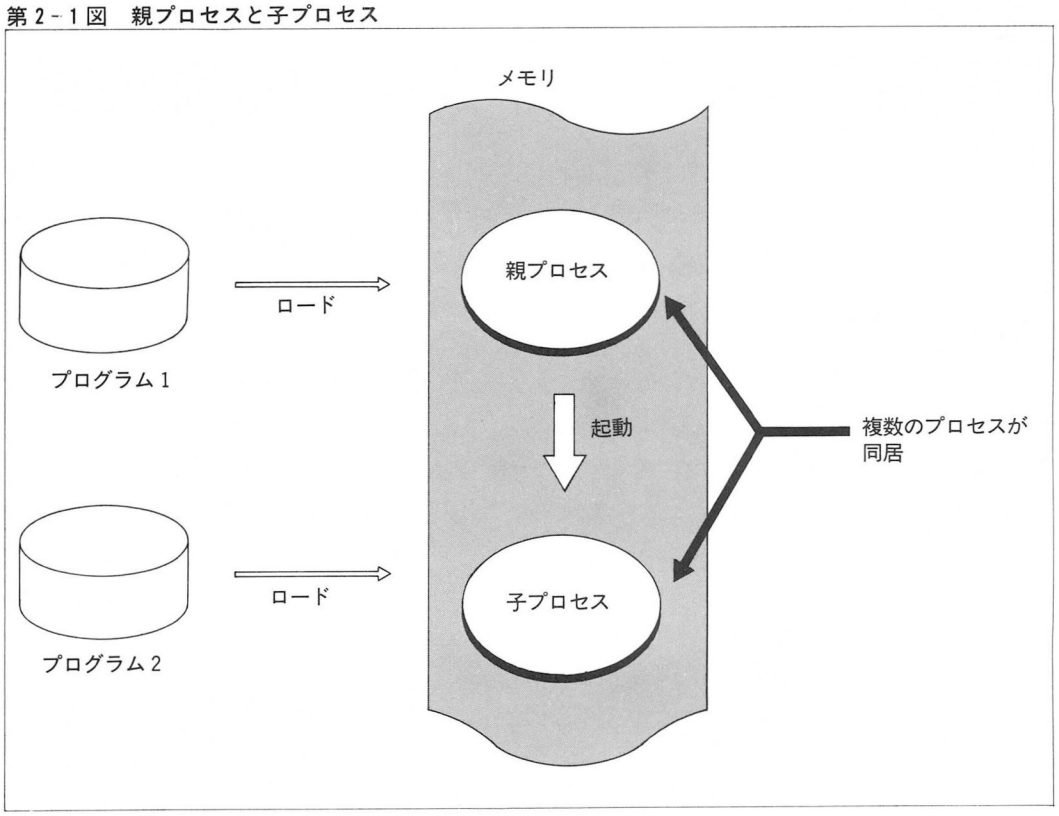
オーバーレイ

オーバーレイ

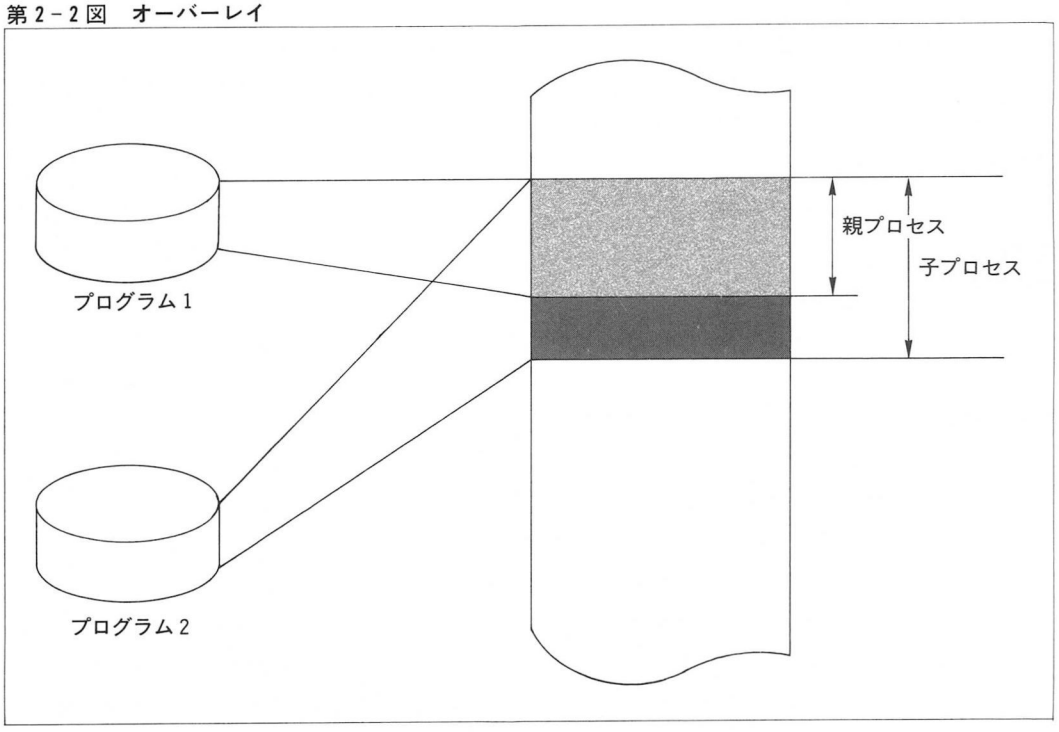
> 子プロセスを親プロセスと同じメモリ領域にロードする
> そのため子プロセスがロードされると同時に親プロセスは消滅する
> 親プロセスは子プロセスにチェインされる

することも（**第2-2図**参照），また子プロセスを親プロセスとは別の領域にロードすることもできます。ですから同時に複数のプロセスをメモリ上に置くことができるわけです。しかし，Human68kはマルチタスクのOS

第 2 - 1 図　親プロセスと子プロセス



第 2 - 2 図　オーバーレイ



ではありませんから，この複数のプロセスが同時に走るわけではありません。親プロセスは，子プロセスの実行が終了するまで

wait状態

wait——待っている

わけです。

子プロセスとしてのX-BASIC

プロセスという立場からX-BASICを見ますと，X-BASICは

親プロセス

この場合，通常は**コマンドシェル**か**ビジュアルシェル**
バッチファイルからX-BASICが起動することも
またある**プログラム**の中からX-BASICが起動することも可

から起動された**子プロセス**ということになります。すなわちコマンドシェルから

BASIC ↵

と入力してX-BASICを起動しているのは，子プロセスとしてX-BASICを生成しているのに過ぎません。親プロセスであるコマンドシェルは同時にメモリ上に残ってX-BASICが終了するのを待っています。そのX-BASICを終了させ，親プロセスに戻す行為が

system　または　exit（）

にほかなりません**【第2－3図】**。

systemとexit（）の機能

事実，X-BASICの中で

system（sys.と略しても良い）↵

または

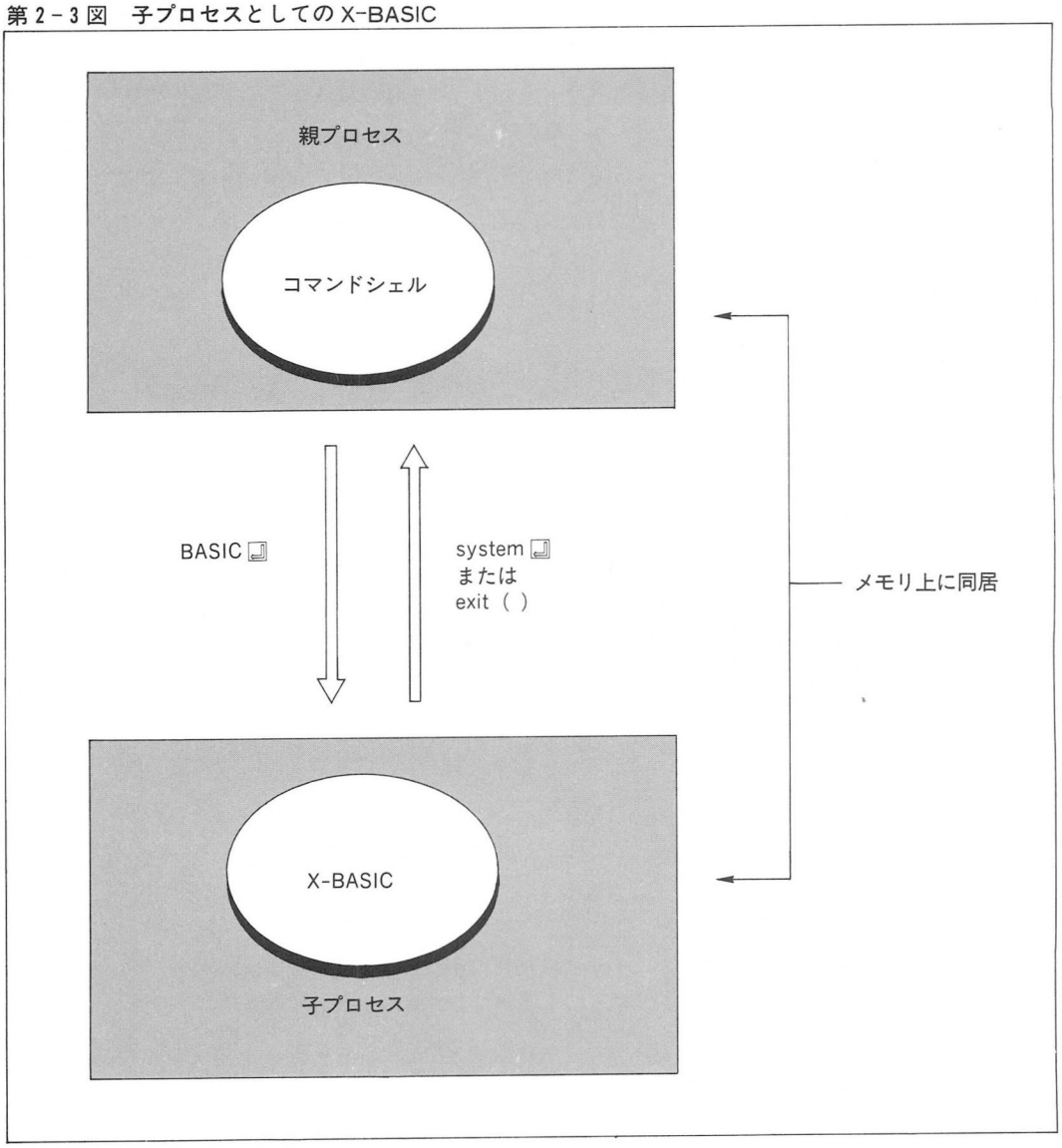
exit（）↵

と入力しますと，X-BASICを起動したコマンドシェルまたはビジュアルシェルに戻ることができます。それでは，このsystemとexit（）の違いは何でしょう？

命令としての属性が異なる

BASICの命令は，大きく

第 2－3 図　子プロセスとしての X-BASIC



命令の種類

①コマンド
主としてダイレクトモードで使用され，OS 的な仕事をする

②ステートメント
主としてプログラムの中で使用する命令
外部関数とは異なり，最初から X-BASIC に組み込まれている

③関数
ステートメントが内部的な命令であるのに対し，X-BASIC の外で作られた後天的な命令

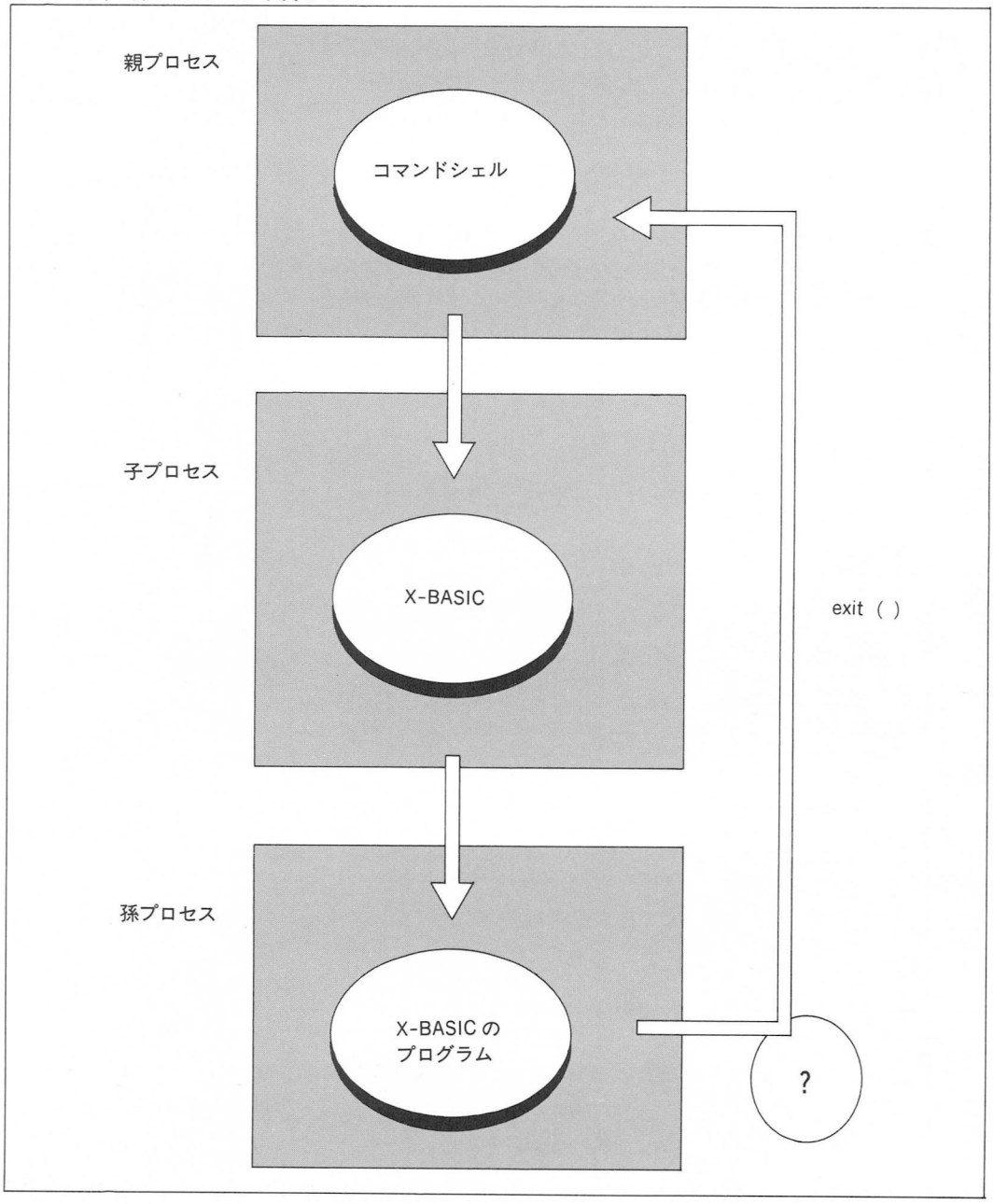
の3種類があります。この分け方でいきますと，

system——コマンド

exit（）——ステートメント

ということになり，これら2つの命令はその属性が異なります。事実，system はダイレクトモードでのみ使用可能で，プログラムの中で使うとエラーとなります。それに対し，exit（）はダイレクトモードでもプログラム

第2－4図　孫プロセスから戻る？



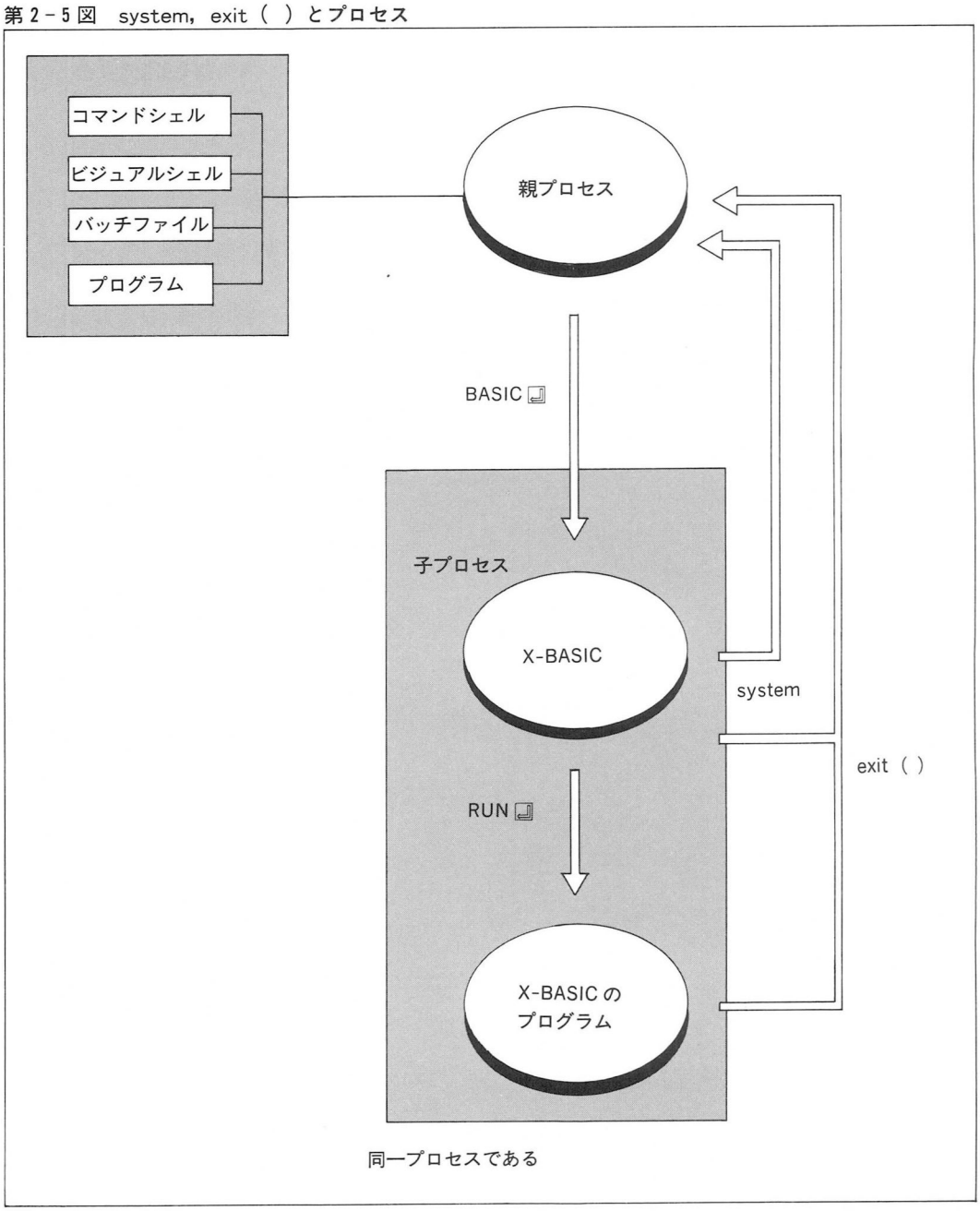
の中でも使用可能です。

孫プロセス？

すると，ちょっとおかしなことが起こります。

プログラムの中で exit（ ）を使用しますと，そこでいきなり X-BASIC

第 2－5 図　system，exit（ ）とプロセス



を起動した親プロセスに戻ってしまいます。X-BASICは，コマンドシェル等の親プロセスから起動されます。そして，X-BASICのプログラムはX-BASICから起動されます。つまりX-BASICのプログラムは，元の親プロセスから見れば孫プロセスのようなものです。ところがexit（）1つで，孫プロセスから一挙に親プロセスに戻ってしまうように見えます**【第2－4図】**。

これは誤りで，X-BASICのプログラムはX-BASIC実行中の1つの手続きにすぎません。つまりOSから見ればX-BASICもそこで走るプログラムも同一のプロセスに見えます。ですからX-BASICのプログラム中からexit（）を使用することで一挙にコマンドシェル等に戻ることができるのです**【第2－5図】**。

Q 8 エラーコードを返すには？

A 8

エラーコードとは何か？

Human68k は，親プロセスから子プロセスに戻る時に

エラーコード

エラーコード

を返すことができます。エラーコードというのは，もともとはエラーとは何の関係もない概念です。単なる整数ですが，この値によって親プロセスが終了した時の状態を子プロセスに知らせることができるのです。慣習的に

0 ――――正常終了
1 以上――エラー発生
　　その値により，エラーの種類を知らせる

の値を使うことが多いようです。このエラーコードは，たとえば

コンパイラ

等に使われています。コンパイラは慣習的に，エラーがなく正常にコンパイルできた時は，0を返してきます。もし重大なエラーが発生した場合は，1以上のエラーコードを返します。ですからたとえばバッチファイルの中でコンパイラを起動すれば

errorlevel

```
if errorlevel 1 goto エラー処理
```

により，エラーが発生した時エディタを起動する等のふるい分けを行うことができます。

〈補　注〉

Human68k では，エラーコードとして**2バイトの符号無し整数**を仮定しているようです。65536以上の数は，**65536の mod** が採用されます。ですから－1は65535として扱われ，65536はちょうど0として扱われます。

エラーコードを指定できる

このエラーコードまで考慮に入れますと，system と exit（）の2つの命令は明確な違いが出てきます。すなわち

エラーコードを返す命令

- system
  常にエラーコードとして0を返す
- exit（ ）
  返したいエラーコードを( )の中に記述することによりエラーコードを自由に設定できる
  （ ）の中を省略するとエラーコードとして0を返す

の違いがあるのです。

〈注意〉

exit（ ）の引数については，マニュアルに記述がありません。
おそらく私の調べたことで間違いないと思います。

エラーコードを調べる仕掛け

このことを具体例をあげて説明します。

ERR．bas

いま，ERR．basというX-BASICのプログラムを考えてみます**【第2-6図】**。

第2-6図　ERR.bas

```
10 /* ＯＳにエラーコードを返す
20 int e
30 input "エラーコード "; e
40 exit(e)
```

キーボードから適当なエラーコードを入力させる

そのエラーコードを親プロセスに返す

このプログラムは，input文で適当な数を変数cに入力させ

```
exit (c)
```

のようにそれをエラーコードとして，親プロセスに返そうというわけです。

tst_exit.bat

このエラーコードの返却を受ける親プロセスとしては，tst_exit.batというバッチファイルを使うことにします **【第2-7図】**。

このバッチファイルでは，2行目で

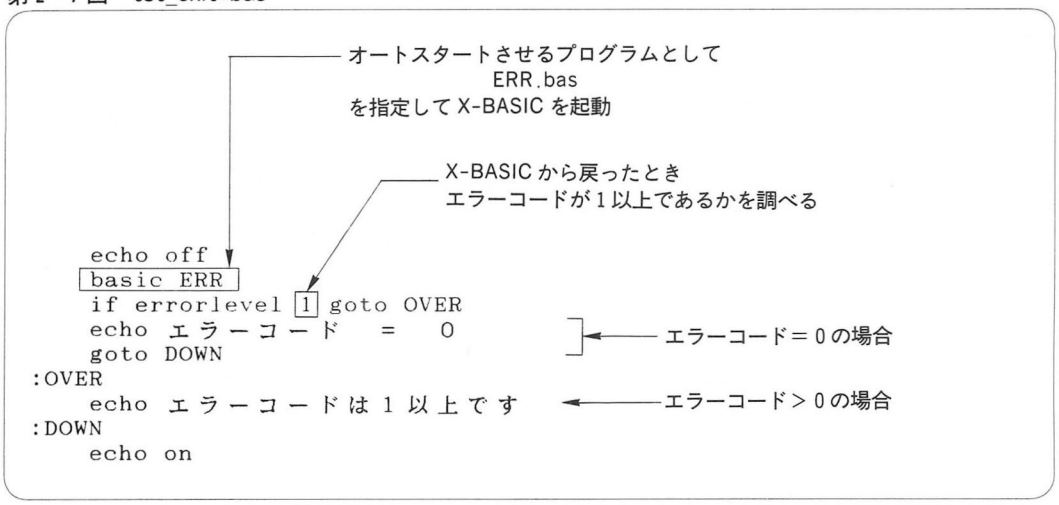
```
basic ERR
```

のようにX-BASICを起動しながらERR．basをオートスタートさせています。ですからX-BASICが起動すると同時に**第2-6図**のERR．basが走り，その中のexit（ ）命令によりこのバッチファイルに戻ってきます。そして，その時返されたエラーコードを

```
if errorlevel 1 goto ～
```

により，１以上かどうかをチェックしています。

第２－７図　tst_exit.bas



エラーコードを調べる

それでは，このバッチファイルをコマンドシェルから起動してみます。

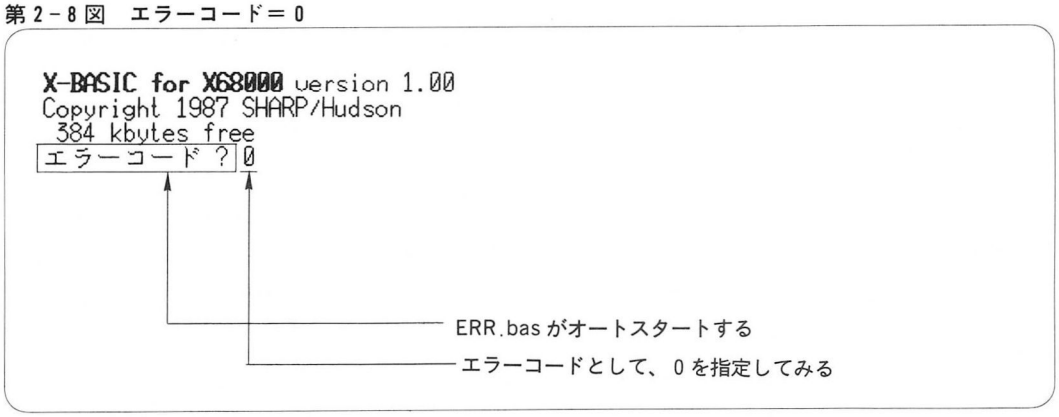
tst_exit ↵

によりバッチファイルをが起動しますと，ERR．basがオートスタートします。エラーコードを入力するように促して来ますので，最初は

0

を返してみます【第２－８図】。

第２－８図　エラーコード＝０

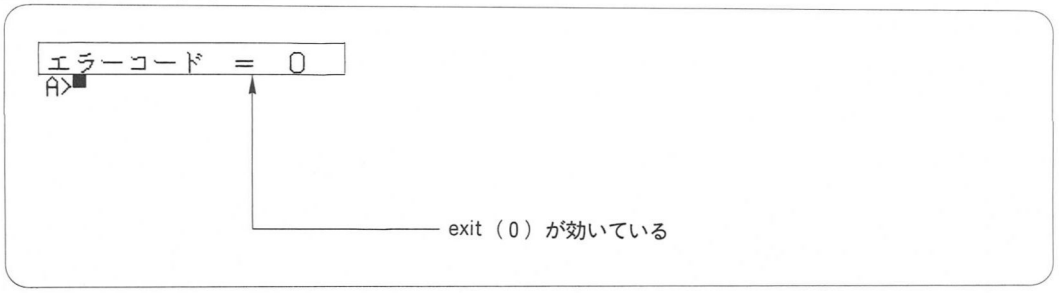


すると，バッチファイルはちゃんとその０を受け取り，その旨を知らせて来ます。つまり

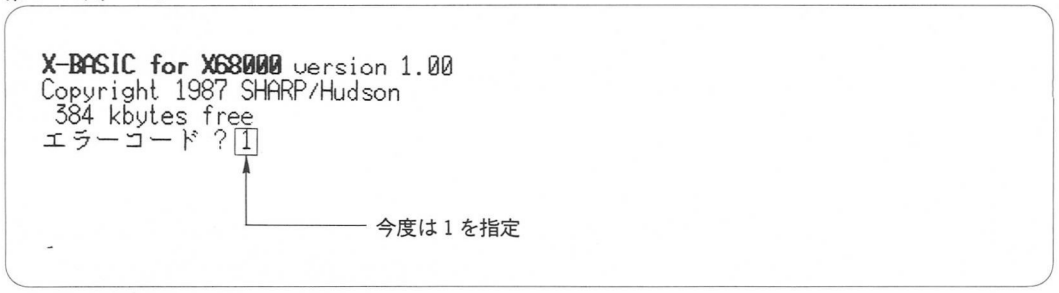
exit（０）

が効いています【第２－９図】。

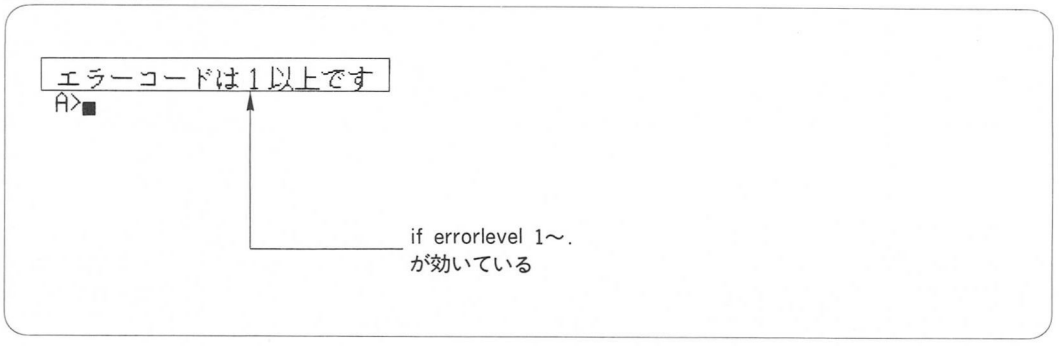
第2-9図　exit（0）で戻った場合



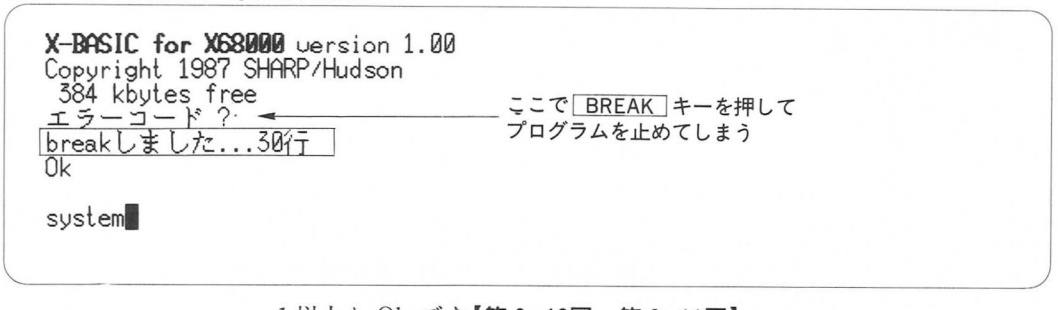
第2-10図　エラーコード＝1



第2-11図　exit（1）で戻った場合



第2-12図　強制的にsystemで戻す



1以上もOkです【第2-10図，第2-11図】。

またERR．basが走りだしたところで，〈BREAK〉キーを押し，

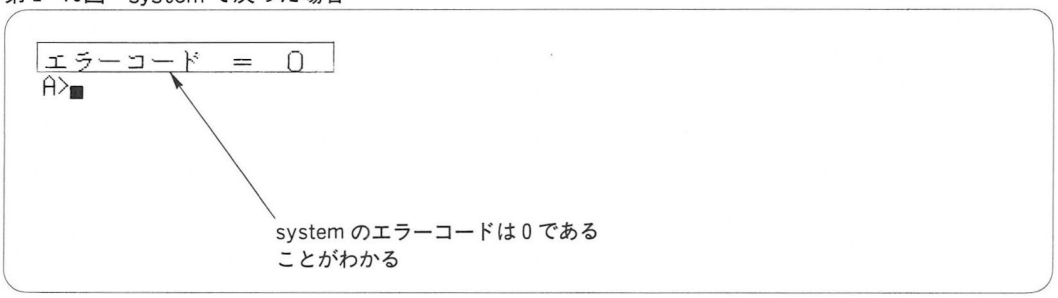
system ↵

でバッチファイルに戻ってみます【第2-12図】。

すると，バッチファイルとしてはエラーコード0を受け取ります。つま

り system の返すエラーコードは，0 なのです【第 2 -13図】。

第 2 -13図　system で戻った場合



systemとexit（）の相違

以上の実験と，前Q 7 により

system と exit（ ）の相違

が明らかになったと思います。両者は一見よく似た機能を持ちますが，**プロセス**という概念からとらえますと明確な機能上の違いが存在します。

両者の違いを**第 2 -14図**にまとめておきます。

第 2 -14図　system と exit（ ）

	命令の種類	ダイレクトモードでの使用	プログラムの中での使用	画面クリア	エラーコード
system	コマンド	○	×	する	0
exit( )	ステートメント	○	○	する	( )の中で指定　省略は0

Q9 ディレクトリを見るには?

A9 ディレクトリとは?

Human68kでは,dirコマンドで**ディレクトリ**を見ることができます。ただし，こういうディレクトリは，正確な意味でのディレクトリでないことに注意してください。Human68kは，ファイルやサブディレクトリを管理するのに

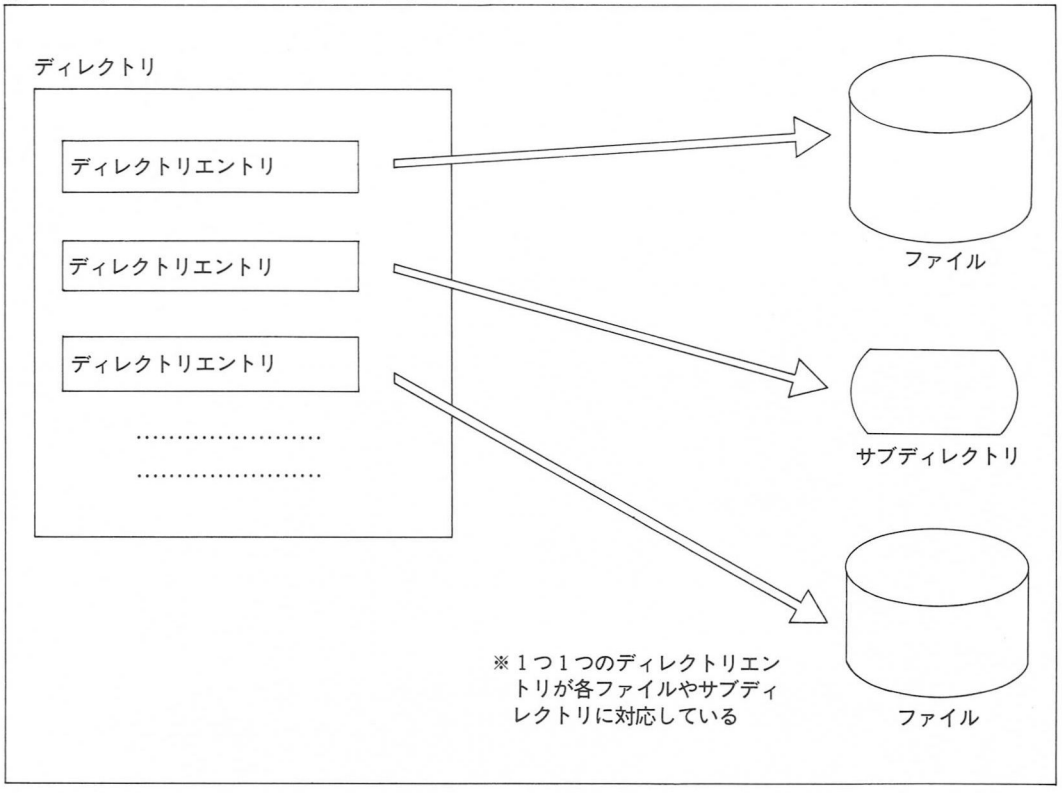
ディレクトリ　　**ディレクトリ**

というテーブルを持っています。ディレクトリは，

ディレクトリエントリ　　**ディレクトリエントリ**

の集合です。その1つ1つのディレクトリエントリが，各ファイルやサブディレクトリに対応しています【第2-15図】。

第2-15図　ディレクトリとディレクトリエントリ



dir コマンドは，そのディレクトリに納められているディレクトリエントリのうち，ユーザーが必要とする情報(たとえばファイル名や日付等)をピックアップし，読みやすい形で表示するものです（ファイルの物理的位置等必要のないものは表示しません)。ですから普通に「ディレクトリを見る」といった場合，「ファイルの一覧を見る」といった意味に使っているわけです。

X-BASICではfilesコマンドを使う

さて，X-BASICが起動してしまいますと，もちろんdirコマンドは使えません。代わりに

files
fi.

　　files
　　(fi. のように省略して入力することができる)

というコマンドを使ってファイルの一覧を見ることができます。このfilesというコマンドは過去のBASICにもあったのですが，X-BASICのそれは，ほとんどdirコマンドと同等に使えるように拡張されています。

filesコマンドのいろいろ

以下，X-BASICにおけるfilesの使い方を説明しておきます。

●カレントディレクトリを見る

まず

　　files ↵（またはfi. ↵）

でカレントディレクトリのディレクトリが見られます【第2-16図】。

第2-16図　カレントディレクトリを表示

```
          ←―― files の省略形
fi.
  240 Kバイトが使用可能です
    "A:¥CONFIG          .SYS"   /*     208 87/10/20 09:14:16
    "A:¥AUTOEXEC        .BAT"   /*      40 87/11/02 10:44:56
    "A:¥X68K_M          .DIC"   /*  625664 87/03/15 12:00:00
    "A:¥X68K_S          .DIC"   /*   14336 87/03/15 12:00:00
    "A:¥SYS             .   "   /* --DIR-- 87/10/05 10:55:42
    "A:¥BIN             .   "   /* --DIR-- 87/10/05 10:55:46
    "A:¥BASIC           .   "   /* --DIR-- 87/10/05 10:55:50
    "A:¥temp            .bas"   /*      96 87/11/04 13:45:12
    "A:¥tst_exit        .bat"   /*     168 87/11/04 16:08:40
    "A:¥err             .bas"   /*      96 87/11/04 16:06:12
Ok
```

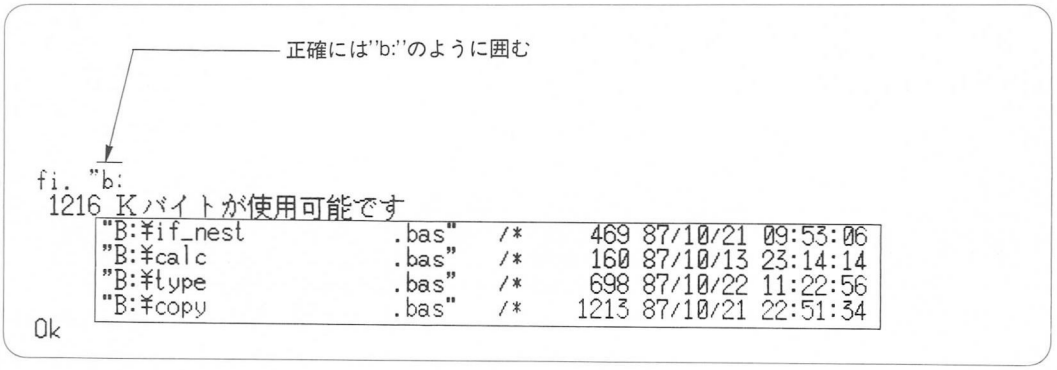
●ドライブBのディレクトリを見る

もし　ドライブBのディレクトリを見るなら

files "b:"↵

のようにします【第2-17図】。

第2-17図　ドライブBのディレクトリを表示



昔のBASICですと「files 2」のようにしましたが，アルファベットのドライブ名を使うことに注意してください。また，文字列として" "の中に入れる必要があります。なおX-BASICはマイクロソフトのBASICのように**文字列が文の最後に使われた時，最後の" を省略**することができます。ですから短く記述するなら

fi.　"b:↵（最後の" は省略できる）

のように入力することができます。

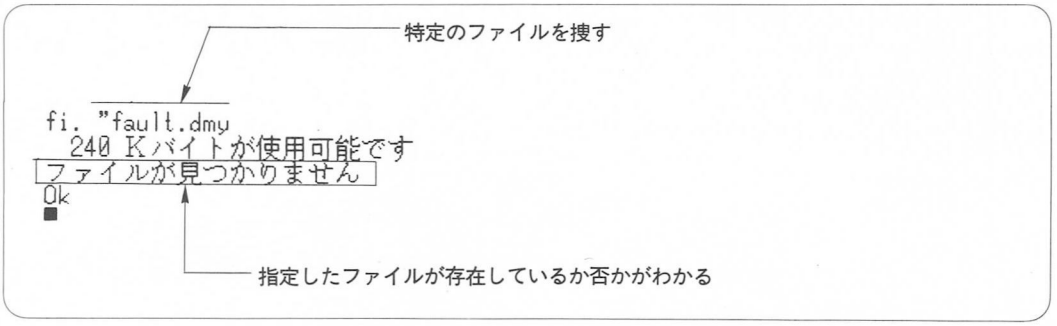
●ファイルを捜す

filesコマンドは，その対象として

files "fault.dmy"↵

のように**特定のファイル名**を指定することができます。この場合は，そのファイルがディレクトリの中にあるかを捜すのに使用できます【第2-18図】。

第2-18図　ファイルを捜す



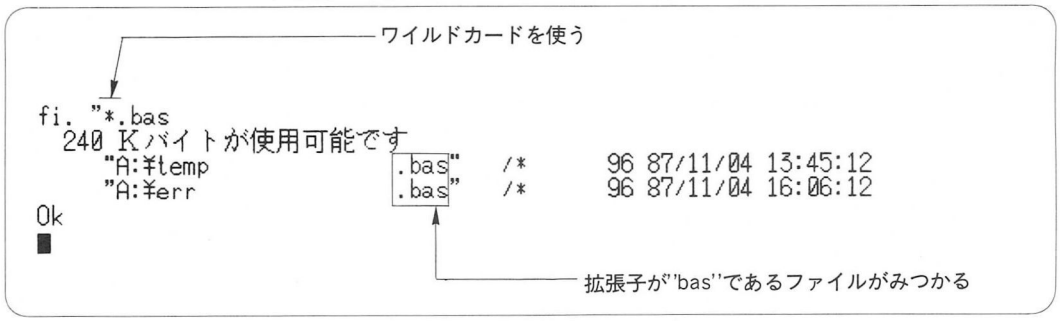
●ワイルドカードの利用

さらにそのファイル名に，**ワイルドカード**を使用することができます。たとえば

files　"＊．bas"↵

としますと，カレントディレクトリ中の拡張子．basのファイルを捜すのに使用できます【第2-19図】。

第2-19図　ワイルドカードを使ってサーチ



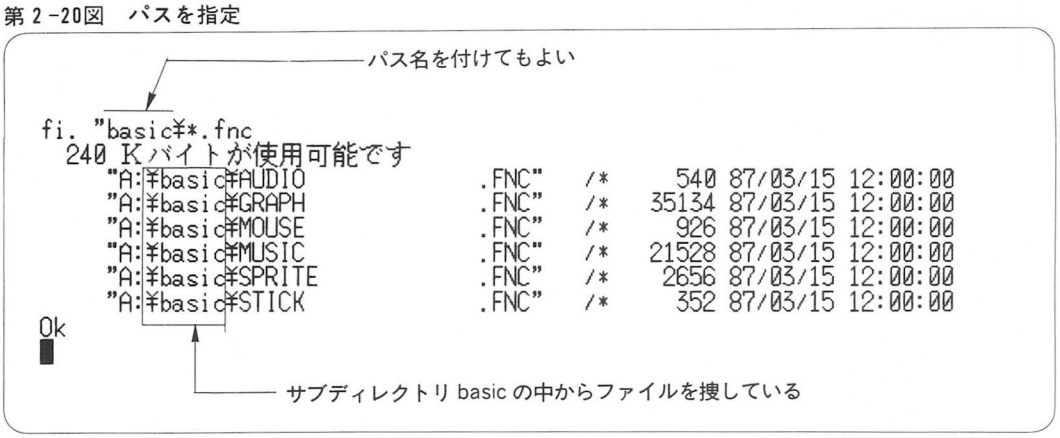
●他のディレクトリを見る

filesコマンドは**パス名**を付けることができますので，特定のサブディレクトリの様子を調べることができます。たとえば

files　"basic¥＊.fnc"↵

としますと，サブディレクトリbasicの中のコンフィギュレーションファイルを見ることができます【第2-20図】。

第2-20図　パスを指定



〈補注〉

filesの代わりにlfilesを使いますと，結果をプリンタに出力することができます。

Q10 カレントディレクトリ／カレントドライブを変更するには?

A10

マニュアルには載っていませんが，X-BASICの中でもカレントディレクトリ／カレントドライブを変更することができます。

カレントディレクトリの変更——chdir

chコマンド（カレントディレクトリの変更）に相当するX-BASICのコマンドがchdirコマンドです。

chdir

● chdir

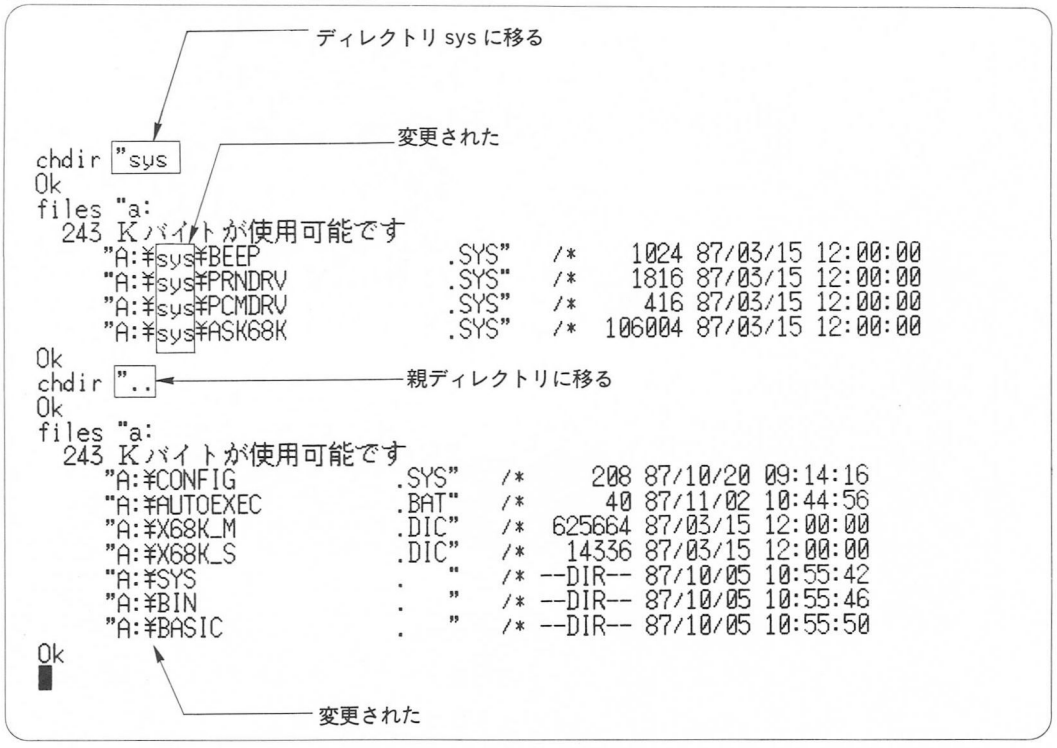
［属　性］　コマンド

［書　式］　chdir　”ディレクトリ名”

［機　能］　カレントディレクトリを変更する

たとえばカレントディレクトリをsysに変更したい場合は

第2-21図　chidirコマンドの使用例



chdir　"sys"↵

のように入力します。ディレクトリ名には

¥――ルートディレクトリ

..――親ディレクトリ

も使用可能です【第 2 -21図】。

カレントドライブの変更――chdrv

カレントドライブを変更するには，chdrv コマンドを使用します。

chdrv

chdrv

［属　性］　コマンド

［書　式］　chdrv　"ドライブ名"

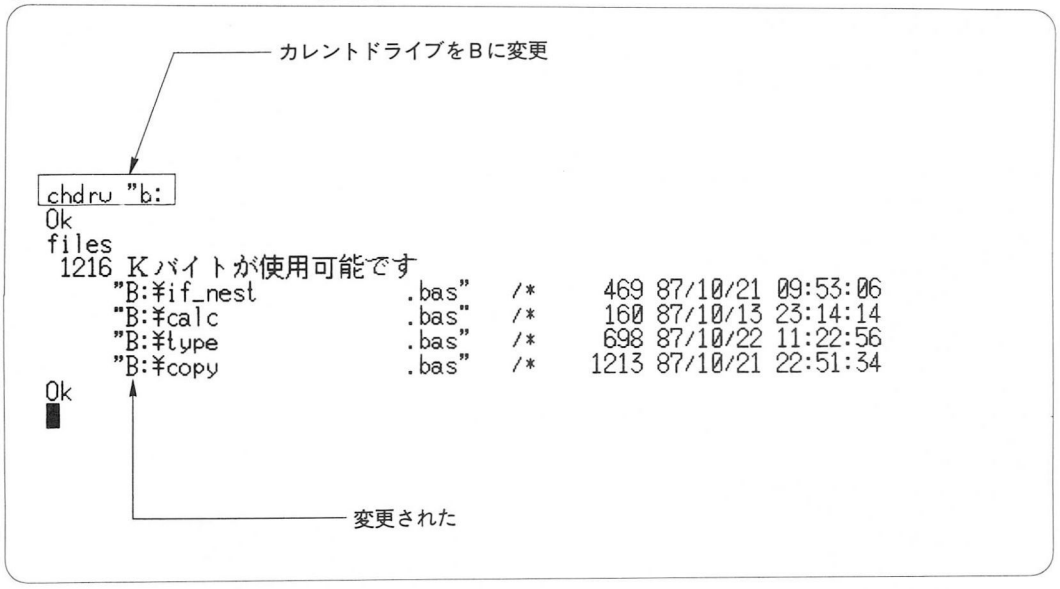
［機　能］　カレントドライブを変更する

たとえばカレントドライブをＢに変更するには，

chdrv　"b:"↵

のようにします【第 2 -22図】。

第 2 -22図　chdrv コマンドの使用例



Q11 ファイルを削除／ファイル名を変更するには？

A11

これもマニュアルには載っていませんが，X-BASICの中でファイルを削除したり，ファイル名を変更することができます。それぞれについて

コマンド

関数

の両方が用意されていますので，その違いについても説明しておきます。

ファイルの削除(コマンド)——kill

まずは，ファイルを削除するkillコマンドからいきます。

kill

● kill

［属　性］　コマンド

［書　式］　kill　"ファイル名"

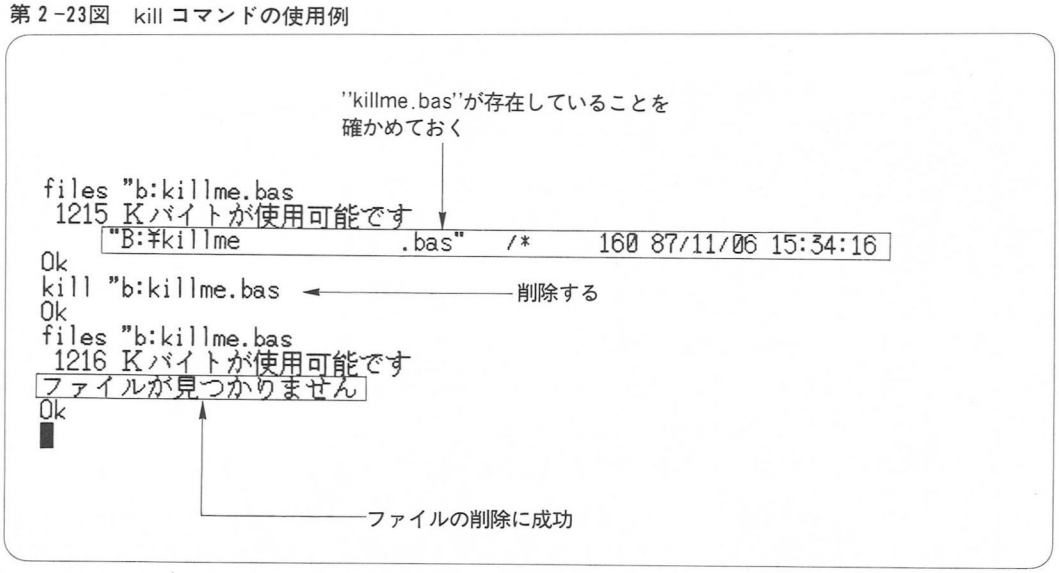
［機　能］　ファイルを削除する

たとえばkillme．basというファイルを削除したい場合は

kill　"killme．bas"↵

のようにします【第2-23図】。

第2-23図　killコマンドの使用例



〈補　注〉

X-BASICの killコマンドは，Human68kの delコマンドとは異なり，ワイルドカードを使用することはできません。

killはプログラムの中では使えない

X-BASICには，もう1つファイルを削除する命令としてfdelete( )が用意されています。つまりファイルを削除するための命令が2つ存在するわけです。この両者の違いは，

kill　　――――コマンド

fdelete ( )　――関数

のようになっています。このようにkillはコマンドですので，プログラムの中で使用することはできません【第2-24図】。

第2-24図　killはプログラムの中では使えない

```
run
コマンドは実行できません...10行
   10 kill "b:delme.bas"
      ^
Ok
```

ファイルの削除(関数)――fdelete ( )

プログラムの中でファイルを削除する命令を使用したいなら，killではなく，関数であるfdelete ( ) の方を使用します。

fdelete

● fdelete ( )

［属　性］　関数

［書　式］　fdelete ("ファイル名")

［戻り値］　int　　0――正常終了

－1―エラー

［機　能］　ファイルを削除する

たとえば次のfdelete．basは，fdelete( )をプログラムの中で使用した例です【第2-25図】。

いますぐにこのプログラムを理解する必要はありませんが，70行のところでfdelete( )が使用されていることに注意してください。このプログラムは，X-BASICのファイルを削除するもので，ファイル名の入力に拡張子．basを省略することができます【第2-26図】。

第2-25図　fdelete.bas

```
 10 /* fdelete() のテスト
 20 error off
 30 str fn
 40 while 1
 50      input "どのファイルを削除しますか ('e'でストップ)"; fn
 60      if fn = "e" then break
 70      if fdelete(fn + ".bas") = -1 then {
 80          beep
 90          print fn; ".bas は存在しません"
100      } else {
110          print fn; ".bas を削除しました"
120      }
130 endwhile
```

ここでファイルを削除している
fdelete（）は、エラーが起こると－1を返す

第2-26図　fdelete.bas の実行

```
run
どのファイルを削除しますか ('e'でストップ)? tst
tst.bas は存在しません
どのファイルを削除しますか ('e'でストップ)? delme
delme.bas を削除しました
どのファイルを削除しますか ('e'でストップ)? e
Ok
```

存在しないファイルを指定

今度は存在するファイルを指定
拡張子"bas"が付加される

これで stop

〈補　注〉

kill コマンドと同じように，fdelete(　)もワイルドカードは使えません。

ファイル名の変更(コマンド)——name

ファイル名の変更にも，コマンドと関数の2つが用意されています。

name　————コマンド

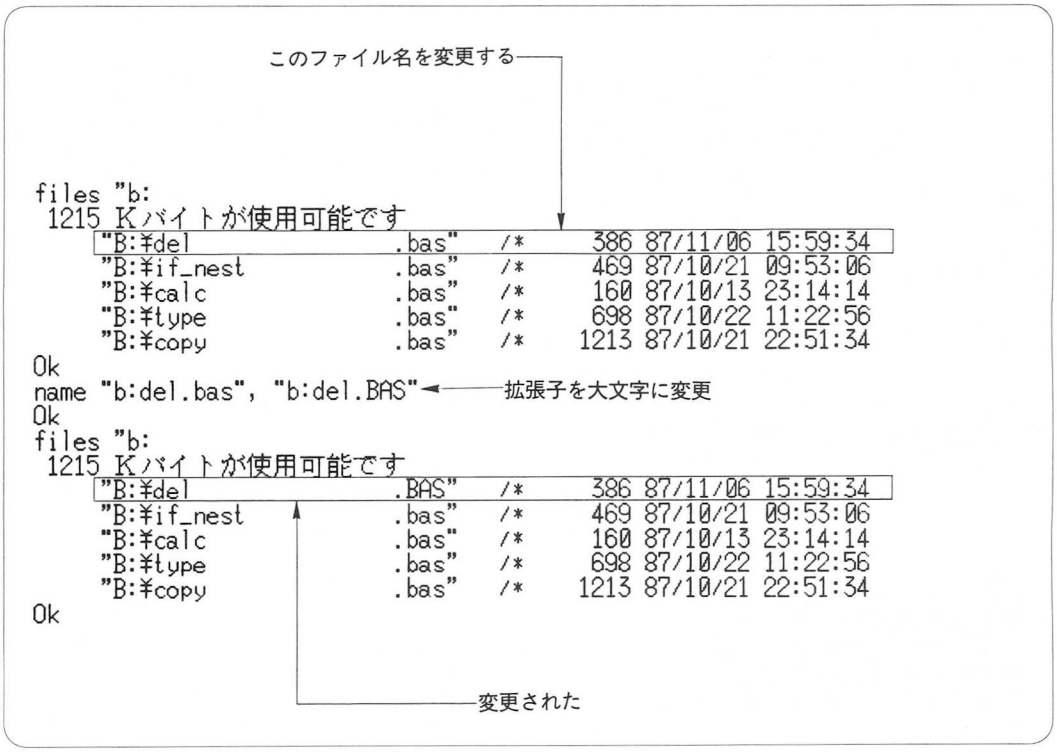
frename（）　——関数

name を用いて，del．bas のファイル名を del．BAS に変更するには

name "del．bas","del．BAS"↵

のように入力します【第2-27図】。

第 2 -27図　name コマンドの使用例



name

● name

［属　性］　コマンド

［書　式］　name（"旧ファイル名", "新ファイル名"）

［機　能］　ファイル名を変更する

ファイル名の変更(関数)——frename ( )

ただし，name はコマンドですからプログラムの中では使用できません。プログラムの中でファイル名を変更したいなら関数である frename ( ) を使用します【第 2 -28図】。

frename

● frename ( )

［属　性］　関数

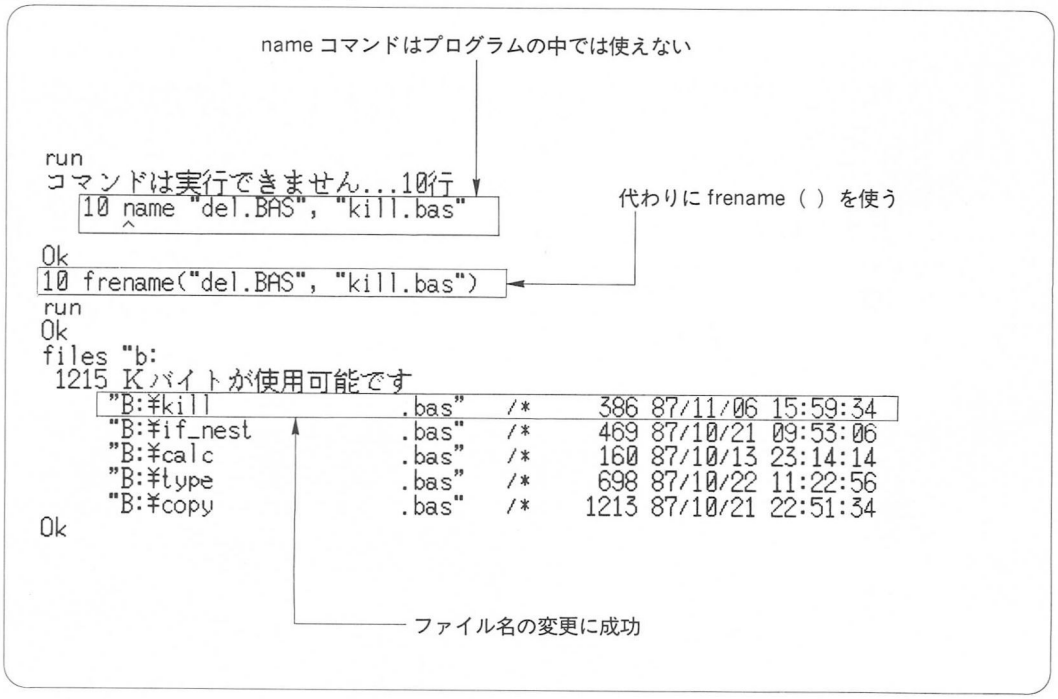
［書　式］　frename（"旧ファイル名", "新ファイル名"）

［戻り値］　int　　0 ——正常終了

－1…エラー

［機　能］　ファイル名を変更する

第2-28図　nameとfrename（　）



Q12 チャイルドプロセスを実行するには？

A12

すべてのOS的なコマンドを使用するには？

OSのコマンドを実行する

ファイル名を変更したり等のOS的なコマンドを紹介して来ましたが，他にも実行したいOSのコマンドはたくさんあるでしょう。しかし，X-BASICで使えるOS的なコマンドは，Q9～Q11で紹介したものですべてです。これ以外のコマンドを使用したい場合は，直接X-BASICの中でHuman68kのコマンドを実行するしかありません。つまり

チャイルドプロセス（子プロセス）として

Human68kのコマンドを実行

するのです。その方法をこのQで説明しておきます。

child

チャイルドプロセスの実行

X-BASICの中でHuman68kのコマンドを実行するには

child コマンド

を使用します。ただし，マニュアル上の名前はchildですが，実際は

！記号

を使用します。つまり！に続けて，Human68kのコマンド名を入力するのです。たとえばtypeコマンドで"hello．bat"というファイルの内容を読みたい場合は

```
！type hello．bat ↵
```

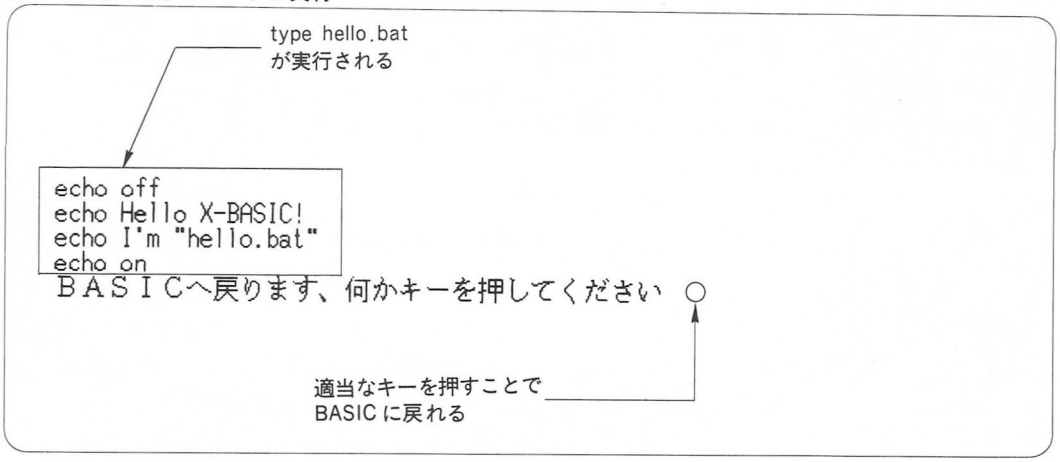
のようにします。これを入力しますと，画面がクリアされコマンドシェルの本体であるcommand.xがロードされます(その間，少し待たされます)。そして，！以下に書かれた部分がコマンドシェルへのコマンドとして解釈され，実行されます。

typeコマンドをチャイルドプロセスとして実行した例を**第2-29図**に示します。

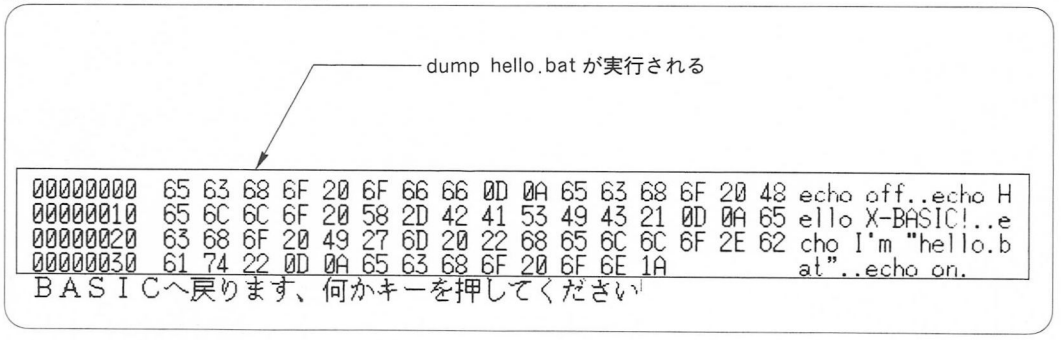
外部コマンド／バッチファイルも実行可能

マニュアルによりますとチャイルドコマンドで実行できるのは，Human68kの内部コマンドだけということになっています。しかし，実際

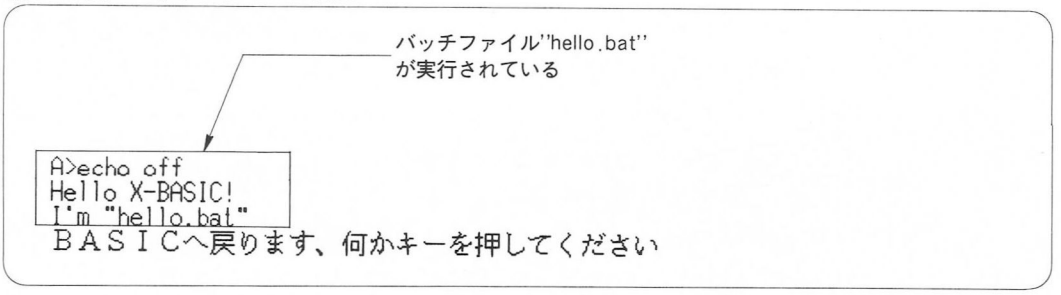
第2-29図　内部コマンドの実行



第2-30図　外部コマンドの実行



第2-31図　バッチファイルの実行



すべてのコマンドを　は
実行

外部コマンド――拡張子．x のコマンドファイル

バッチファイル――拡張子．bat のコマンドファイル

のいずれも X-BASIC の中から実行することができます。例として

第2-30図――外部コマンド dump で"hello．bat"をダンプ

第2-31図――バッチファイル hello．bat を実行

したところを示しておきます。

Q13 チャイルドプロセスとしてシェルを起動するには?

A13

シェルを起動するメリット

前Q11で，X-BASICの中からHuman68kのコマンドを使用する方法を説明しました。しかし，この方法では**1回に1つ**のコマンドしか使用できません。Human68kのコマンドをいくつか連続的に実行したい場合，この方法でもできないことはありませんが，能率が良くありません(childプロセスは！を1回実行する度にcommand.xをロードする)。

このような場合は，X-BASICの中から

シェルを起動

チャイルドプロセスとしてシェルを起動

してしまうのが便利です。そうすれば何回でもOSのコマンドを使用することができますし，コマンドシェル，ビジュアルシェルのうち好きな方を使用することもできるからです。

シェルの起動の仕方

X-BASICの中からシェルを起動する——その方法は，いわれてみれば何てことはないことです。つまりchildコマンドで外部コマンドであるシェルを実行してしまえばよいのです。

起動法

●コマンドシェルを起動する場合

！command

●ビジュアルシェルを起動する場合

！vs

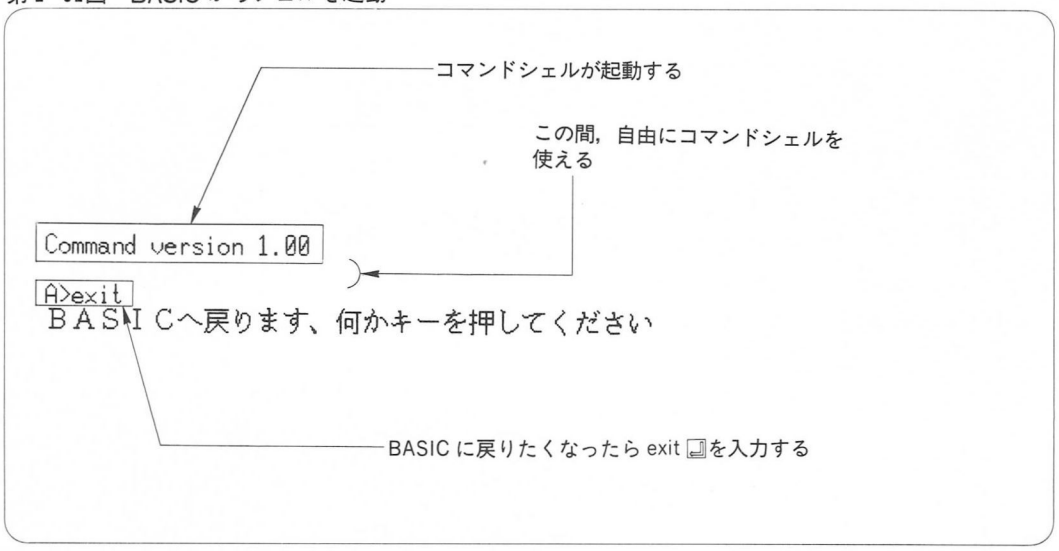
［ただし，vs．xをシステムディスクに転送しておく必要がある］

第2-32図は，X-BASICの中からチャイルドプロセスとして**コマンドシェル**を起動したところです。X-BASICに戻りたい場合は

exit ↵

を入力します。

第2-32図　BASICからシェルを起動



第3章

スクリーンエディタ

Q14　スクリーンエディタを利用するには？

Q15　テキストをセーブ／ロードするには？

Q16　行番号をカットしてテキストを表示・プリントするには？

Q17　文字列を検索するには？

Q18　まとまった行をまとめて編集するには？

Q19　1つの行を2行に分割したり，2つの行を1行に統合するには？

Q20　コントロールコードとは？

Q21　コントロールコードを利用した高度な編集を行うには？

Q14 スクリーンエディタを利用するには?

A14

このQ以下では，**初めてBASICに接する人**を対象にスクリーンエディタの利用の仕方を説明していきます。すでにBASICに馴染みのある方は無視してください。

X-BASICの基本——スクリーンエディタ

X-BASICの中で

- X-BASICのプログラムを入力する
- 普通の文章を作成する

といった作業を行うには，**スクリーンエディタ**を利用します。スクリーンエディタで入力できるのは，プログラムばかりとは限りません。普通の文章を入力することもできます。つまりスクリーンエディタを簡易ワープロのように使用することも可能です。ちなみにスクリーンエディタの入力の対象となるものを

テキスト

テキスト

といいます。

このようにスクリーンエディタの使い方を知らなければ，X-BASICを利用することはほとんど不可能です。そこで，まずこのQで手っ取り早くスクリーンエディタを利用する方法をご伝授しましょう。

テキストエリアの掃除——New

new

スクリーンエディタ利用の第1歩は，newというコマンドを使用することです。スクリーンエディタでは，プログラムや文章等のテキストをメモリ上の

テキストエリア

という領域に格納します。newは，このテキストエリアを空っぽにする命令です。もしかすると，前に入力したテキストが残っているかもしれません。ですからスクリーンエディタで新しいテキストを入力する前には，必ずnewを実行するとよいでしょう。入力は，簡単です。

new ↵

とすればよいのです。これで，テキストエリアは，空っぽになり，新しい

テキストの入力を開始できます【第3－1図】。

第3－1図　スクリーンエディタの初期化

```
new
Ok
■
```

テキストの入力

たとえば，「X-68000で使用できるソフトのデータベース」をスクリーンエディタで作ってみましょう。

まず1つ目のテキストとして，MUSIC PROのデータを入力してみます。スクリーンエディタは，行番号オリエンティッドなエディタ（行番号を基準に構成されているエディタ）ですので，先頭に適当な

行番号

1～65535のいずれか
通常は10より始めて10おきに行番号を付けていく

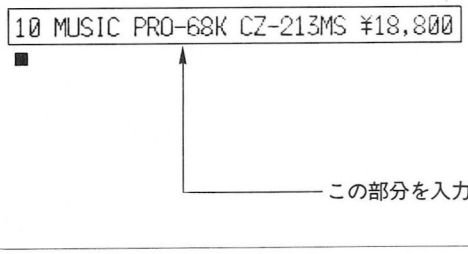
を付け，各行を

入力する時の1行の構成

〈行番号〉〈本文〉↵

の構成で入力していきます【第3－2図】。

第3－2図　1行入力



入力したテキストを表示する——list

入力したテキストは，スクリーンエディタのテキストエリアに格納されます。入力したテキストを確認したい場合は，

list

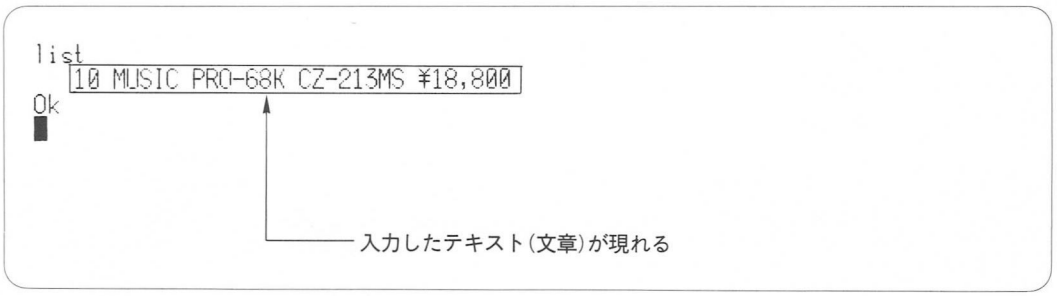
list コマンド

を使用します【第3－3図】。

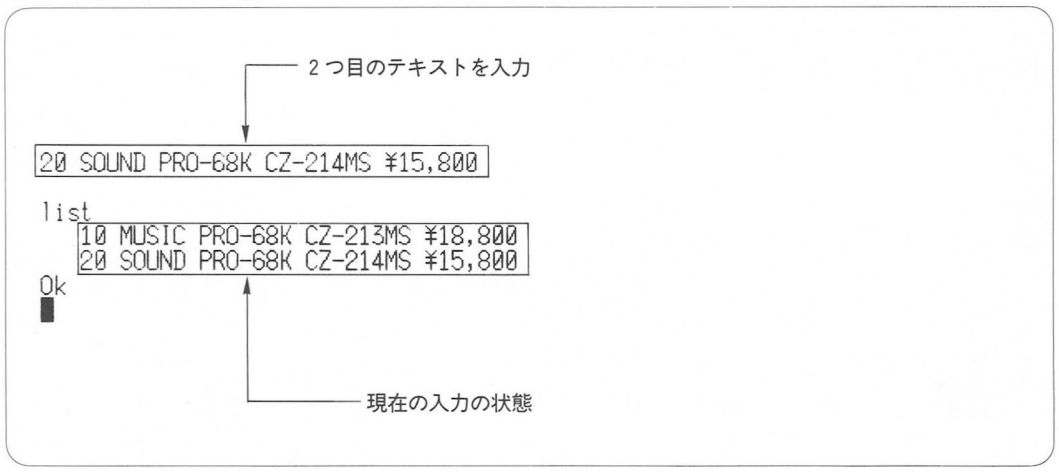
さらに2番目のテキストを入力してlistを取りますと，第3－4図のよう

になります。

第3－3図　リストを取る



第3－4図　第2行の入力



途中に挿入

いままでは，行番号を10から始めて10おきに付けていきました。もしこれ以外の行番号を付けたらどうなるでしょう？

第3－5図は，5，15という行番号を付けて入力したものです。ご覧のようにスクリーンエディタは，入力されたテキストを行番号順に並び換えてテキストエリアに格納しています。つまり今まで入力したテキストの途中に

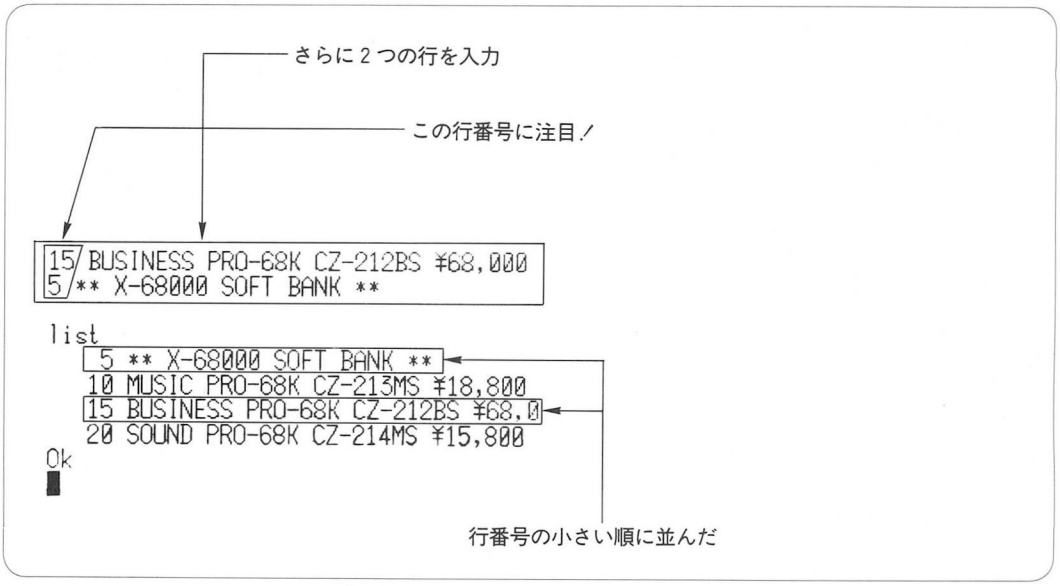
新しい行を挿入

したい場合は，挿入したい行番号と行番号の間の行番号を使用すればよいのです。

リストをプリンタに出力——llist

入力したテキストは，list コマンドで見ることができました。list の代わりに llist コマンドを使用しますと，プリンタに出力することができます。

第3－5図　昇順に整列



第3－6図　プリンタに出力されたリスト

```
 5 ** X-68000 SOFT BANK **
10 MUSIC PRO-68K CZ-213MS ¥18,800
15 BUSINESS PRO-68K CZ-212BS ¥68,0
20 SOUND PRO-68K CZ-214MS ¥15,800
```

list　　すなわち

llist ↵

のように入力します。――これで，**第3－6図**のような出力を得ることができます。

以上，ざっとスクリーンエディタの最も簡単な使い方を見てきました。

入力したテキストは行番号順に整列される

これがX-BASICにおけるスクリーンエディタの基本中の基本です。

Q15 テキストをセーブ/ロードするには?

A15

スクリーンエディタで入力したテキストは，テキストエリア——すなわちメモリ上の領域に格納されるだけですからそのままでは，**電源オフ**で消えてしまいます。ディスクにセーブして初めて保管することができるわけです。

テキストのセーブ——save

テキストをディスクにセーブするには，save **コマンド**を使用し

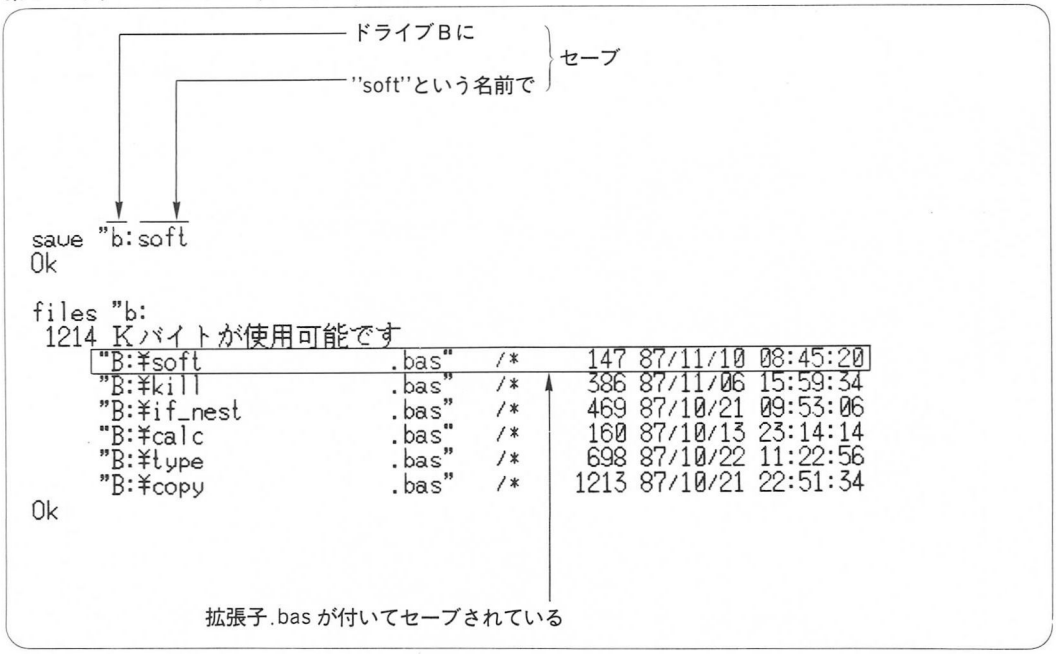
save　　save"**ファイル名**"↵

のようにします。**第3－7図**は，Q14で入力したテキストを soft というファイル名でドライブBのディスクにセーブしたところです。save コマンドを使用しますと，ファイル名拡張子として

.bas

が自動的に付加されます。

第3－7図　プログラム（テキスト）のセーブ



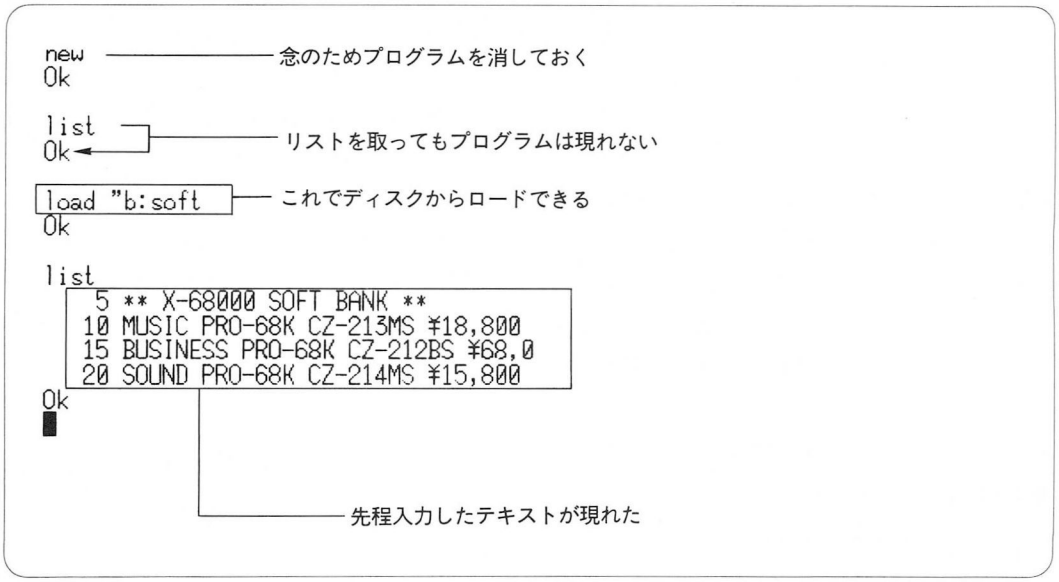
テキストのロード——load

逆にディスクにセーブされたテキストをロードするには，load **コマンド**を使用し

load　　load "**ファイル名**"↵

のようにします。**第3－8図**の使用例で確かめてください。

第3－8図　プログラム（テキスト）のロード



Q16 行番号をカットしてテキストを表示・プリントするには？

A16

なぜ行番号をカットするか？

Q14，Q15で見てきましたように，スクリーンエディタはBASICのプログラムだけでなく，

通常の文章

も扱うことができます。これは，使えそうです。たとえば先に示したソフトのデータベースや住所録程度なら今すぐにでも利用できそうです。

しかし，できあがったリストをllistコマンドでプリンタに出力しますと，**行番号**も一緒に出力されてしまいます。住所録等では，行番号は必要ありません。スクリーンエディタで作成したテキストを行番号なしで出力する方法はないものでしょうか？

行番号をカットしてCRTに表示する

そこで，スクリーンエディタで作られたファイルを行番号なしで出力するための簡単なプログラムを作ってみました。

行番号をカットするプログラム

CRTに表示するもの――cutcrt.bas

プリンタに出力するもの――cutpr.bas

の2種類あります。

第3－9図が，CRT用のプログラムcutcrt.basのリストです。短いものですからすぐに入力できるでしょう。Q14でやった方式で入力し，Q15の方式でディスクにセーブしておいてください。X-BASICのプログラムを実行するには

run ↵

と入力します。すると表示したいファイルの名前を聞いてきますので，希望ファイル名を入力してください。**第3－10図**のように行番号なしで表示してくれます。

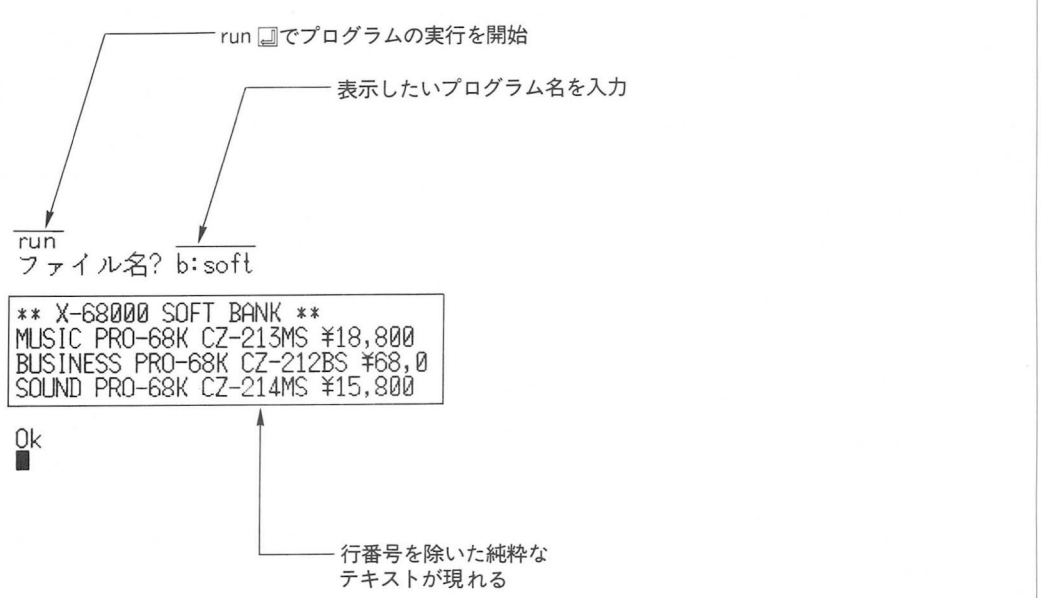
行番号をカットしてプリンタに表示する

そのプリンタ版が，**第3－11図**のcutpr.basです。こちらのプログラムの方が役に立つでしょう。

第3－9図　cutcrt.bas

```
 10 /*-------------------------------------------
 20 /* basic のプログラムをＣＲＴに出力する
 30 /*      行番号はカットする
 40 /*      ファイル名の入力で .bas は省略する
 50 /*                                  87.11.10
 60 /*-------------------------------------------
 70 error off
 80 str fn, buf[255]
 90 input "ファイル名"; fn
100 fp = fopen(fn + ".bas", "r")
110 if fp = -1 then {
120     print "ファイルがオープンできません"
130     end
140 }
150 print
160 while not feof(fp)
170     fseek(fp, 6, 1)                  /* 行番号を読み飛ばす
180     freads(buf, fp)
190     print buf
200 endwhile
210 if fclose(fp) then {
220     print "ファイルがクローズできません"
230     end
240 }
```

第3－10図　cutcrt.bas の実行



実行の仕方は，cutcrt.bas とほとんど同じです。**第3－12図**のような出力が得られます。これだけでも簡単なデータベースとして使えそうでしょう。短いプログラムですので，利用してください。

第3-11図　cutpr.bas

```
 10 /*-------------------------------------------
 20 /* basic のプログラムをプリンタに出力する
 30 /*      行番号はカットする
 40 /*      ファイル名の入力で .bas は省略する
 50 /*                                  87.11.10
 60 /*-------------------------------------------
 70 error off
 80 str fn, buf[255]
 90 input "ファイル名"; fn
100 fp = fopen(fn + ".bas", "r")
110 if fp = -1 then {
120     print "ファイルがオープンできません"
130     end
140 }
150 print
160 while not feof(fp)
170     fseek(fp, 6, 1)                 /* 行番号を読み飛ばす
180     freads(buf, fp)
190     lprint buf
200 endwhile
210 if fclose(fp) then {
220     print "ファイルがクローズできません"
230     end
240 }
```

cutcrt.bas と本質的に異なるのは
この行だけ(コメント行を除く)

第3-12図　行番号をカットしてプリンタに出力されたリスト

```
** X-68000 SOFT BANK **
MUSIC PRO-68K CZ-213MS ¥18,800
BUSINESS PRO-68K CZ-212BS ¥68,0
SOUND PRO-68K CZ-214MS ¥15,800
```

Q17 文字列を検索するには？

A17

長いプログラムを作っていますと，だんだんとどこに何があったかわからなくなってきます。また必要な箇所を捜すにも時間がかかるようになってきます。こんな時，

文字列を検索

できると便利です。たとえば**サブルーチン**や**関数定義の場所**をすばやく捜すのに利用できます。

文字列検索の意義

文字列の検索を行う——search

もともとX-68000のスクリーンエディタには，文字列検索の機能が備わっています。しかしマニュアルには書かれていませんので，このQで説明しておきます。

X-68000のスクリーンエディタで，文字列を検索するには search というコマンドを使用して

search コマンド

```
search　"捜したい文字列"↵
```

のようにします。たとえばいま**第3-13図**のようなプログラムがあったとします。この中から《print》という文字列を検索するには

```
search　"print"↵
```

とします。なお search は，

省略型

```
se.
```

のような**省略型**が認められています。またX-68000の文字列は，閉じる方の" を省略できますので

```
se.　"print ↵
```

のように入力しても Okです。

検索結果は，**第3-14図**のとおりです。print を含むすべての行が表示されます。この時，1print のように print を含む文字列も検索してくれます。

第3-13図　サンプルプログラム

```
 10 /*-------------------------------------------
 20 /* basic のプログラムをプリンタに出力する
 30 /*      ファイル名の入力で .bas は省略する
 40 /*                                  87.10.21
 50 /*-------------------------------------------
 60 error off
 70 str fn, buf[255]
 80 input "ファイル名"; fn
 90 fp = fopen(fn + ".bas", "r")
100 if fp = -1 then {
110     print "ファイルがオープンできません"
120     end
130 }
140 print
150 while not feof(fp)
160     freads(buf, fp)
170     lprint buf
180 endwhile
190 if fclose(fp) then {
200     print "ファイルがクローズできません"
210     end
220 }
```

このプログラムの中から''print''という文字列を含む行を
捜し出してみる

第3-14図　search コマンドの実行

正式には''search''

正式には ''で閉じる
X-BASIC はマイクロソフト系 BASIC のように
文字列の最後の ''を省略できる

```
se. "print
  110     print "ファイルがオープンできません"
  140 print
  170     lprint buf
  200     print "ファイルがクローズできません"
Ok
```

1つの単語の途中に含まれる''print''も
捜している

Q18 まとまった行をまとめて編集するには？

A18

ブロックを対象とした編集

X-68000のスクリーンエディタでプログラムを編集する時の基本は

編集の基本

> listで編集したい行を表示する
> ↓
> 修正したい位置にカーソルを移動する
> ↓
> 文字を消すなら
> 〈BS〉キー ——カーソルの左の文字を消去する
> 〈DEL〉キー——カーソルの位置している文字を消去する
> 文字を挿入するなら〈INS〉キーを押してから入力する
> ↓
> 最後に↵を押す

です。しかし，だんだんとスクリーンエディタに慣れてきますと

ある行から行の間のプログラム
——これを**ブロック**と呼ぶことにします

をまとめて編集する（たとえば削除する）といった**ブロックを対象にした編集機能**がほしくなってきます。

X-68000のスクリーンエディタに備わっている「ブロックを対象としたコマンド」をこのQで説明しておきましょう。

行番号の自動発生——auto

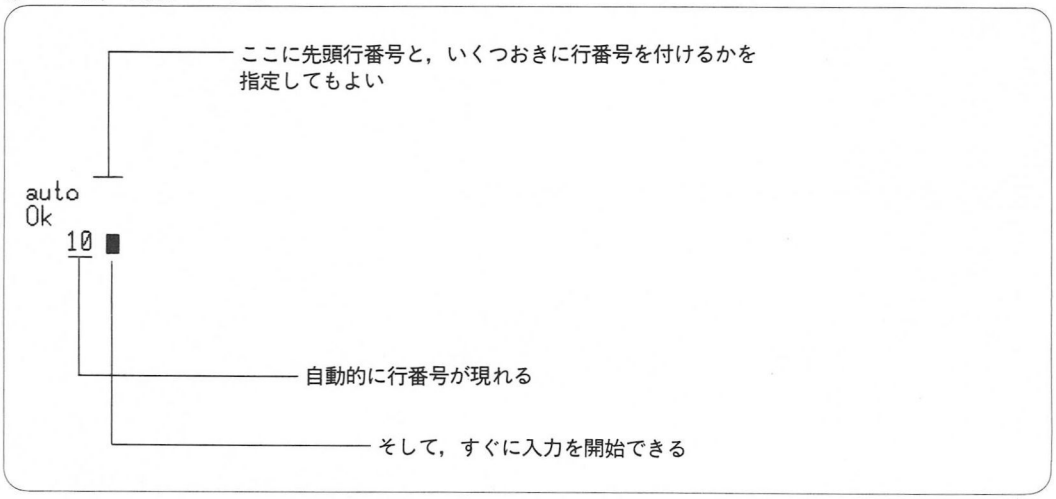
autoコマンドは，新しい行をまとめて入力するときに便利です。自動的に行番号を発生してくれるからです。使い方の例をいくつか示しましょう。

auto

> auto↵
> ——行番号10から10おきに行番号を発生します【第3-15図】
> auto　200↵
> ——行番号200から10おきに行番号を発生します

auto　500,20↵
——行番号500から20おきに行番号を発生します

第3-15図　auto コマンド



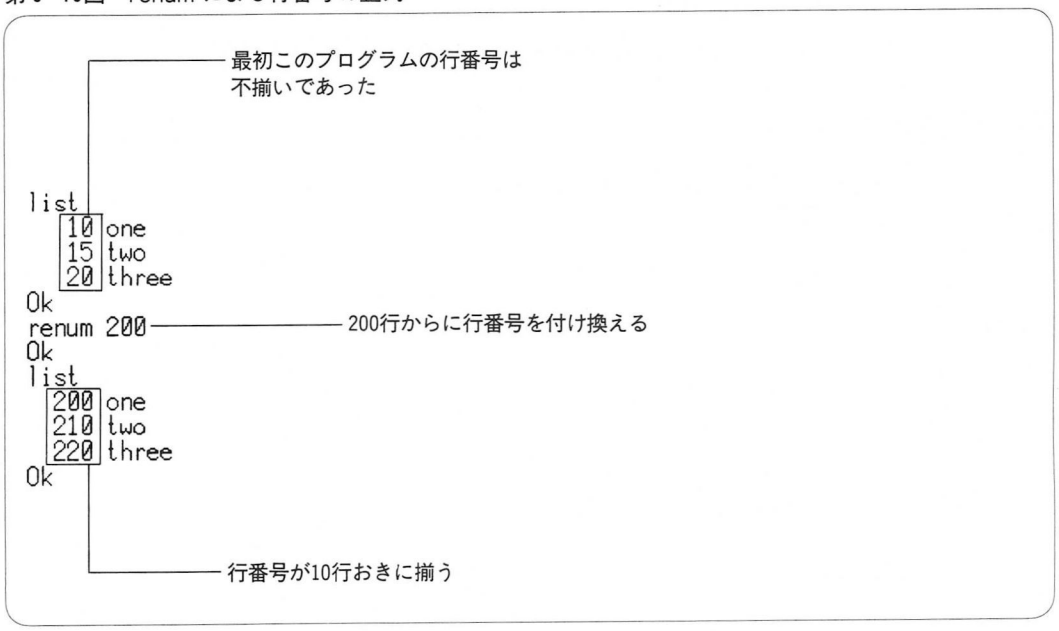
行番号の整列——renum

renum コマンドの用途は，主に
1>完成したプログラムの行番号をきれいに整列する
2>プログラムを挿入するのに行番号と行番号の間が詰まって余裕がなくなったといった場合に使用します
使い方の例をいくつか示しましょう。

renum

renum↵
——行番号を10から10おきに付け換えます
renum　200↵
——行番号を200から10おきに付け換えます【第3-16図】
renum　200,500↵
——500行以下の行番号を200から10おきに付け換えます
renum　200,500,30↵
——500行以下の行番号を200から30おきに付け換えます

第3-16図　renumによる行番号の整列

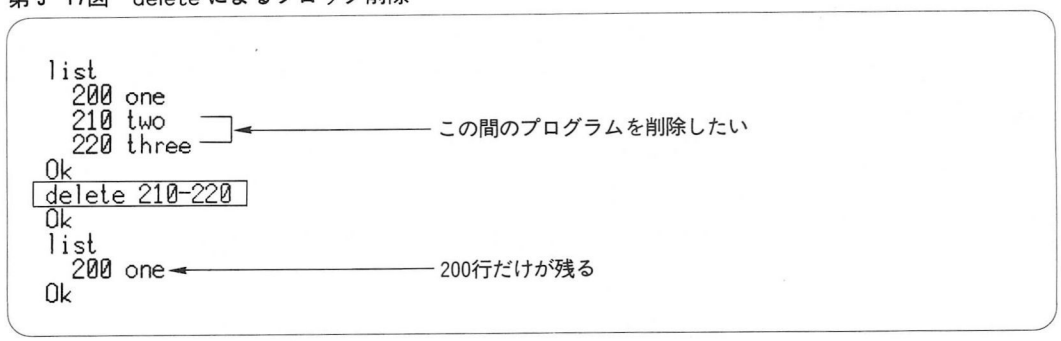


ブロックの削除——delete

delete

delete コマンドは，ある行から行までのまとまったブロックを削除するのに使用します。たとえば**第3-17図**は210行～220行の間にあるプログラムを削除したところです。

第3-17図　deleteによるブロック削除



Q19 1つの行を2行に分割したり2つの行を1行に統合するには?

A19

これもマニュアルには書かれていませんが，X-68000のスクリーンエディタは

1つの行を2行に分割する

2つの行を1行に統合する

といった編集を行うことができます。その方法を説明します。

分割と統合

1つの行を2行に分割するには?

第3-18図を例に1つの行を2行に分割する方法を説明します。それには，カーソルを分割したい位置に移動させます。そして，

コントロールJ

コントロールJ

——〈CTRL〉キーと〈J〉を同時に押す

を押します。すると，カーソル以後の部分が次の行の先頭に移動します。そこで，〈INS〉キーを押して挿入モードにします。そして，行番号を入力して↵を押します。これで，後半の行ができあがります。

なおこのままでは，統合前の行がソックリ残っていますので，カーソルを前半の行に持って行って↵を押してください。

2つの行を1行に統合するには?

第3-19図を参考に2つの行を1つに統合する方法を説明します。それには，カーソルを前の行の1番最後に持って行きます。そして，

コントロールW

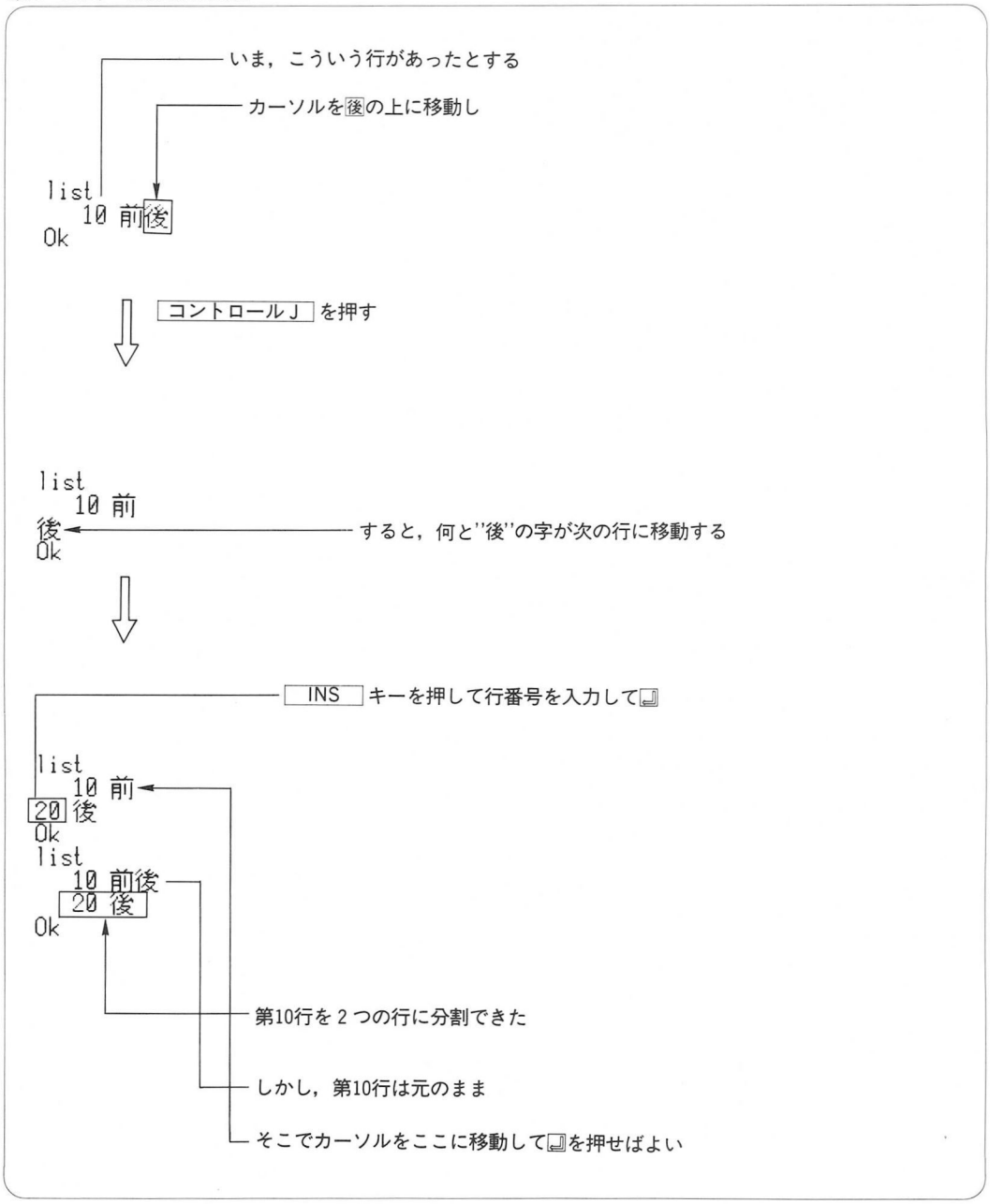
コントロールW

——〈CTRL〉キーと〈W〉を同時に押す

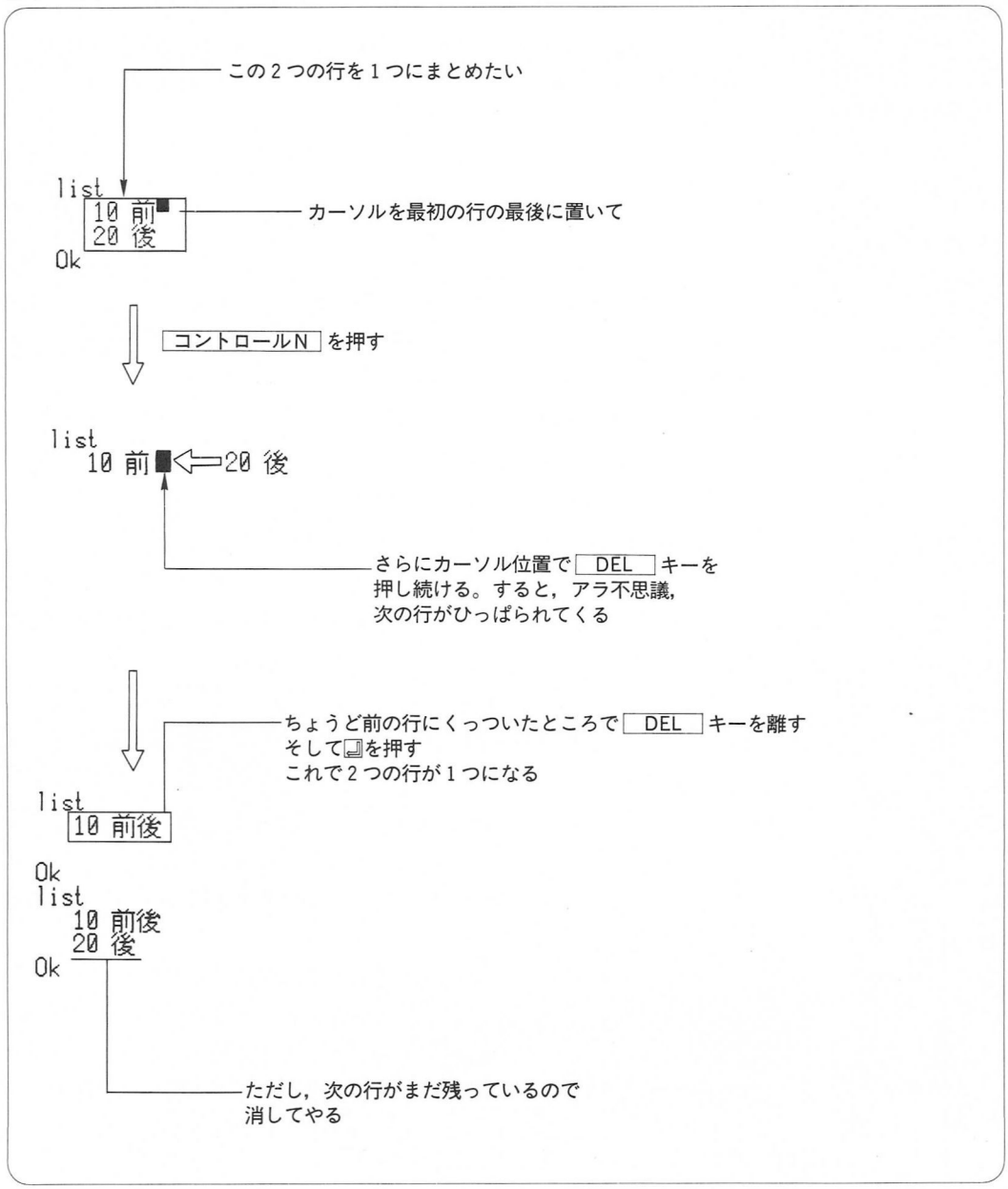
を押します。次に〈DEL〉キーを押しますと（離さないで押し続ける），後の行が引き寄せられてきます。ちょうど前の行にくっついたところで〈DEL〉キーを離します。そして，↵を押します。これで2つの行が1行に統合されます。

なおこのままでは，後の行が残っていますので，後の行を削除してください。

第3-18図　行分割の実習



第3-19図　行の総合の実習



Q20 コントロールコードとは？

A20

キャラクタコード

コントロールコード
キャラクタコード

コントロールコード——まずこの点から説明しておきましょう。

キーボードを押しますと，内部的には**キャラクタコード**と呼ばれる２桁の16進数が発生します。たとえば

Aのキャラクタコード＝41H
５のキャラクタコード＝35H
＄のキャラクタコード＝24H

といった具合です。理論的には，キャラクタコードは

00H～FFH

の256種類があります。ただし，実際にキーボードから入力できるのは20H以降のキャラクタコードに限られます【第３-20図】。

コントロールコードは入力できない

ところでキャラクタコードのうち

00H～1FH

の32種類は，**コントロールコード**と呼ばれています。実はこのコントロールコードの中に

スクリーンエディタを利用する上で大変に貴重な機能

が隠されているのです。ただし，いま説明しましたように**コントロールコードは，直接キーボードから入力することはできません。**ですから何らかのくふうによりコントロールコードをキーボードから入力することができれば，そのコントロールコードに隠された有用な機能を使用することができるわけです。

コントロールコードをキーボードから入力するには？

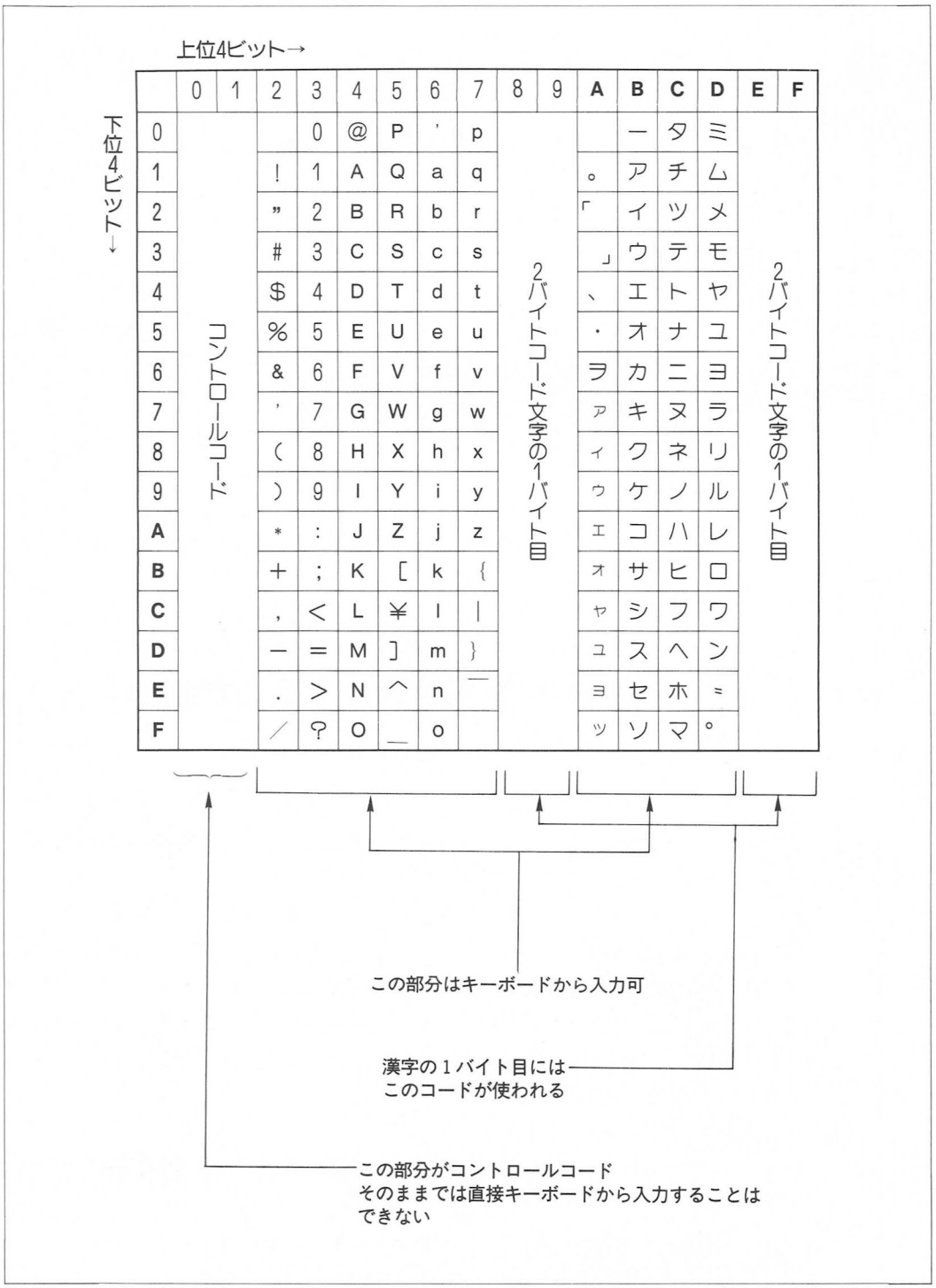
そこで登場するのが，**コントロールキー**です。コントロールキーというのは，

CTRL

〈CTRL〉キー

のことで，その名のとおりコントロールコードを入力するために存在して

第3-20図　コントロールコードとキャラクタコード



います。

〈CTRL〉キーと同時に他のキーを押しますと，そのキャラクタコードから

40H を減じたコード

を発生します（正確には**ビット 6 を 0** にする）。たとえば〈CTRL〉と〈A〉を同時に押したとします。すると，

Aのキャラクタコード＝41H

ですので

41H－40H＝01H

のコードが発生します。これは，01H というコントロールコードを押したことを意味します。このように〈CTRL〉キーと

@(40H) ～_ (5FH)

のキーを同時に押すことにより，コントロールコードを入力することができるのです。

コントロールコードを利用した高度な編集については，次のQで説明します。

Q21 コントロールコードを利用した高度な編集を行うには？

A21

コントロールコードの意味やその入力の仕方については，Q20で説明しました。ここでは，そのコントロールコードを利用した高度な編集の仕方を説明します。なお以下では，〈CTRL〉と〈A〉を同時に押すことを

コントロールAを押す

といった呼び方をします。他のキーとの組み合わせについても同様です。

実は，コントロールコードを利用した高度な編集については，すでに1部を説明済みです。Q19で説明した

コントロールJ……1つの行を2つに分割する
コントロールW……2つの行を1つに統合する

の2つがそれです。その他のコントロールコードについては，次の通りです。

挿入・削除を効率良く

スクリーンエディタにおける編集では，不要な部分を消去したり，新しい文字を挿入するといったことが基本になります。

スクリーンエディタで新しい文字を入力するには，〈INS〉キーを押して**挿入モード**にする必要があります。ところが〈INS〉キーはアルファベットキーから離れた部分にありますので，ブラインドタッチ（キーボードを見ないで打鍵する）のじゃまになります。こんな時は，〈INS〉キーを打つ代わりに

コントロールA　　**コントロールA**

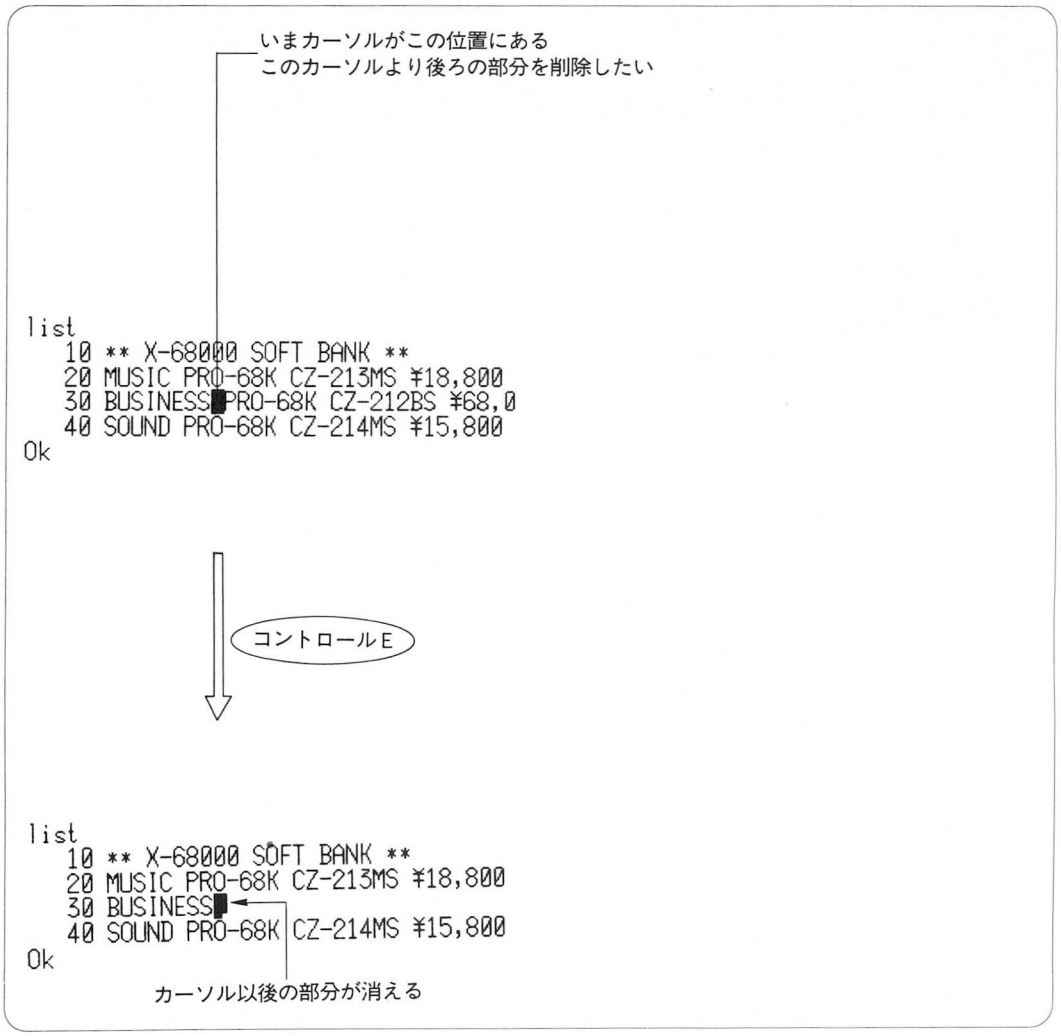
を押してみてください。ちゃんと〈INS〉キーが点灯して挿入モードになります。

逆に文字を消去するには，〈BS〉キーを押します。ところがこの〈BS〉キーも離れた部分にあります。しかし

コントロールH　　**コントロールH**

を〈BS〉キーの代わりに使用することができます。

第3-21図　コントロールE



ブロック消去

行全体を消去するなら行番号だけを入力したり，deleteコマンドを使用することができます。しかし，1つの行の中のまとまった部分を消去したいことがあります。**1行のうちの後半の部分をまとめて消去**するならよい方法があります。それには，カーソルを消去したい部分の先頭に持っていきます。そして，

コントロールE　　**コントロールE**

を押します。これでカーソル以後の部分が消去されます【第3-21図】。

ただし，これだけではまだその行は元のままの状態で残っていますので，必ず最後に

を押すのをお忘れなく！

編集用領域を作る

画面がゴチャゴチャしてきますと編集がやりにくいので，よく〈CLR〉キーを押して画面を消去します。しかし，これだとリストの部分も消えてしまいますので，画面消去後 list コマンドで必要な部分を再表示させてやる必要があります。このような場合，うまい方法があります。1つは，

コントロールZ

コントロールZ

を使う方法です。コントロールZを押しますと，カーソル以後の部分全部を消去してくれます。コントロールZは，コントロールEの拡張版みたいなものです。もう1つは，

〈ROLL UP〉　——カーソルの上に空白行を作る
〈ROLL DOWN〉　——カーソルの下に空白行を作る

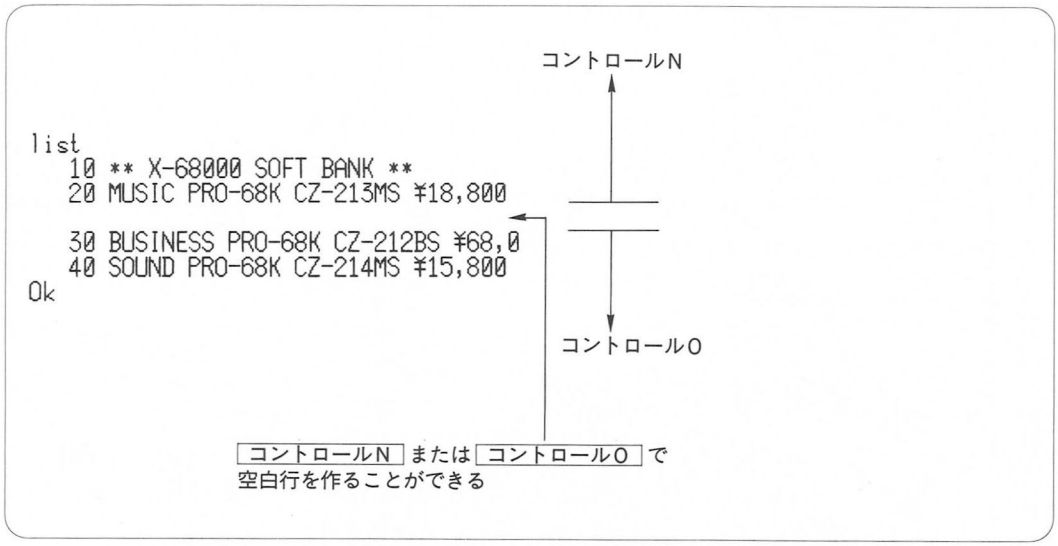
を利用する方法です。ただし，やはり〈ROLL UP〉，〈ROLL DOWN〉キーは離れたところにありますので，これもコントロールキーを使用し

コントロールN
コントロールO

〈ROLL UP〉　——**コントロールN**
〈ROLL DOWN〉　——**コントロールO**

を代わりに使用するとよいでしょう【第3-22図】。

第3-22図　空白行を作る



第3-23図　スクリーンエディットで使えるコントロールコード

CTRL ＋	コード	機　　能	対応キー
エディタの機能を強化するコード			
E	05	現在のカーソル以降1行を消す	
Z	1A	現在のカーソルより下のテキスト画面をすべて消去する	
J	0A	現在のカーソル以降を次の行に分ける	
W	17	現在のカーソルがある行と次の行をつなぐ	
カーソル移動を強化するコード			
B	02	現在のワードの先頭にカーソルを戻す	
F	06	次のワードの先頭にカーソルを進める	
特殊キーの代わりをするコード			
H	08	バックスペース（1文字分消して戻る）	BS
I	09	水平TABを行う	HTAB
A	01	インサートモードにする（トグル動作でON/OFFする）	INS
K	0B	カーソルを画面のホーム位置に移す	HOME
L	0C	テキスト画面を消去する	CLR
M	0D	キャリッジリターンをする	↵, ENTER
N	0E	現在のカーソルから上を上方向にスクロールする	ROLL UP
O	0F	現在のカーソルから下を下方向にスクロールする	ROLL DOWN
¥	1C	カーソルを右へ移動する	→
]	1D	カーソルを左へ移動する	←
^	1E	カーソルを上へ移動する	↑
_	1F	カーソルを下へ移動する	↓

以上を含め，スクリーンエディタで利用できるコントロールコードを機能別にまとめておきます【第3-23図】。

第4章

X-BASICでつまずかないために

Q22　2つの命令実行形式とは？

Q23　命令の3形態とは？

Q24　関数の3形態とは？

Q25　プログラムを途中から実行するには？

Q26　なぜ変数宣言が必要なのか？

Q27　どのような場合に変数宣言を省略できるか？

Q28　文字型と整数型の使用上の違いは何か？

Q29　文字列型の使い方は？

Q22 2つの命令実行形式とは？

A22 2つの実行モード

X-68000には，命令を実行するモードが2つあります。

2つの命令実行形式

ダイレクトモード
プログラムモード

の2つです。X-68000が起動しますと，**コマンドレベル**といって命令が受け付けられる状態になっています。その証拠に

Ok

の表示が現れます。Okは，X-68000の**プロンプト**といって，X-68000がオペレータに対して「いつでも命令を入力してもいいよ」ということを示しています。

ダイレクトモードとプログラムモード

ダイレクトモード

●ダイレクトモード

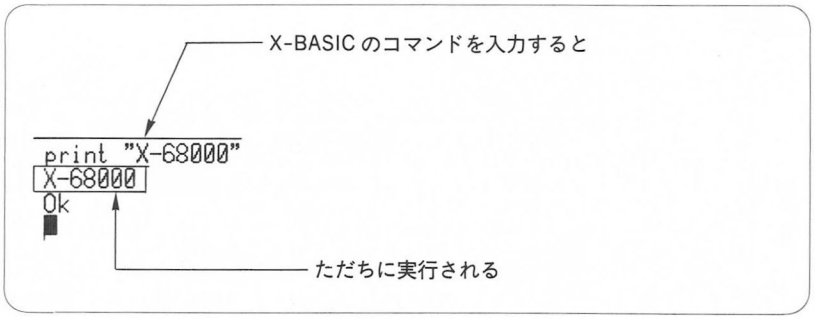
この状態（Okが表示されている状態）で，何か命令――たとえば

print"X-68000"↵

を入力してみます。するとX-68000は，直ちにこの命令を解釈し，実行します**【第4－1図】**。

このようにX-68000がコマンドレベルにある時，命令を投入して直ちに実行させるモードを**ダイレクトモード**といいます。

第4－1図　ダイレクトモード



プログラムモード

●プログラムモード

これに対し，

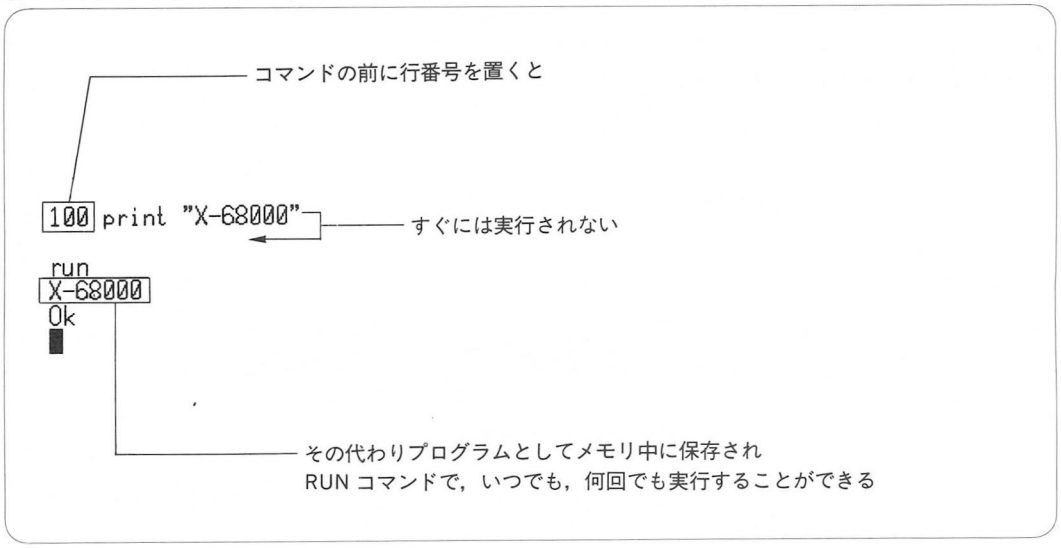
100 print"X-68000"↵

のように命令の前に数字を付けますと，この命令は直ちには実行されません。入力された命令は，プログラムとしてメモリ上に登録されるだけです。ただし登録された命令は，

run ↵

を入力するだけで，何回でも実行することができます【第4－2図】。

第4－2図　プログラムモード



X-68000は，runの命令をキャッチしますと，レベルを

レベルの移行

コマンドレベル
↓
プログラムレベル

に移行させます。そして，メモリ上に登録されているプログラムを逐次実行しようとします。このような命令の実行形態を**プログラムモード**といいます。

ノイマン型と非ノイマン型

ひとまとめの命令をとりあえずメモリ上に蓄えておき，後から一括して実行させる方式のコンピュータを

ノイマン型

ノイマン型のコンピュータ

といいます。数学者**フォン・ノイマン**によって提唱された方式で，今日の

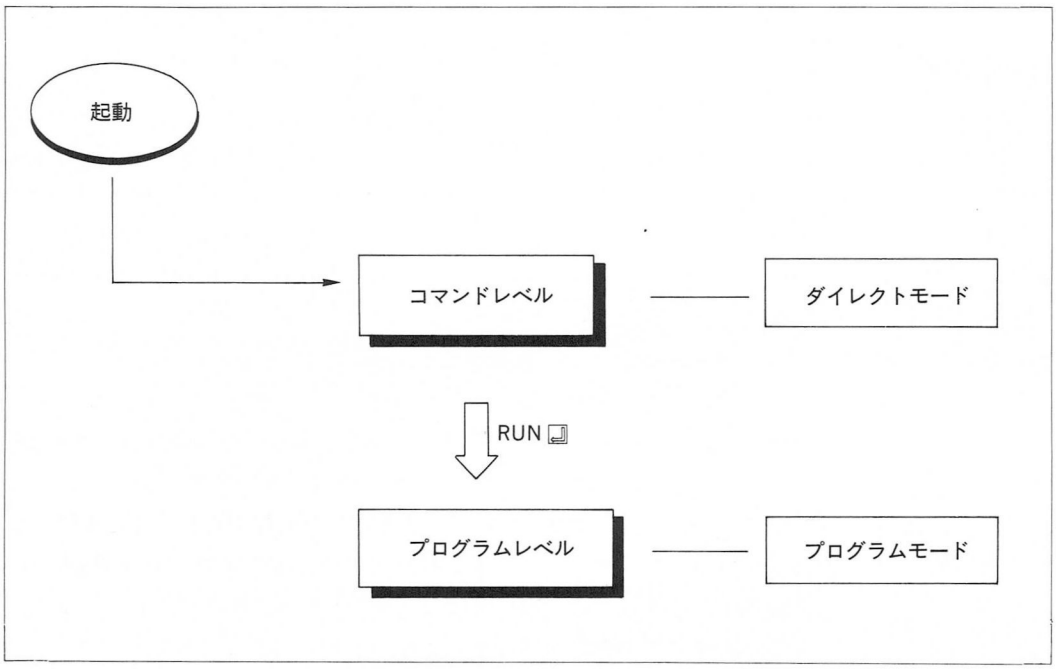
コンピュータはほとんどこの方式を採用しています。ちなみに世界最初のコンピュータといわれているENIACは，ノイマン型ではありませんでした。またICOT（新世代コンピュータ技術開発機構）で研究が進められている第五世代のコンピュータは，ノイマン型の設計方針を捨て新しい非ノイマン型での方式を模索しているといわれています。

X-68000におけるプログラムモードは，まさに典型的なノイマン型によるプログラム実行方式といえます。それに対しダイレクトモードは，オペレータが命令を問うとコンピュータがすぐにそれに答える

対話型による実行形式

になっています【第4－3図】。

第4－3図　2つの命令実行形式



Q23 命令の3形態とは？

A23

命令には3つの種類がある

Q22で述べましたように，命令の実行には2つの形態があります。ところがその2つの実行形態に対し，実行される命令の方も**3つの形態**に分類されます。

命令の3形態

コマンド
ステートメント
関数

の3つです。さらに関数は

関数の種類

標準関数
登録関数
ユーザー関数

の3形態に分かれます。

このように一口に命令といってもいろいろな種類があり，また運用のされ方も微妙に異なります。いやしくもX-68000でプログラムを組むなら，その基本となる命令の形態をしっかり押さえておく必要があるでしょう。そこらあたりをこのQでしっかり押さえておくことにします。

コマンドはダイレクトモードで

まずは，**コマンド**です。コマンドは，X-BASICのマニュアルの上部に《コマンド》と書かれていますのですぐわかります。コマンドは，

run ——プログラムを実行する
kill ——プログラムを削除する

等，主として**プログラムを扱うための命令群**です。そのためコマンドは，その性格からして必ず**ダイレクトモードで実行されます【第4－4図】**。

●例　外

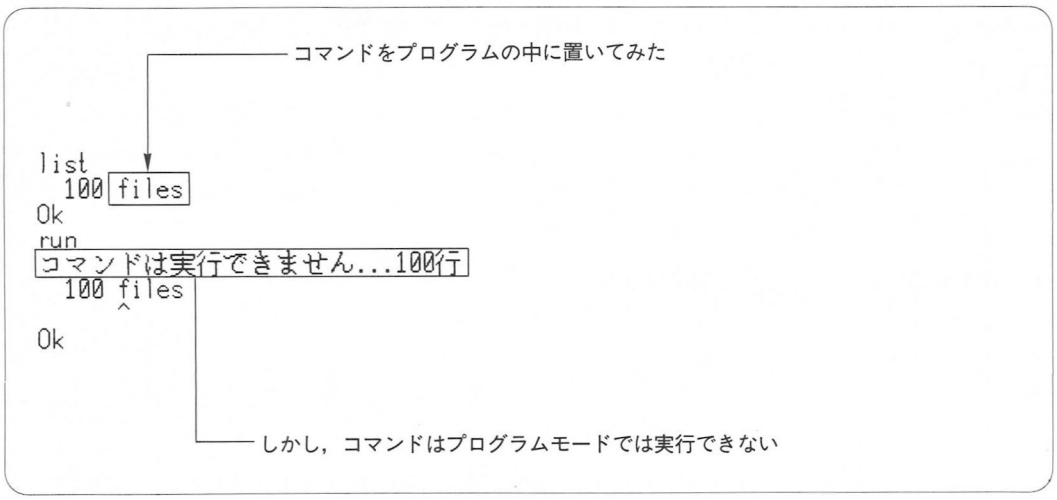
しかし，中には例外もあります。たとえばマニュアルを見ますと

プログラムモードでも使えるコマンド

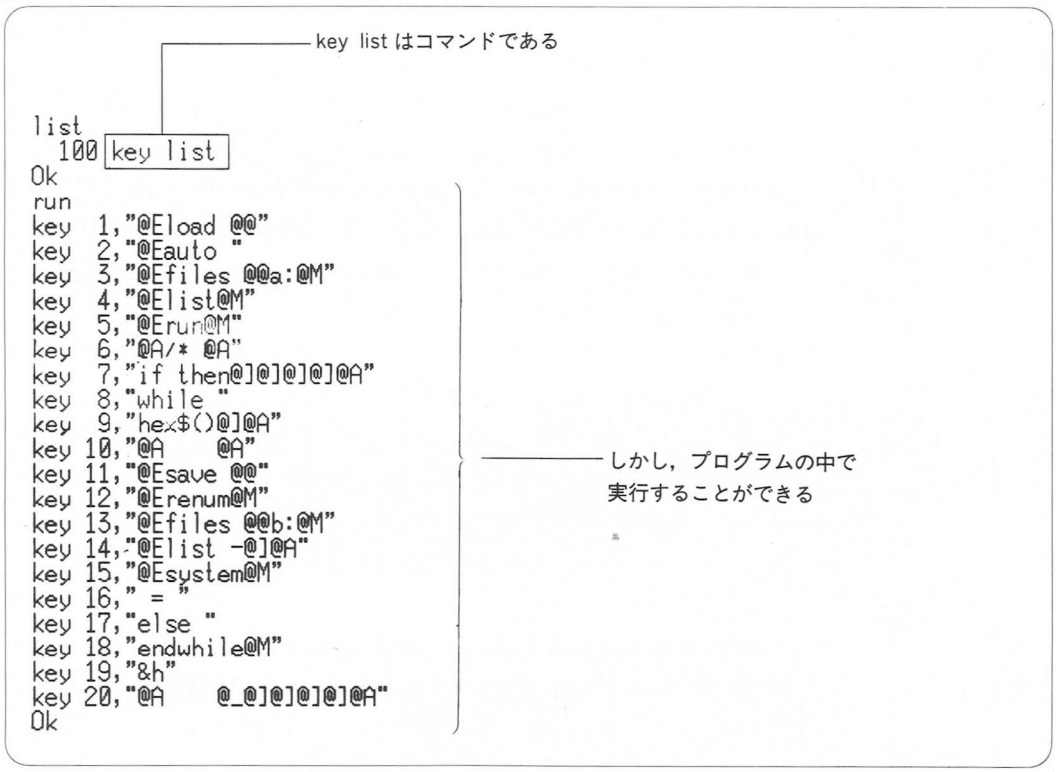
key list

という命令は，コマンドということになっています。しかし，key list はコマンドであるにもかかわらず，プログラムの中で使用することもできます【第4－5図】。

第4－4図　コマンドはダイレクトモードで！



第4－5図　プログラムモードで使えるコマンドもある



ステートメントは両用使い

次に**ステートメント**についてです。

このステートメントこそが，まさに**プログラムを構成する命令群**です。ですからステートメントは，プログラムの中で使用されます。しかし，ステートメントは**ダイレクトモードでも使用することができます**。さもないと，BASICらしさ，ダイレクトモードらしさがなくなってしまいます。

BASICの良さは，プログラムの開発のしやすさにあり，それに大いに貢献しているのがダイレクトモードです。なぜならダイレクトモードですべての命令を簡単に実行することができるからこそ，それがデバッグを容易にしているからです。

ステートメント……この命令は，プログラム中でも，そしてダイレクトモードでも実行することができるのです。

関数とは？

最後に《関数》についてですが，これはその名の通りまさに関数です。

関数は，もともとは数学用語です。中でも最も有名なのは，1次関数

1次関数

$$f(x)=ax+b$$

（a，bは定数でaは0でない）

でしょう。この関数は，**引数xを与えることにより結果f(x)を返します**（この場合は，その1次関数の計算結果を返す）。

X-68000の関数も同様で，引数を与えることにより何らかの処理を行い，その結果を返してくれます。そのふるまいがちょうど命令のように機能しますので，命令として扱っていますが，その実態はあくまでも関数であって

関数の定義

引数を与えると結果が一意に決まるもの

が関数です。とはいえ，X-68000の関数は

・引数を必要としないもの

・結果を返さないもの

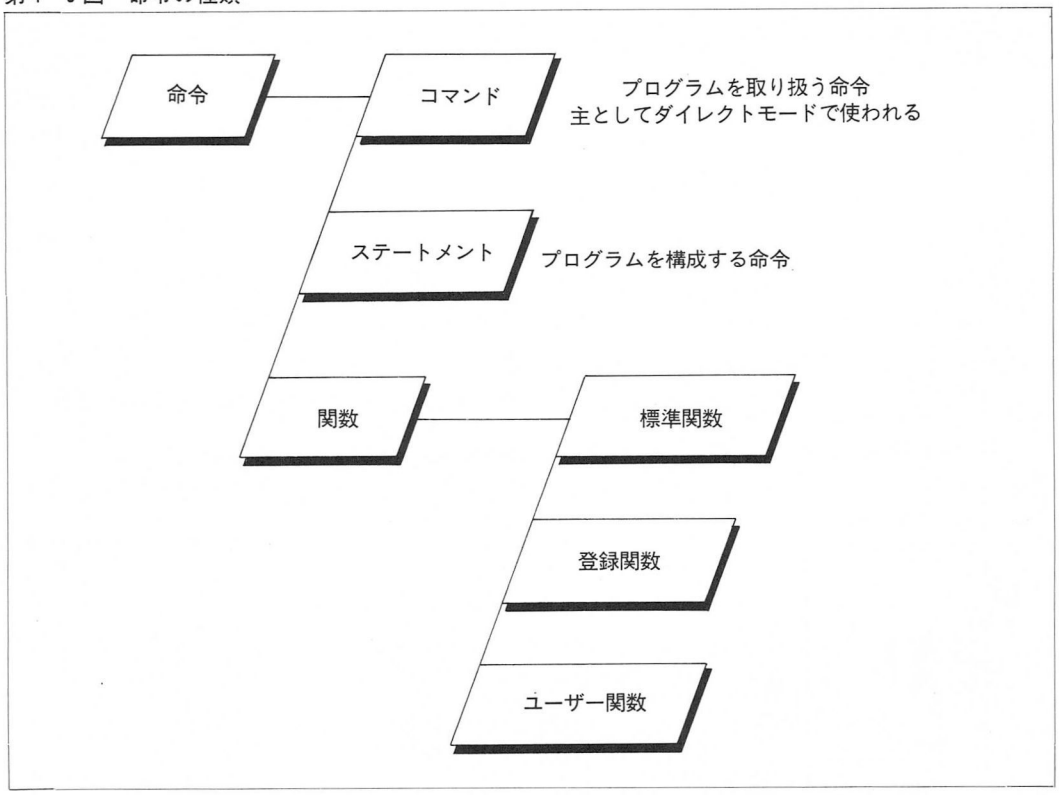
もあって，中には**サブルーチン**のように機能するものもあります。それについては，後の関数定義のところで説明することにします。また，関数には

標準関数
登録関数
ユーザー関数

の3形態がありますが，それについても後のQで説明することにします。

以上のようにX-68000の命令には，3つの形態があります。その違いを**第4－6図**にまとめておきます。

第4－6図　命令の種類



Q24 関数の3形態とは？

A24

ユーザー関数

Q23で命令の3形態を説明しましたが，その1番最後に出てきた関数には，

標準関数
登録関数
ユーザー関数

の3つの異なる形態があります。その違いを説明しておきます。

まずは，**第4－7図**のプログラムをご覧になってください。100行～120行で

function(x)

という関数を定義しています。function( )は，最もポピュラーな関数

10x＋5

ユーザー関数　という1次関数を定義したものです。その実行例を**第4－8図**に示します。

第4－7図　func.bas

```
10 /*
20 /*  1 次 関 数
30 /*
40      int x
50      print "f(x) = 10x + 5"
60      input "          ... x "; x
70      print "f("; x; ") = "; function(x)
80      end
90 /*
100 func function(x)
110     return(10 * x + 5)
120 endfunc
```

関数を使用するためには，定義が必要である

第4－8図　func.bas の実行

```
run
f(x) = 10x + 5
          ... x ? 3
f( 3 ) =  35
Ok
```

適当にxの値を入力する

入力したxに対する関数値が得られる

このようにX-68000は，関数をユーザーが自由に定義することができます。ユーザーが定義した関数を**ユーザー関数**といいます。

標準関数

X-68000のマニュアルを見ていきますと，上部に《**標準関数**》と書かれている命令があります。たとえば

```
abs(n)
```

標準関数

がそうです。abs(n)は，絶対値を返す関数です。たとえば

```
print abs(−5)
```

で，−5の絶対値である5がプリントされます。ところでこの標準関数は，**ユーザーがプログラムの中で定義する必要はありません。**いきなり使うことができます。標準関数は，すでに最初からX-BASICの中に組み込まれているからです。

登録関数

登録関数は標準関数に似ている？

最後に《**登録関数**》についてです。そこで，X-BASICのマニュアルの

```
box( )
```

のところを見てください。上部の命令の分類のところには《**GRAPH**》と書かれています。しかし，これも関数の1つです。たとえば**第4－9図**にbox( )が使われています。もちろんユーザー関数のように定義もしていません。しかし，このプログラムを実行してもエラーとはなりません。円（それに四辺形）が現れるでしょう。

それでは，box( )は標準関数かというとそうではありません。box( )は**登録関数**です。それでは

標準関数と登録関数の違いは何か？

ということになります。

第4－9図　graph.bas

```
10 screen  1, 1, 1, 1
20 box(0, 0, 100, 100, 9)
30 circle(80, 80, 50, 11,,,256)
```

box( )が使えなくなる

その違いを説明するため，次のような実験をしてみましょう。1度

```
system ↵
```

でX-BASICを抜け，Human68kに戻ります。そして

cd basic ↵

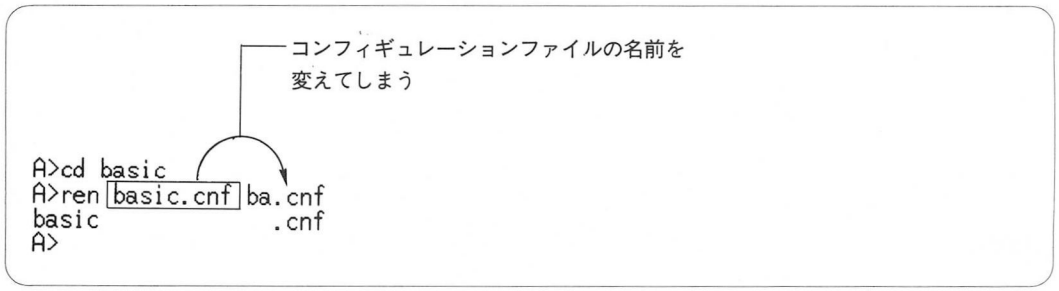
BASIC.CNFの削除

でサブディレクトリBASICに移り，**BASIC. CNF**というファイルのファイル名を適当なものに変更してしまいます。つまり形式的にBASIC. CNFというファイルを滅失してしまうのです【**第4-10図**】。

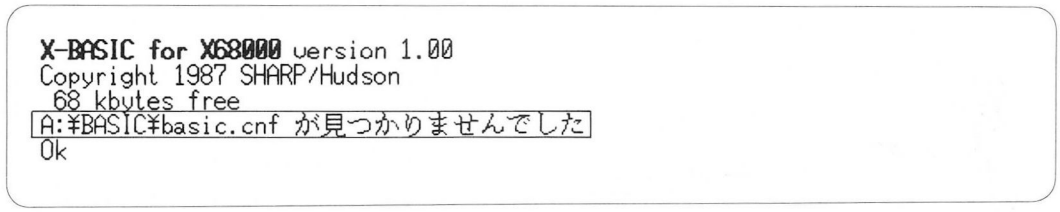
こうしておいてX-BASICを起動します【**第4-11図**】。

そして，もう1度**第4-9図**のプログラムを実行してみます。すると，先ほどは使えたbox( )という関数が使えなくなっているのがわかります【**第4-12図**】。

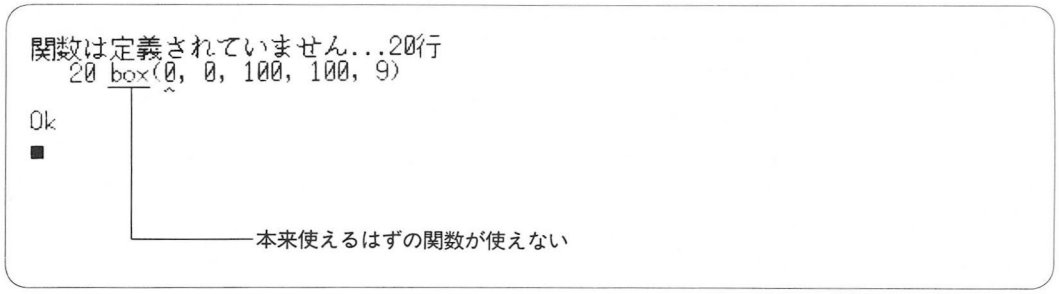
第4-10図　関数の登録を消す



第4-11図　コンフィギュレーションファイルなしで起動



第4-12図　box( )がエラーに！



登録関数

BASIC.CNFは，Q4で説明しましたようにX-BASICが起動する時に参照される特別なファイルでした。その内容は**第4-13図**のようになっています(これは，標準で添付されるものをそのままtypeしたものです)。この最後の6行はいずれも

登録関数の登録

FUNC＝……

の形式をしています。このFUNCこそが，Q 4では説明を省略しておいたX-BASICの関数を登録するためのものです。

先にユーザーが自由に作れるユーザー関数の説明をしましたが，ユーザー関数は常にその定義をプログラムの中に置いておく必要がありました。しかし，アセンブラで関数を記述しますと，その関数をX-BASICの中に登録することができます。それを行うのがBASIC．CNFの

FUNC　＝　……

です。そして，FUNC行により登録される関数が

登録関数

です。これで，標準関数と登録関数の違いが明確になったでしょう。

標準関数——登録の必要はない
登録関数——BASIC．CNFにより登録が必要である

第4-13図　BASIC.CNFの内容

```
A>type basic.cnf
FREE  = 384
WIDTH = 96
BEEP  = ON
CAPS  = OFF
FUNC  = AUDIO  ————————ADPCM用
FUNC  = GRAPH  ————————グラフィック用
FUNC  = MOUSE  ————————マウス用
FUNC  = MUSIC  ————————FM音源用
FUNC  = SPRITE ————————スプライト用
FUNC  = STICK  ————————ジョイスティック用

A>
```

標準で用意されている登録関数

登録関数の良いところは，**必要なければそれを組み込む必要がない**ということです。組み込まなければそれだけメモリに余裕ができます。

一般に登録関数もユーザーが作成するものですが，標準で用意されている登録関数も(たくさん)あります。box( )もその1つです。X-BASICのマニュアルを見ていきますと，**MUSIC**とか**MOUSE**といった表示のされている関数があります(上部の命令の分類のところ)。これらは，すべて登録関数です。これらを使用するには，たとえば

FUNC＝MUSIC

のように登録する必要があります。**第4-13図**を見ますと，標準のBASIC．CNFはすべてこれらの関数が使えるようになっているのがわかります。

Q25 プログラムを途中から実行するには？

A25

基本的なプログラムの実行方法

まずは，**第4-14図**のプログラムをご覧ください。1つのメインルーチン，1つのサブルーチン，1つの関数から成るプログラムです。このプログラムを使用して，いろいろな起動方法を説明します。これら全部の方法を知っている人は，——ウーン，あなたはX-BASICの通りですね。

● run ↵

これが，最も基本的な方法です。**第4-15図**のように実行されます。

● run "run"↵

これは，まだプログラムがメモリ上にない場合の実行方法です。

```
load "run"↵
run ↵
```

としたのと同じです。"run"は，ファイル名で拡張子の.basは省略できます。もしプログラムがドライブBにファイルされている場合は，

```
run "b:run"↵
```

のようになります。この方法も比較的ポピュラーですね。

プログラムの途中から実行するには？

次に**プログラムの途中から実行**する方法です。たとえば70行から実行を開始するとしたらどうしますか？

2つの方法が考えられます。

```
run 70↵
goto 70↵
```

いずれも結果は同じです。プログラムは，第70行から実行されます【**第4-16図**】。

第4-14図　run.bas

```
 10 /*
 20 /*  実行開始の方法
 30 /*
 40      int num = 100
 50      print "メインルーチンです"
 60      print num
 70      gosub 100          ←サブルーチン呼び出し
 80      fntst()            ←関数呼び出し
 90      end
100 /*
110 /* -- サブルーチン                          ┐
120      print                                  │
130      print "私はサブルーチンです"           │ サブルーチン定義
140      print "これでサブルーチンを抜けます"   │
150      return                                 ┘
160 /*
170 func fntst()                                ┐
180      print                                  │
190      print "私は関数です"                   │ 関数定義
200      print "これで関数を抜けます"           │
210 endfunc                                     ┘
```

第4-15図　run.basの実行

```
run
メインルーチンです
 100

私はサブルーチンです
これでサブルーチンを抜けます

私は関数です
これで関数を抜けます
Ok
■
```

第4-16図　途中から実行

```
run 70

私はサブルーチンです
これでサブルーチンを抜けます

私は関数です
これで関数を抜けます
Ok

goto 70

私はサブルーチンです
これでサブルーチンを抜けます

私は関数です
これで関数を抜けます
Ok
■
```

結果は同じに見える

runとgotoの違いは?

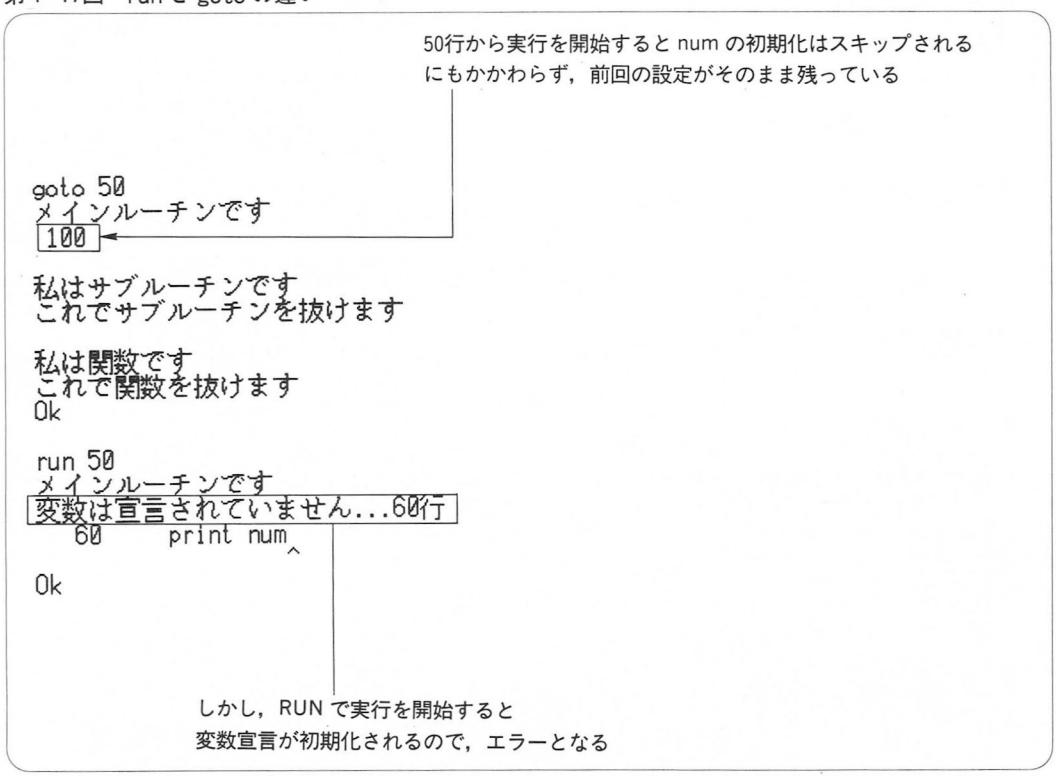
それでは,

プログラムの途中から実行開始

run　《行番号》
goto　《行番号》

の違いはご存じですか？　いずれもプログラムの途中から実行を開始させるのに使えます。そして,**第4-16図**を見る限り両者に違いは見あたりません。しかし，両者には明確な違いがあります。それは，プログラムを50行から実行してみればわかります【**第4-17図**】。

第4-17図　runとgotoの違い



いかがですか？　gotoの方はOkですが，runの方はエラーとなってしまいました。この違い――説明できますか？

変数の初期化

エラーの起こった60行は

```
print num
```

という行です。numは変数ですからX-BASICの規約にしたがって，**変数宣言**がされていなければなりません。それは，40行で行われているのですが，プログラムを50行から実行してしまいますと，その部分がスキップされています。その際，そのことをどう見るかがgotoとrunでは異なります。

gotoとrunの違い

```
goto ——変数の初期化を行わない
run  ——変数の初期化を行う
```

の違いがあります。

すでにこのプログラムは，1回実行されています。そのとき変数numは，変数宣言され，100に初期化されています。gotoは，この値をクリアしません。ですからnumの値は100のままでエラーとはなりません。一方，runの方は必ず実行に先立ち**変数の初期化**を行ってしまいます。これがエラーとなった原因です。

gotoとrun——この両者には，明確な違いが存在しています。おわかりいただけたでしょうか？

サブルーチンや関数のテスト

次に120行からのサブルーチンをテスト（デバッグ）したいとします。どうしますか？　いまと同じように

```
goto  120↵
```

としてプログラムの途中から実行を開始してもよいのですが，最後のreturnのところで**エラー**となってしまいます。このような場合は

```
gosub  120↵
```

とすればよいのです**【第4-18図】**。

プログラムの途中から実行を開始する——しかもそれがサブルーチンの場合は，

サブルーチンのテスト

```
gosub  《行番号》↵
```

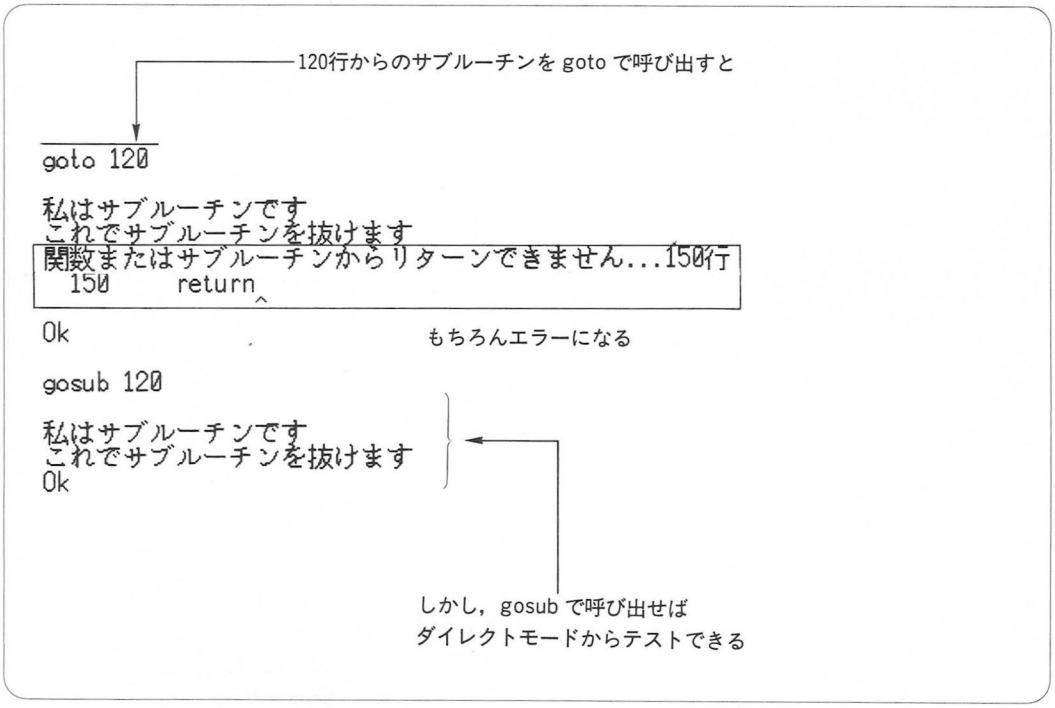
を使用しましょう。同様に関数のテストをするなら

```
fnsts( )↵
```

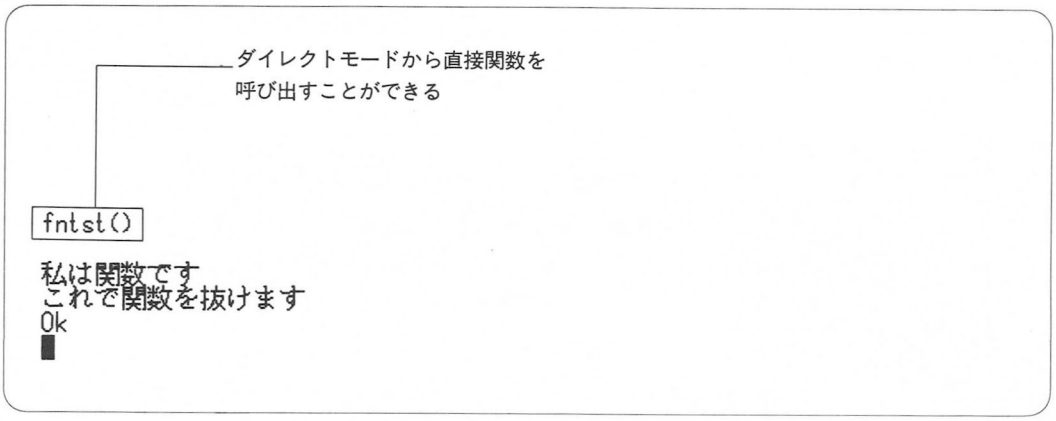
関数のテスト

のように**関数名**を入力するようにします**【第4-19図】**。

第4-18図　サブルーチンのテスト



第4-19図　関数のテスト



Q26 なぜ変数宣言が必要なのか？

A26

X-BASICでは変数宣言が必要

第4-20図のプログラムをご覧ください。input文で数を入力し，それをprint文で出力するというきわめて典型的なBASICのプログラムです。この類のプログラムは，どのBASICの教科書にも載っているでしょう。

さて，このプログラムをX-BASICの上で実行するとどうなるか？ ——もちろん

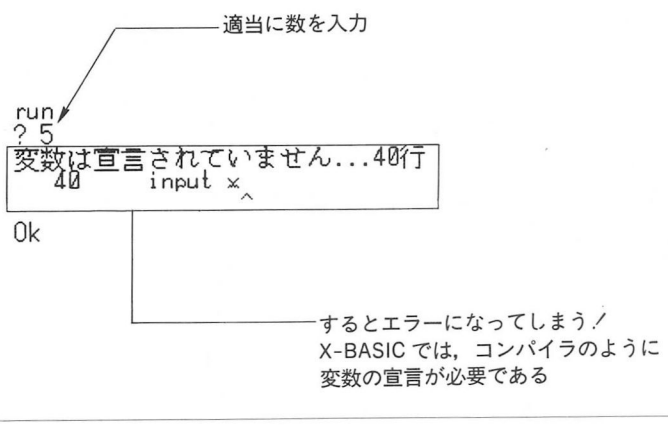
エラー！

となります。X-BASICは，原則として**変数は定義しないと使用できない**からです。それを**変数宣言**と呼ぶことにします【**第4-21図**】。

第4-20図　ごく一般的なBASICのプログラム

```
10 /*
20 /*  平方数の出力
30 /*
40      input x        ————————変数xに数を入力し
50      print x * x    ————————その平方数を出力する
```

第4-21図　しかし，実行するとエラーとなる



変数領域確保の2つの方法

一般に変数宣言は，(プログラマではなく)言語の設計者にとって意味を

持ちます。たとえばX-BASICでは，

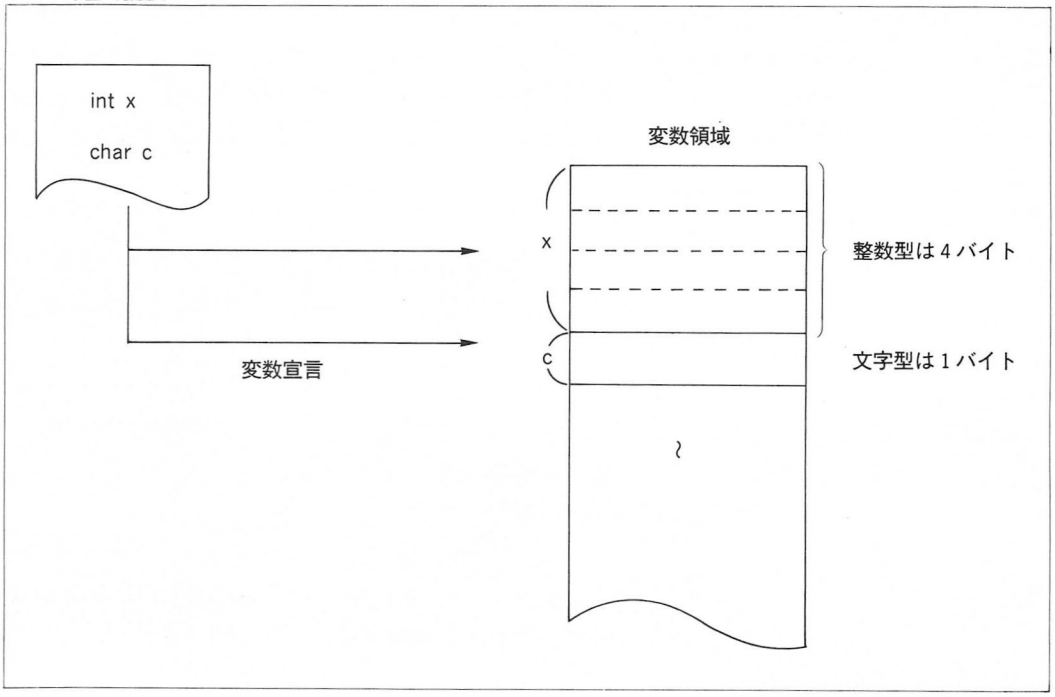
```
int     x
char    c
```

といった変数宣言を行います。この部分を見るだけでX-BASICは，

変数x用に4バイト（整数型は4バイト）
変数c用に4バイト（文字型は1バイト）

の変数領域を用意すればよいということがすぐわかります【第4-22図】。

第4-22図　変数宣言と変数領域



《変数領域の確保》

●静的な変数領域の確保

静的変数

コンパイラならコンパイル時に，またインタプリタなら実行開始と同時に予め変数領域を確保してしまう方法

●動的な変数領域の確保

動的変数

プログラムの進行と同時に新しい変数が現れたところで変数領域を確保する方法

静的な変数領域の確保は，特に1パスコンパイラにとっては有難いことです。これに対し，従来のBASICでは変数宣言が必要ありませんでした。BASICはインタプリタですので，変数が出てくる度にその変数領域を確保すればよかったからです。

変数宣言の意味

X-BASICはインタプリタですので，どちらの方法でもよかったわけです。しかし，現在の新しい言語は一般的に静的な変数領域の確保を取っているケースが多いようです。この方が**プログラム実行中に変数領域を操作しないで済む**（ガベージコレクションも起こらない）という長所があります。X-BASICの場合，後でC言語にコンバートすることも考えていますので，変数宣言があると便利なのです。

プログラムを書く人にとって，変数宣言をすることは一見面倒に見えます。しかし，きちんと使用する変数を宣言することにより，整然としたプログラムを書くことが出来ます。かつその裏では，実行速度やメモリの省力化等に様ざまな恩恵を受けているのです。

変数宣言の基本

変数宣言の基本は，

変数宣言の2要素

変数の型——これにより言語プロセッサはその変数領域に何バイト必要かを知る

変 数 名——変数の識別に使う

の2つを指定することです。**第4-20図**のプログラムに変数宣言を加え，動くプログラムに書き換えたものを**第4-23図**に示しておきます。

第4-23図　変数宣言を追加

```
list
   10 /*
   20 /* 平方数の出力
   30 /*
   35     int x          ←―― 使用前の変数宣言を！
   40     input x
   50     print x * x
Ok
run
? 5
 25                       ←―― エラーにならない
Ok
```

Q27 どのような場合に変数宣言を省略できるか？

A27

Q26で説明しましたように，X-BASICでは《原則》として変数宣言なしに変数を使用することはできません。ただし，これはあくまでも原則であって，場合によっては**変数宣言を省略**できる場合があります。またその方がプログラムが読みやすくなることさえあります。

どのような場合に変数宣言を省略できるのか？　――それをこのQで押さえておきましょう。

変数は参照前に初期値を与える

X-BASICのマニュアルの変数宣言のところを見ますと，

「変数に値を代入する前にその値を参照するとエラーになります」と書かれています。難しそうな表現ですが，要するに

変数を使う前に初期値を代入しておきさえすれば
エラーとはならない

ということが読み取れます。

そこで**第4-24図**をご覧ください。変数xを宣言なしに使用していますが，エラーとはなっていません。これは，変数xの値を参照（ここでは20行でxの値を参照してprintしている）する前に，変数xに初期値5を与えているからです（代入を行っている）。ゆえに

変数宣言を省略するために

《規則1》
変数を使用する前に代入で初期値を与えておけば変数宣言を省略できる

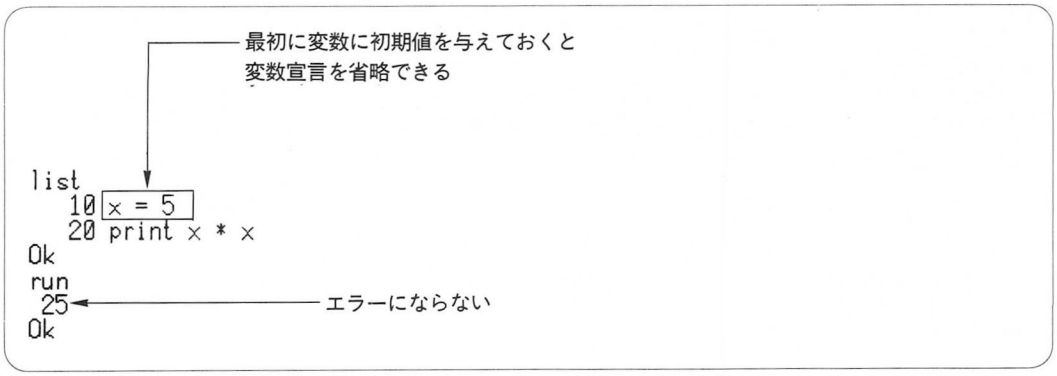
という規則ができあがります。まずこの原則を押さえておいてください。

ただしint型に限定される

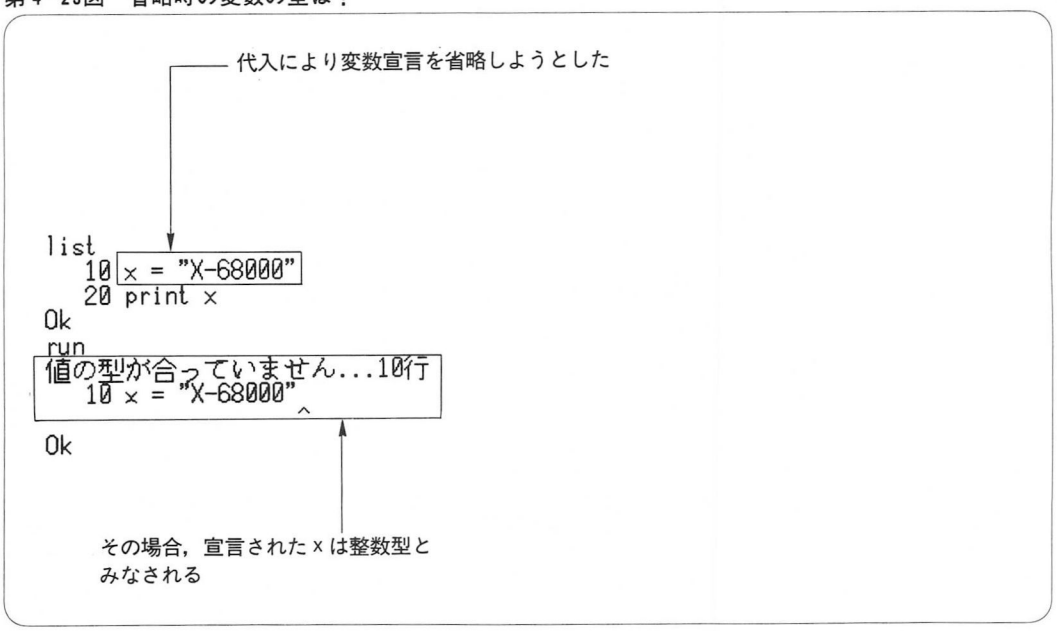
ところが，**第4-25図**の例をご覧ください。10行で変数xに初期値を与えているにもかかわらずエラーとなっています。これは，X-BASICのマニュアルの

「宣言せずに新しい変数に値を代入して使うこともできます。型を宣言しないと自動的にint型の変数とみなされます」

第 4 -24図　変数宣言が省略できる場合



第 4 -25図　省略時の変数の型は？



の部分にひっかかったからです。

変数に初期値を与えることにより，その変数の宣言を省略することができます。しかしその場合，その変数の型は自動的に

int

とみなされてしまうのです。ですから10行で新しく作られた変数ｘはint型（整数型）です。その整数型の変数に文字列を代入しようとしてもそれは無理です。これが**第 4 -25図**でエラーが発生した理由です。やはり文字列型の変数は宣言が必要です**【第 4 -26図，第 4 -27図】**。

制　限

《規則 2 》

初期値を与えて変数宣言を省略できるのはint型だけである

第4-26図　文字列型の変数宣言

```
list
    5 str x
   10 x = "X-68000"
   20 print x
Ok
run
X-68000
Ok
■
```

整数型以外では，やはり変数の宣言が必要

第4-27図　宣言と同時に初期値を与える

```
list
   10 str x = "X-68000"
   20 print x
Ok
run
X-68000
Ok
■
```

宣言と同時に初期値を与えることもできる

規則1，規則2で，X-BASICにおける変数宣言の省略はすべてです。

どういう場合に変数宣言を省略するか?

X-BASICはC言語を意識していますので，やはり使用する変数は予め宣言してから使用するのがよいと思われます。ただし，

積極的に省略

forループのループカウンタ

(for　i=1　のiのこと)

だけは，宣言を省略してもよいでしょう。なぜならforループのループカウンタは，int型に限定されますので省略が可能なのです。それにループカウンタは繰り返しを数えるだけでプログラムの本質とは無関係だからです。それを大げさに宣言する必要はないでしょう。forループに使用する変数は

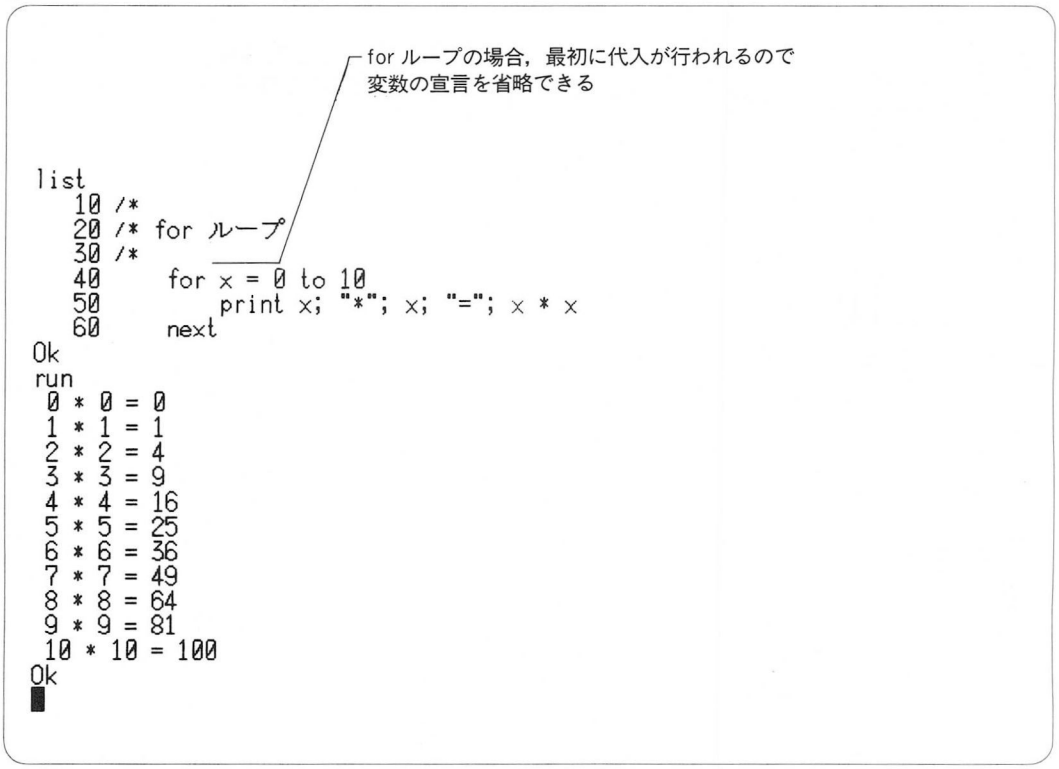
i，j，k，——

(Fortrunのなごり)

等の1文字変数に統一しておくとよいでしょう。

forループのループカウンタを省略したサンプルを**第4-28図**に示しておきます。

第 4 -28図　for ループへの利用




```
list
   10 /*
   20 /* for ループ
   30 /*
   40     for x = 0 to 10
   50         print x; "*"; x; "="; x * x
   60     next
Ok
run
 0 * 0 = 0
 1 * 1 = 1
 2 * 2 = 4
 3 * 3 = 9
 4 * 4 = 16
 5 * 5 = 25
 6 * 6 = 36
 7 * 7 = 49
 8 * 8 = 64
 9 * 9 = 81
 10 * 10 = 100
Ok
```

Q28 文字型と整数型の使用上の違いは何か?

A28

X-BASICは一般のBASICとは異なり,

char型——文字型

が使用できます。そのためBASICの参考書を見ても,あまりchar型の使い方が出ていません。しかしchar型の使い方がわかりますと,メモリの効率化に役立てることができます。また1文字単位の処理は,char型を使って記述する方が自然です。

char型

char型とint型の使い方は同じ

実はchar型とint型はその使い方がまったく同じです。たとえば整数100をchar型の変数bとint型の変数bに代入するのは,いずれも

b＝100

で同じです。またX-BASICは,char型の定数として

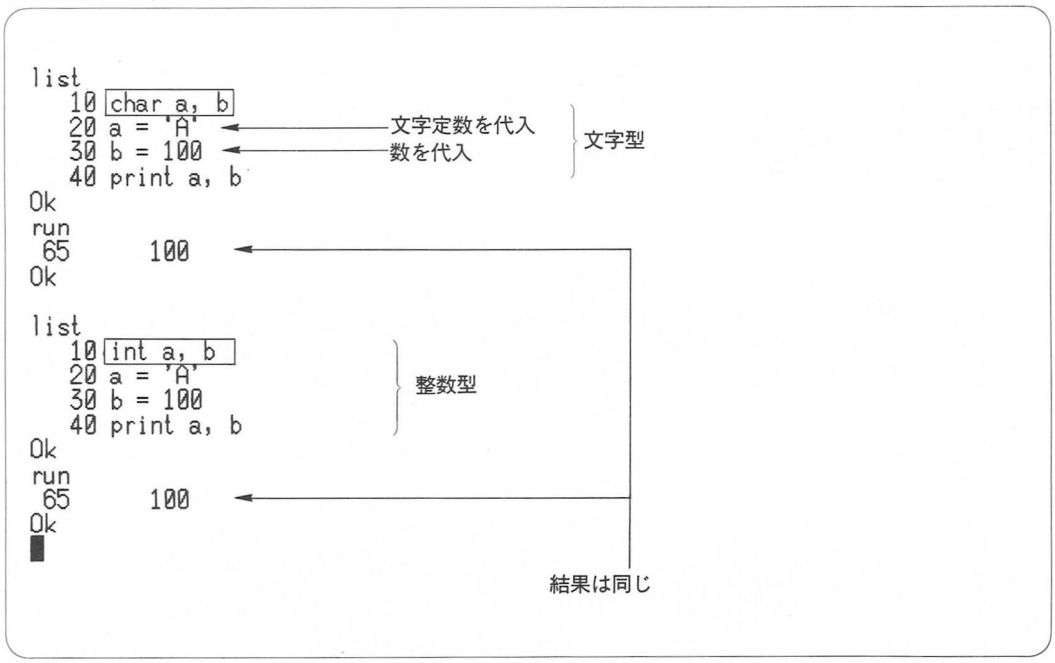
文字定数

文字定数——たとえば 'A'

(1文字を' 'で囲めばよい)

という型の定数を使用することができます。この文字定数もchar型変数

第4-29図　文字型と整数型は同じ?



のみならず，int 型の変数に代入することもできるのです【第 4 -29図】。

《補　注》

文字定数の内部表現は，その文字のキャラクタコードです。ですから文字定数'A'を変数 a に代入することは，A のキャラクタコードである

65（16進数では41H）

を変数 a に代入することと同じです。

char 型の存在意義

これだけを見ますと，char 型が何のために存在するのかわかりません。しかし，**第 4 -30図**の例を見れば char 型の存在理由がわかるでしょう。つまり

char 型の制限

char 型と int 型は使い方がまったく同じである
しかし，char 型は 0 ～255の整数しか扱うことはできない！

という制限があるのです。ですから char 型の変数に255より大きい数を代入しようとするとエラーとなります。

第 4 -30図　違いが現れる！

```
list
   10 char c = 256
   20 print c
Ok
run
型変換に失敗しました...10行
   10 char c = 256
              ^
Ok

list
   10 int c = 256
   20 print c
Ok
run
 256
Ok
```

文字型は 0 ～255の数しか扱えない！

整数型なら OK！

char 型にはこのような制限がありますが，**1 文字（キャラクタコード）を納めるには十分な大きさを持っています。**char 型は int 型の1/4の大きさしかメモリを必要としませんので，文字を扱うアプリケーションの開発には積極的にこの char 型を使用するとよいでしょう。

第 4 -31図，第 4 -32図に char 型を利用して小文字を大文字に変換する

プログラムを示します。

第4-31図　letter.bas

```
 10 /*
 20 /* 小文字を大文字に変換する
 30 /* 入力の最後はコントロールZ
 40 /*
 50     char c
 60     char EOF = &H1A     /* コントロールZ */
 70     while (c <> EOF)
 80         c = asc(inkey$)
 90         if (c >= 'a') and (c <= 'z') then {
100             c = c - ('a' - 'A')
110         }
120         print chr$(c);
130     endwhile
```

第4-32図　実　　行

```
run
KLKJJHBB&&1865HH
Ok
■
```

小文字で入力しても
大文字に変換される

Q29 文字列型の使い方は？

A29

str 型変数の変数宣言

X-BASIC には文字列を扱うための str 型が用意されています。str 型の変数を使用するには(str 型は int 型ではありませんので)，変数宣言が必要です。たとえば str 型変数 s を使いたければ

str 型の変数宣言

```
str　s
```

のように宣言します。str 型の変数には，

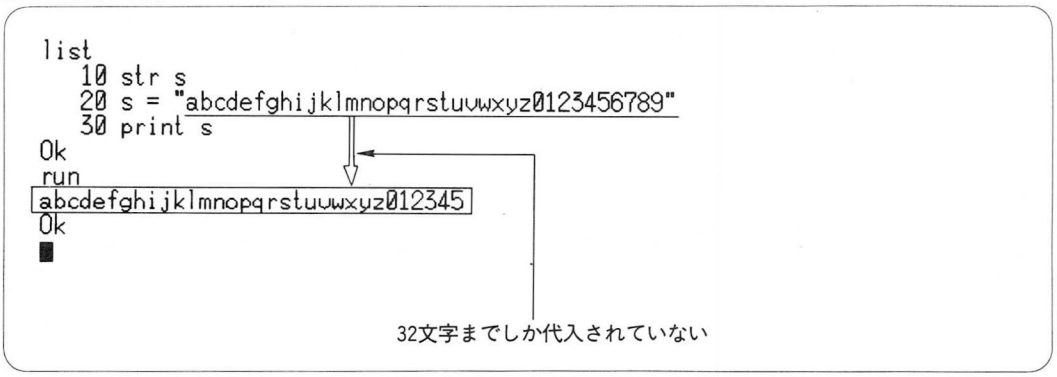
文字列——文字列定数または結果が文字列となる式

を代入することができます。文字列定数は，文字列を" "で囲ったものです。

そこで，**第 4 -33図**をご覧ください。str 型変数 s に**36文字**から成る長い文字列を代入しようとしています。ところが実行結果を見ますと，s には**32文字**分のデータしか代入されていないことがわかります。残りの 4 文字分は捨てられています。

このように特に宣言をしない場合，str 型変数は32文字までの文字列を扱うことができます。

第 4 -33図　文字列は32文字まで？



文字数を指定するには？

このことは，逆にいえば，特別な宣言をしない限り X-BASIC の文字列

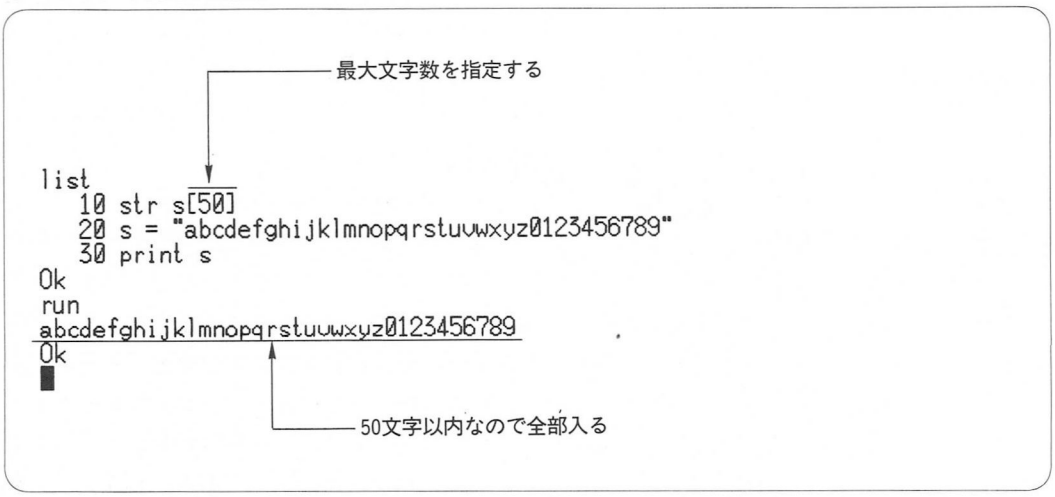
変数は，32文字までしか扱えないことを意味します。それでは，32文字以上の文字列を扱いたい場合はどうするか？　――この場合は，文字列変数を宣言する時に，その**文字数**も一緒に宣言する必要があります。文字数の宣言は

最大文字数を指定

```
str　s[50]
```

のように，その変数で扱いたい最大文字数を［ ］の中に指定します。この場合は，50文字までの文字列を扱うことができるようになります**【第4-34図】**。

第4-34図　文字数を指定



X-BASICにはガベージコレクションは存在しない

なぜX-BASICは，str型の変数を使用するのにその文字数まで指定しなければならないのでしょう？

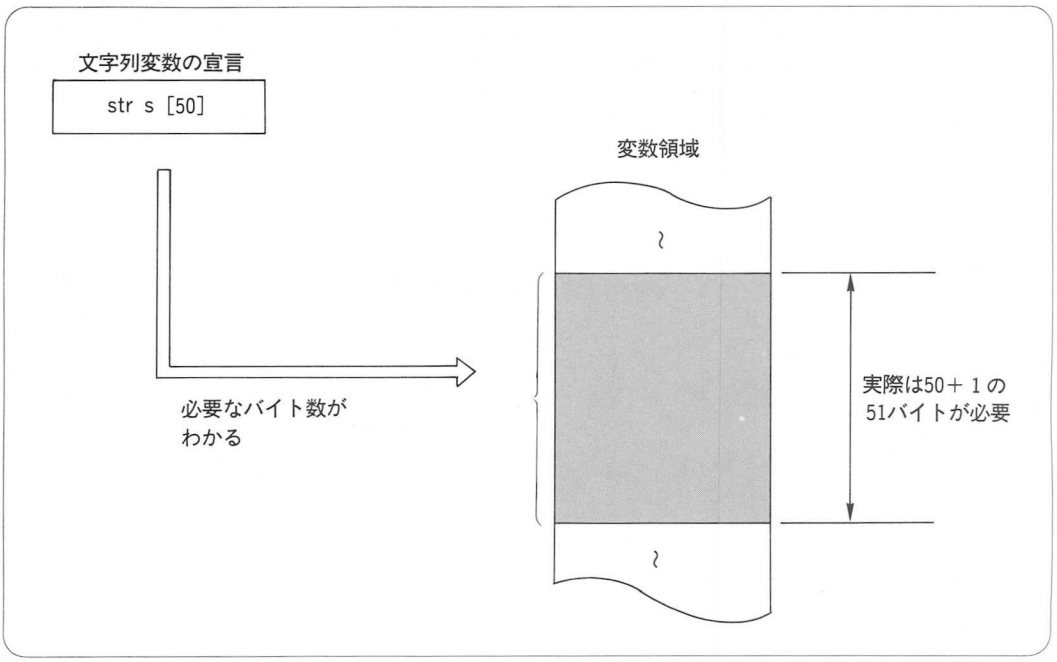
それは，X-BASICが

直接文字列を変数領域に格納している

からです。文字列の長さは不定です。ですから文字列を直接変数領域に格納するためには，予めその大きさがわかっていなければなりません。X-BASICの場合，strの変数宣言により各文字列変数の大きさを知ることができますので(大きさの指定がない場合は32文字分の領域を用意する)，このような芸当が可能なのです**【第4-35図】**。

これに対し，従来のBASICは，文字列変数の大きさが不定なため，変数領域には文字列の先頭アドレス（アドレスならバイト数が一定）だけをおくようにしていました。そして，実際の文字列はそのアドレスが示す文字

第4-35図　文字列変数宣言のメリット



列領域においていたのです。このためプログラムの実行途中で文字列領域が不足するということが起こります。そのような場合は，不要となった文字列を削除するといった処理（これが悪名高き**ガベージコレクション**）が必要だったのです。

X-BASICなら最初に一気に必要なだけの文字列領域を確保してしまうため，ガベージコレクションは起こりません。どちらの方式が優れているかは，明白でしょう。

初期値を与えても文字数の指定が必要

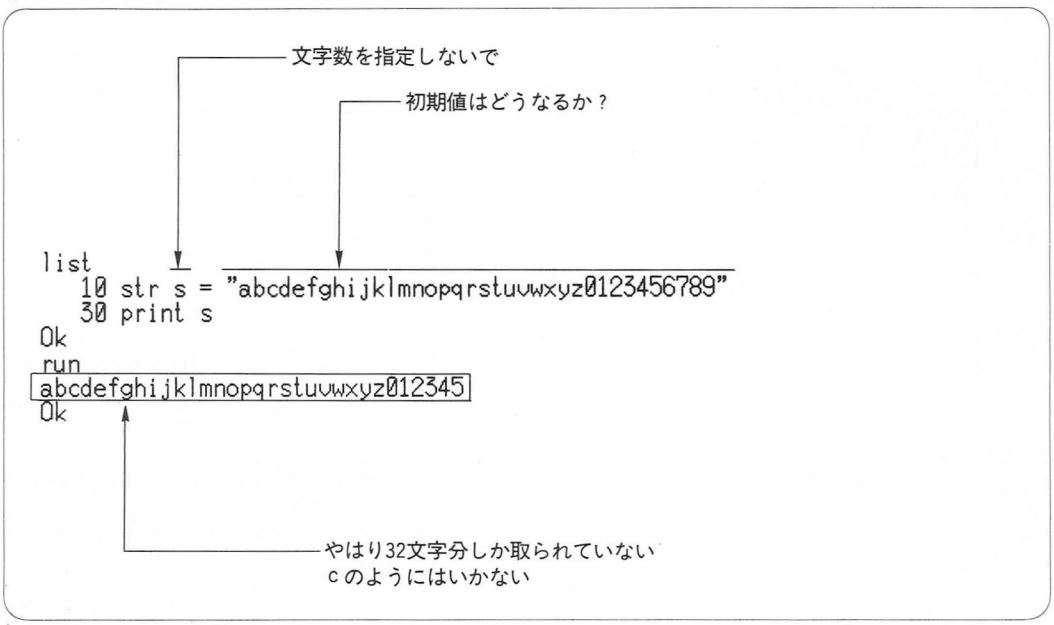
ところでC言語は，文字列変数（実際は文字配列）を確保する際，初期値を与えてやれば文字数を指定する必要はありません。その初期値ピッタシに領域を確保してくれるからです。それでは，X-BASICの場合はどうでしょう？

```
str s= "……"
```

のように初期値を与えて実験してみます。結果は，**第4-36図**の通りです。やはり初期値を与えるにしても必要な大きさを指定する必要があります。

この部分はC言語のようにX-BASICを改良してくれるといいですね。

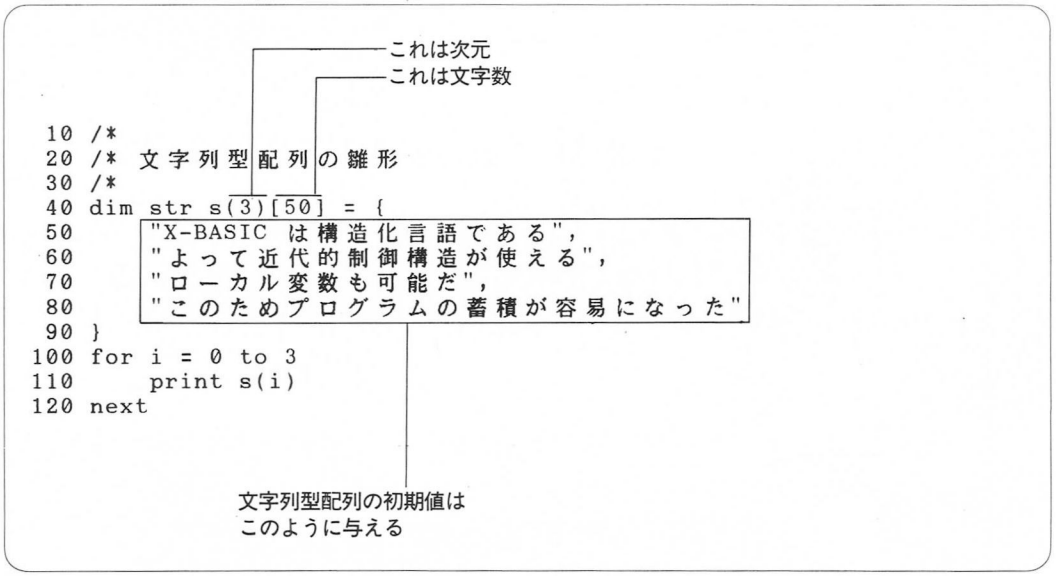
第4-36図　文字数を略して初期値を与えたら？



str型の配列

最後にstr型の配列を使用する場合のサンプルを**第4-37図，第4-38図**に示します。

第4-37図　str.bas



第4-38図　実　　行

```
run
X-BASIC は構造化言語である
よって近代的制御構造が使える
ローカル変数も可能だ
このためプログラムの蓄積が容易になった
Ok
```

次元の指定……（ ）

文字数の指定……［ ］

に注意してください。また**宣言と同時に文字列配列の初期値を与える方法**にも注目してください。この書式がわかれば文字列の使用で困ることはないでしょう。

X-BASIC には，文字列を扱うための関数がたくさん用意されています。1つ1つ使い方を覚えていけば，X-BASIC の達人になれるでしょう。

第5章

構造化プログラミング

Q30　構造化プログラミングとは？

Q31　基本3構造とは何か？

Q32　多重分岐を美しく書くには？

Q33　X-BASICでサポートされている拡張繰り返し文は？

Q34　ユーザー関数の定義の仕方は？

Q35　なぜサブルーチンを使ってはいけないのか？

Q36　ローカル変数とは何か？

Q30 構造化プログラミングとは?

A30 構造化プログラミングとは?

構造化プログラミングは，オランダの

ダイクストラ（E.W.Dijkstra）

により提唱されたプログラミングに関する方法論です。構造化プログラミングは，それまでの自己流によるプログラミングを排し，プログラミングの手法に初めて科学のメスを入れ，その生産性を高めようとしたものです。ダイクストラの論文は，その翻訳がサイエンス社から刊行されていますので，日本語で読むことができます。

構造化プログラミングの手法は，大きく

構造化プログラミングの手法

〈1〉プログラムの構造に関するもの
〈2〉プログラムの設計態度に関するもの
〈3〉プログラムの分割手法に関するもの

の3つに分かれます。

基本3構造とは?

〈1〉の**プログラムの構造**についてはQ31で詳しく説明しますが，その主旨は

基本3構造

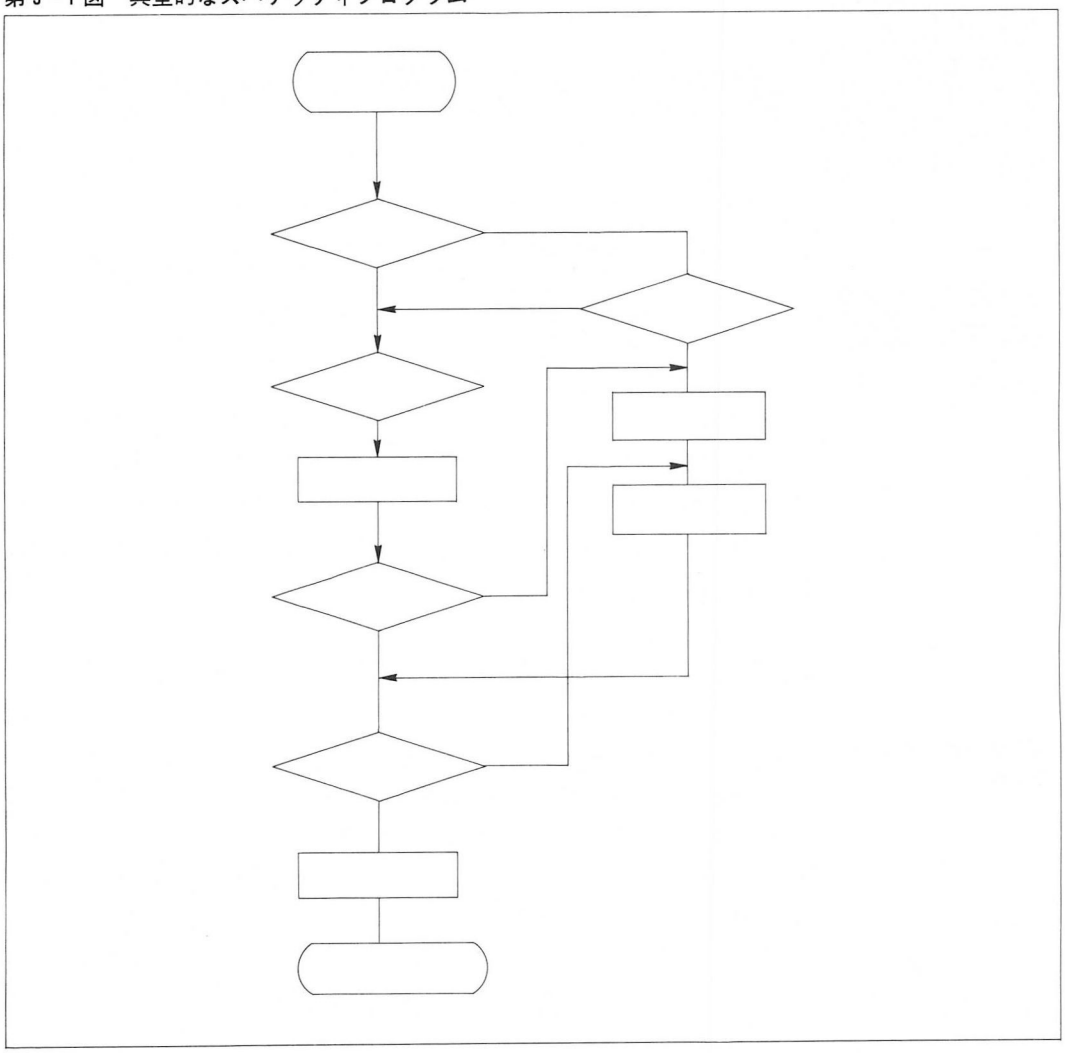
すべてのプログラムは基本的な3構造だけで記述可能である

ということです。そしてその証明もさることながら，

この基本3構造だけを用いてプログラムを記述すれば
読みやすく，バグの少ない，高品質なプログラムを作成できる

というものです。この手法に通じますと，**第5－1図**のような見苦しいフローチャートを見ますと吐き気を催すようになります。反面，**第5－2図**のような基本3構造だけを用いたフローチャートを見ますとホッとするようになります。結構，市販の書籍に掲載されているフローチャートの中にもとんでもないものが見られますので，捜してみると勉強になると思います。

第5－1図　典型的なスパゲッティプログラム



《補　注》

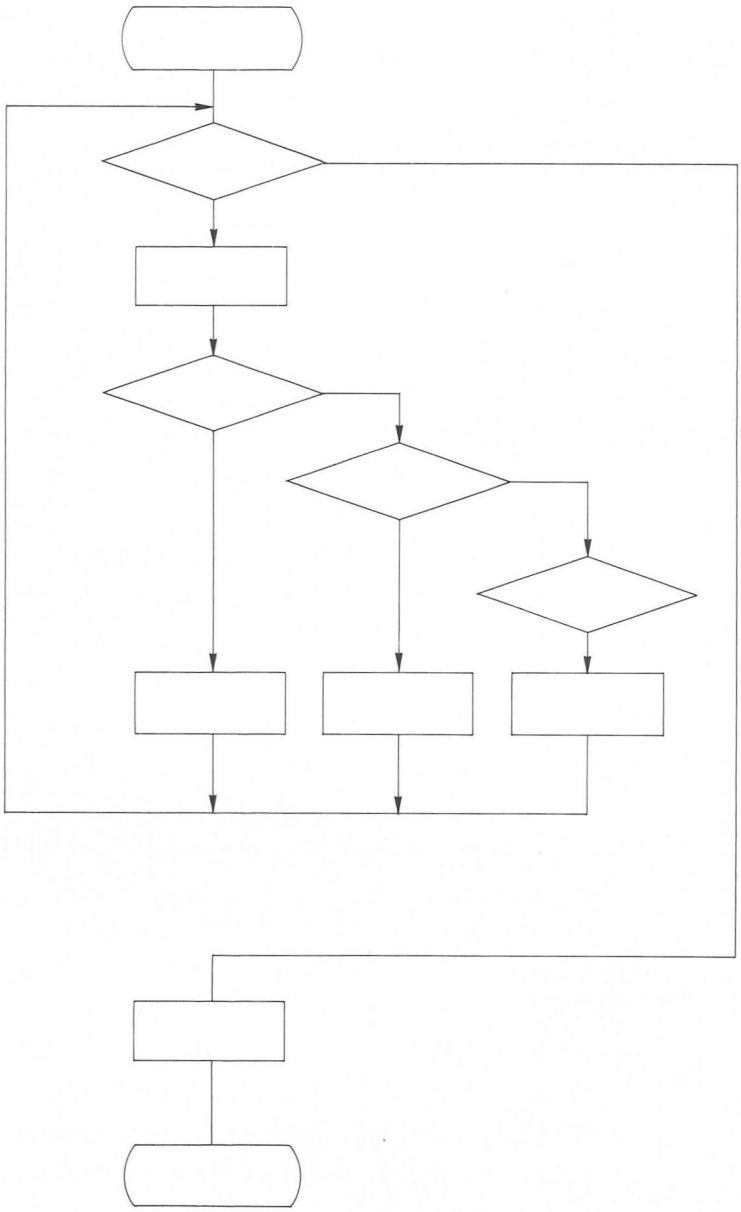
世の中には，フローチャートを非構造化時代の遺物と馬鹿にする人がいますが，とんでもありません。フローチャートを用いたって，基本3構造を立派に記述できますし，いまでも最も構造をつかみやすい図法の1つに変わりはありません。問題は，きちんとその構造がわかっているかどうかです。

トップダウンプログラミング

トップダウンプログラミング
段階的詳細法

〈2〉の**設計態度**については，**トップダウンプログラミング**とか**段階的詳細法**の名前が付いています。未知のプログラムを作成する際，陥りやすいミスの1つに

第5－2図　好感の持たれるフローチャート



※基本3構造のみで記述されている

難しそうなところから作っていく

というのがあります。小さなプログラムの場合は，それでもたいして問題はありません。しかし，大きなプログラムになりますと，この方法ではたいてい失敗します。後で収拾のつかないとんでもないプログラムになって

しまうからです。

そうではなくプログラムは，まずおおまかな**輪郭**から作っていき，徐々に詳細化していきなさいというものです。これは，ちょうど大きな文章を作る際に目次から作っていき，徐々に細部の文章を組み立てていく手法に似ています。トップダウンプログラミングによって設計されたプログラムは，変更がしやすく，また拡張の容易なプログラムに仕上がります。

モジュールに分解

最後の〈3〉は，人間の能力を逆手に取った手法です。すなわち人間が１度に理解できる大きさには限度があって，プログラムも

大きくなる程理解が困難になる

というものです。ですから大きなプログラムは，機能的なまとまり（部品――**モジュール**）に分解し，モジュールのまとまりとしてプログラムを構成しなさいというものです。その時

モジュール

モジュールはできるだけ機能単位に分解する

各モジュールはできるだけ独立するようにインタフェースを考える

ようにします。このようにプログラムを独立したモジュールの組み合わせに構成しておきますと，あるモジュールを変更しても他のモジュールに影響を与えることなく，プログラムの変更が容易になります。またモジュールの再利用も可能になります。

構造化言語としての X-BASIC

従来の BASIC は，この構造化の考え方が取り入れられていないため，非常に構造化プログラミングがやりにくい構造になっていました。これに対し，X-BASIC は新しい考え方の取り入れられた

構造化言語

に仕上がっています。ただし，X-BASIC は従来の BASIC との互換性を保つため，非構造的なプログラムを書くことも可能です(goto や gosub)。もっともそのようなプログラムを書くと，かなり操作性が悪くなるような仕掛け（たとえば renum コマンドに制限がある）にはなっています。

次のQ以下で，X-BASIC の持つ構造化言語の部分にスポットを当てていきます。

Q31 基本3構造とは何か?

A31

基本3構造は，Q30でも触れましたように構造化プログラミングの骨格となる手法で

- すべてのプログラムは基本的な3構造だけで記述可能であること
- 基本3構造だけを用いてプログラムを記述することにより，わかりやすく，バグの少ないプログラムを作成できること

基本3構造

がその基本です。以下，その基本3構造とそれをX-BASICで記述する方法を説明します。

順次構造——sequence

基本3構造の1つ目は，

順次構造

順次構造——sequence

です。これは，「プログラムはできるだけ分岐をなくし，上から下に順に読めるように記述しなさいよ」というきわめて当たり前の教訓です【第5－3図】。

しかし，BASICの世界では，この規則が意外と守られていないようです。意味もなくgoto文を使い，あっちこっちに分岐しているプログラムをよく見かけます。**意味のないgoto文はなるべく使わないようにしましょう**(X-BASICはgotoをまったく使わずにプログラムできるような構造になっています)。

第5－4図，第5－5図に順次構造だけを使用したプログラムを示します。この構造を持ったプログラムは，上から下へ進むだけですから，読むのが非常に楽です。

選択構造——selection

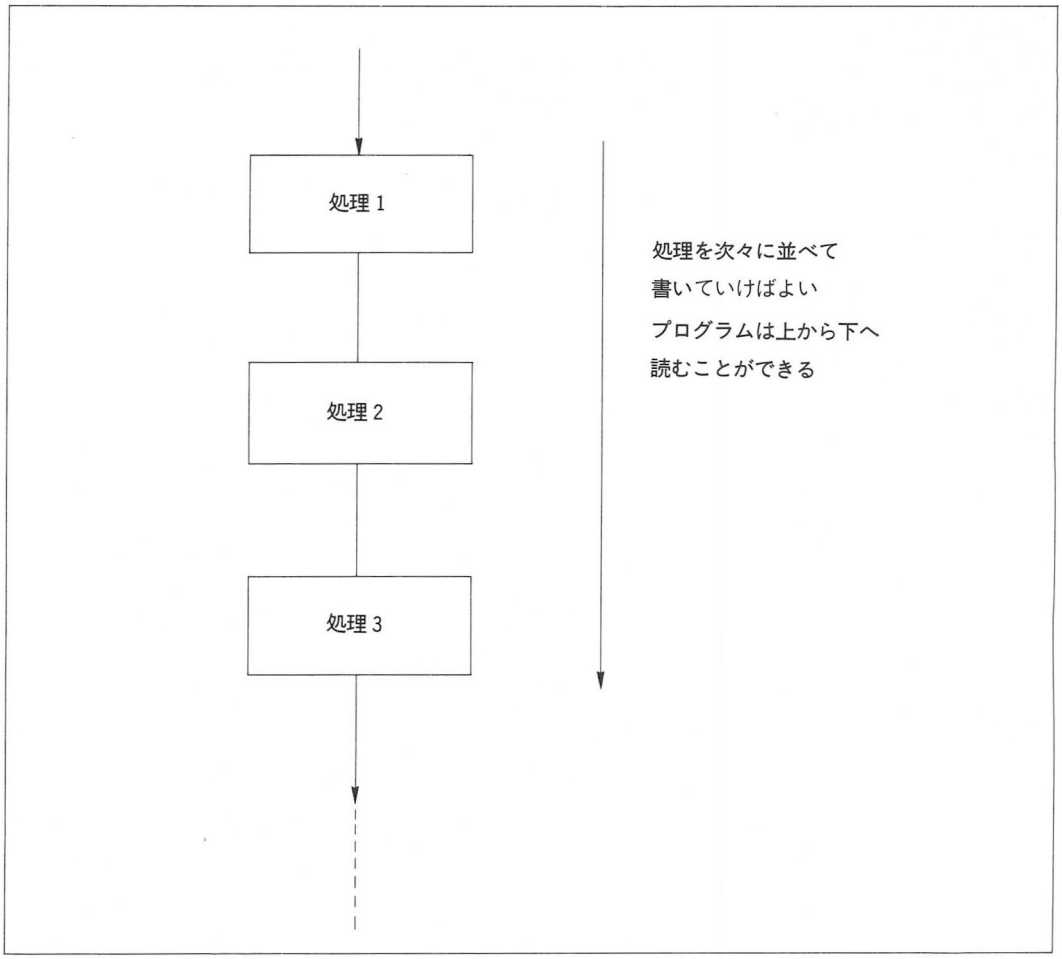
できるだけ順次構造だけを使ってプログラムを書こうとしても，やはりいくつかのケースに場合分けをしなければならない場面が出てきます。そのときは，基本3構造の2つ目である

選択構造

選択構造——selection

を使用します。選択構造によりプログラムの処理は，2つの流れに分離し

第5－3図　順次構造



第5－4図　基本3構造①――順次（sequence）

```
 10 /*
 20 /*  四則演算
 30 /*
 40 int x, y
 50 input "数1  ... "; x
 60 input "数2  ... "; y
 70 print
 80 print x; "+"; y; "="; x + y
 90 print x; "-"; y; "="; x - y
100 print x; "*"; y; "="; x * y
110 print x; "/"; y; "="; x ¥ y; "･･･"; x mod y
```

プログラムは上から下へ読めるように並べる

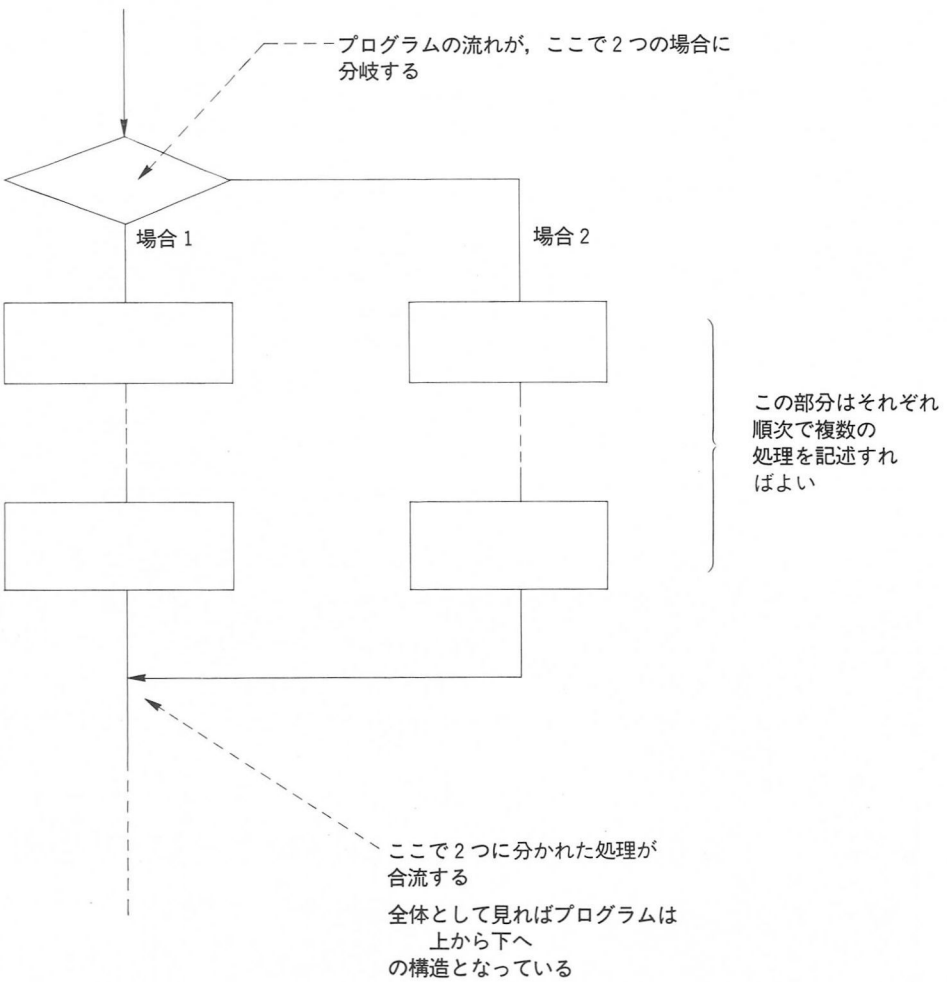
ます。それぞれの流れの中は順次構造で記述できます。そして，最後にこの2つの流れは合流し，再び1つの流れとして下へ流れていきます【第5-6

第5－5図　実　　行

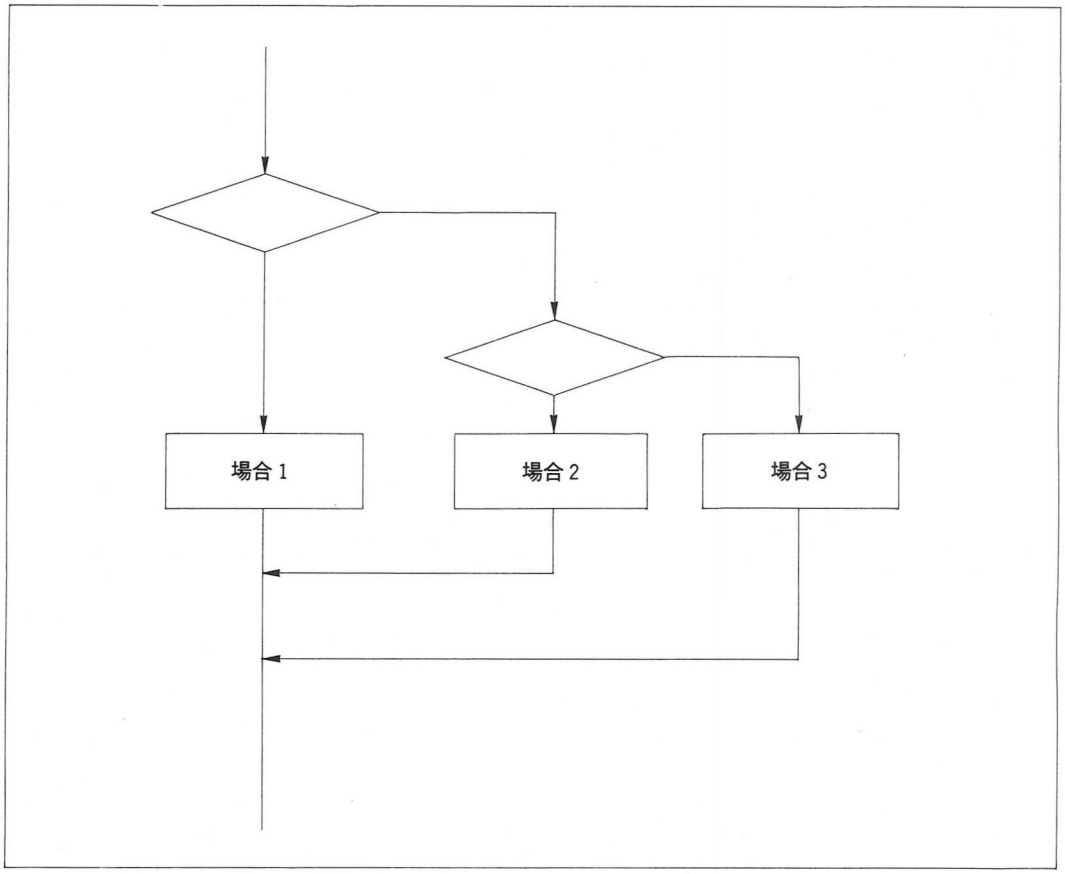
```
run
数1 ... ? 7
数2 ... ? 2
 7 + 2 = 9
 7 - 2 = 5
 7 * 2 = 14
 7 / 2 = 3 ... 1
Ok
```

2数の四則が計算される

第5－6図　選択構造



第5－7図　3つに場合分けする



図】。

構造化プログラミングの選択構造は，2つの流れにしか分岐できないように見えます。しかし，そんなことはありません。選択構造を組み合わせることで，いくつにでも場合分けすることが可能です。問題は，複数の流れに分離させても再び1つの流れに合流させること――これだけを守れば，基本3構造を守った読みやすいプログラムを記述することができます【**第5－7図**】。

従来のBASICで選択構造を記述すると

選択構造は，BASICでいえば

```
if《条件》then～else～
```

に相当します。しかし，従来のBASICですと，if～then全体を1行の中に納めなければならないという制約がありました。このため

if～then　文1:文2:文3――
else　文1:文2:文3――

のようにマルチステートメントのお化けになったり，運が悪いと1行で納まらなくなったりしました。やむをえず

if～then gosub《処理1》else《処理2》

といった方法で逃げていたわけですが，これではとても読みやすいプログラムとはいえません。

X-BASICにおける選択構造の書式

X-BASICなら構造化プログラムに適合したわかりやすいif～thenを記述することができます。なぜなら

- 1つのif～thenを複数の行にまたがって記述できる
- thenの後やelseの後にいくらでも文を記述できる

のように書くことが許されるようになったからです。これにより従来のBASICが持つ**いやらしい制限**からすべて解放されるようになりました。

X-BASICでif～thenを記述する場合，いろいろな書式が考えられます。たとえばC言語を意識した

if～then

```
if（条件）then {
    文1
    文2
    ……
} else {
    文1
    文2
    ……
}
```

のような書式を取るとわかりやすく表現できると思います。gotoによる余計な分岐が必要なく，上から下へ読めることに注意してください。

第5－8図，第5－9図にこの書式にしたがって4つに場合分けをしたプログラム例を示しておきます。

第5－8図　基本3構造②――選択（selection）

```
10 /*
20 /* キャラクタコードの分類
30 /*
40 str s[2]
50 char code
60 /*
70 input "2桁の16進数を入力してください .. "; s
80 code = val("&H" + s)
90 print
100 /*
110 if (code < &H20) then {
120     print "それはコントロールコードです"
130     print "コントロールコードには特別な機能が割り当てられています"
140 } else if ((code >= &H20) and (code <= &H7F)) then {
150     print "それは普通のキャラクタコードです"
160     print "その文字は "; chr$(code); " です"
170 } else if ((code >= &HA0) and (code <= &HDF)) then {
180     print "それはカタカナです"
190     print "その文字は "; chr$(code); " です"
200 } else {
210     print "それは漢字コードの1バイト目です"
220     print "漢字コードは2バイトで構成されます"
230 }
```

第5－9図　実　　行

```
run
2桁の16進数を入力してください .. ? 1f

それはコントロールコードです
コントロールコードには特別な機能が割り当てられています
Ok
run
2桁の16進数を入力してください .. ? 41

それは普通のキャラクタコードです
その文字は A です
Ok
run
2桁の16進数を入力してください .. ? 80

それは漢字コードの1バイト目です
漢字コードは2バイトで構成されます
Ok
run
2桁の16進数を入力してください .. ? b1

それはカタカナです
その文字は ｱ です
Ok
```

入力されたコードを4種類に分類

繰り返し構造――repeat

順次，選択だけでも記述できない処理があります。それが最後の

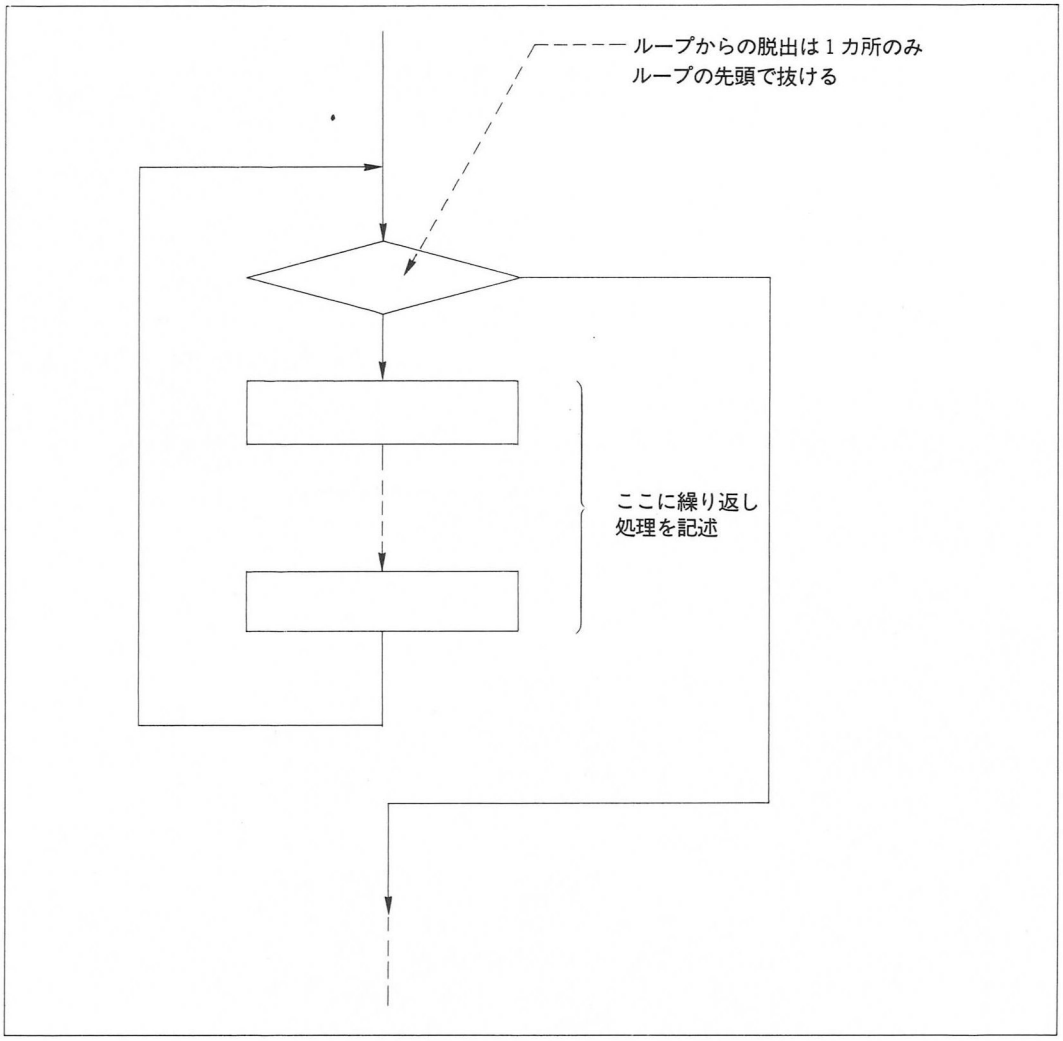
繰り返し構造――repeat

です。この3つが揃って初めてすべてのアルゴリズムを記述できるようになります。構造化プログラムにおける繰り返し構造には，大原則があって

- **ループからの脱出は，1カ所のみ**
- **しかもループの先頭のみに限定する**

を守らなければならないことになっています【**第5-10図**】。

第5-10図　繰り返し構造



ループはともするとプログラムをわかりにくくする原因となります。まして，ループのあっちこっちからあっちこっちへ飛び出してしまっては，読みにくく，かつテストも複雑な構造になってしまいます。ループからの脱出は，**1カ所**のみに限定しましょう。ループからの脱出を**ループの先頭**に限定したのは，すべての繰り返し構造を記述できるようにするためです。なぜなら

条件によってはループの中を1回も通らない

というケースが考えられるからです。この構造を記述できるのは，ループの先頭に脱出のテストがある構造だけです。

X-BASICにおける繰り返し構造の書式

繰り返し構造をX-BASICで記述するには，while文を使用します。その書式は，

while

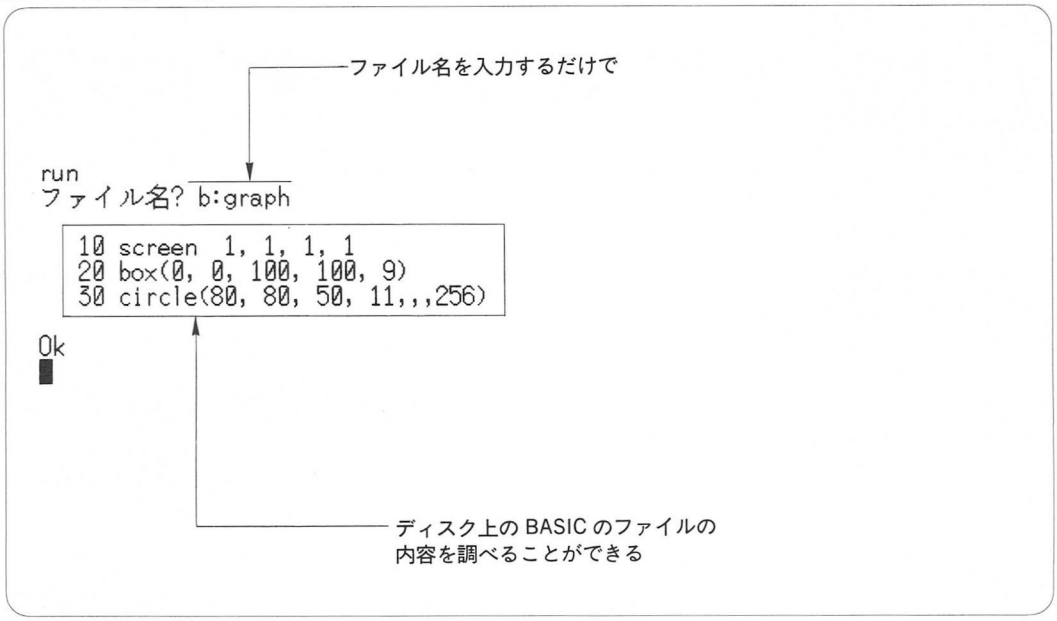
```
while
    文1
    文2
    .....
endwhile
```

のように記述するのがよいでしょう。サンプルを**第5-11図**，**第5-12図**に示します。

第5-11図　基本3構造③——繰り返し（repeat）

```
 10 /*-------------------------------------------
 20 /* basic のプログラムを画面に表示する
 30 /*      ファイル名の入力で .bas は省略する
 40 /*                                  87.10.21
 50 /*-------------------------------------------
 60 error off
 70 str fn, buf[255]
 80 input "ファイル名"; fn
 90 fp = fopen(fn + ".bas", "r")
100 if fp = -1 then {
110     print "ファイルがオープンできません"
120     end
130 }
140 print
150 while not feof(fp)
160     freads(buf, fp)
170     print buf
180 endwhile
190 if fclose(fp) then {
200     print "ファイルがクローズできません"
210     end
220 }
```

第5-12図　実　　行


ファイル名を入力するだけで
run
ファイル名? b:graph
10 screen 1, 1, 1, 1
20 box(0, 0, 100, 100, 9)
30 circle(80, 80, 50, 11,,,256)
Ok
ディスク上の BASIC のファイルの
内容を調べることができる

Q32 多重分岐を美しく書くには？

A32

構造化プログラミングをサポートする制御文は，Q31で述べた通りです。C言語等最近の構造化言語は，これ以外にも**拡張制御文**を設け，プログラムを記述しやすくしています。X-BASICにもC言語に相当する拡張制御文が用意されています。

多重分岐——switch

最初は，選択構造の拡張制御文を紹介します。基本的にはif文があればすべての選択構造を表現することができるわけです。しかし，

かなり多くの分岐

が必要な場合，これをif文だけで表現すると冗長な記述になってしまいます。そのためX-BASICには3つ以上の多重分岐を記述するための**switch文**が用意されています。

switch文の基本は，次のとおりです。

switch

```
switch x
        case 式1   ：文1
        case 式2   ：文2
        case 式3   ：文3
        ……
        ……
        default    ：文n
endswitch
```

xは，基本的には**式**ですが，通常は**変数**をおきます。この変数xの値がたとえば式2と一致すると文2が実行されます。もしどのcaseとも一致しない場合は，defaultの後の文nが実行されます。これにより多重分岐を実現することができます。

breakで脱出させる

switchで注意しなければならないことは，たとえば変数xの値が式2と一致した場合，case 2だけで終わるのではなしに

case 3 以降の文も実行されてしまう！

ということです。これを避けるためには，**break 文**を登場させ

break

```
switch x
        case 式1   ：文1：  break
        case 式2   ：文2：  break
        case 式3   ：文3：  break
        ……
        ……
        default     ：文n
endswitch
```

のように記述します。

case の後に複数の文を置くには?

次に case の後の文の書き方についてです。通常，case の後には**複数の文**がくるのが普通です。それには，

case 式1：文1: 文2： 文3……

のようにマルチステートメント化してしまえばよいのですが，これは感心しません。うまく1行に納まるとは限りませんし，また読みやすい書式ではありません。構造化プログラミングを心がける以上，1つの行には1つの文だけを記述するようにしましょう（C言語では，1つの文を複数行に記述することさえあります)。ですから理想的には

```
case 式1：
    文1
    文2
    ……
    ……
```

のように記述できればよいわけです。しかし，X-BASICでは式1の：の後に1つでも文がないとエラーになってしまいます。そのため

```
case 式1：文1
    文2
    文3
    ……
```

```
        .....
```

と書かねばならずバランスが悪くなってしまいます。そこで，ひとくふうしてみます。まず複文｛｝を用いて

```
case 式1：{
    文1
    文2
    .....
    .....
 {
```

のように記述してみましたが，これは許してくれません（エラーとなります)。最後に苦肉の策として

case

```
case 式1：／*
    文1
    文2
    .....
    .....
    .....
```

のように記述してみましたらようやくうまくいきました。つまり：の後にコメント／*を置いてみたのです。これでバランスの良い書式ができあが

第5-13図　switch. bas

```
 10 /*
 20 /*  2進数の判定
 30 /*
 40 char c
 50 input c
 60 switch c
 70      case 0 : /*
 80      case 1 : /*
 90           print "それは2進数です"
100           print "あなたの入力は"; c; "です"
110           break
120      default : /*
130           print "それは2進数ではありません"
140           break          /* 省略しても良い */
150 endswitch
```

美しく書くには，これがコツ

りました。

この書式を用いたサンプルを**第5-13図，第5-14図**に示しました。なおcase 0の後に文が1つもないことに注意してください。これにより変数cが0，1のいずれの場合もcase 1以後の文を実行させることができます。

第5-14図　実　　行

```
run
? 0
それは２進数です
あなたの入力は 0 です
Ok
run
? 1
それは２進数です
あなたの入力は 1 です
Ok
run
? 2
それは２進数ではありません
Ok
```

２進数かどうかチェックされる

Q33 X-BASICでサポートされている拡張繰り返し文は？

A33

ループの最後で抜ける――repeat

基本3構造による繰り返し構造は，X-BASICでは，ループの先頭で抜けるwhile文が対応しています。しかし，場合によってはループの最後で抜けた方が効率が良い場合があります。それは，ループの中を必ず1回は通る場合です。このようなケースのためにX-BASICではrepeat文が用意されています。その書式は

repeat

```
repeat
    文1
    文2
    .....
    .....
until《式》
```

となっています。この繰り返し構造は必ずループの中を1回は通ります。そして，最後の式の値が真（0でない数）になるまで繰り返されます。

サンプルを第5-15図，第5-16図に示します。

第5-15図　repeat. bas

```
10 /*
20 /*  入力チェック
30 /*
40 int x
50 repeat
60     input "1から6までの数を入力"; x
70 until (x >= 1) and (x <= 6)
```

第5-16図　実　　行

```
run
1から6までの数を入力? 0
1から6までの数を入力? 7
1から6までの数を入力? 5
Ok
```

1～6の数が入力されるまで，繰り返される

繰り返しの回数がわかっている場合——for

X-BASICの繰り返し文は，もう1つ**for文**が用意されています。ただし，X-BASICのfor文はC言語のものより過去のBASICの構文に近いものになっています。つまり繰り返しの回数が予めわかっている場合に便利な構造になっているのです。その書式は

for

```
for ループ変数＝初期値 to 最終値
    文1
    文2
    ……
    ……
next
```

です。サンプルを**第5-17図，第5-18図**に示します。

第5-17図　for. bas

```
10 /*
20 /* for ループ
30 /*
40 for i = 1 to 6
50     for j = 1 to i
60         print "□";
70     next
75     print
80 next
```

内側のfor　　外側のfor

第5-18図　実　　行

```
run
□
□□
□□□
□□□□
□□□□□
□□□□□□
Ok
```

Q34 ユーザー関数の定義の仕方は？

A34

X-BASICでは，原則として**サブルーチンを使いません**。その代わりサブルーチンよりもはるかに強力な

関数

を使用することができます。関数を使用することで**行番号に依存しない**プログラムの部品を作ることができます。また関数内部だけで使用可能な**ローカル変数**も使用できるようになります。

関数定義と呼び出し——単純版

以下，X-BASICにおける関数の使い方を段階的に説明していきます。

最初は，最も単純な形の関数です。そこには，引数も戻り値も存在しません。この形の関数はほとんどサブルーチンと同じと考えてよいでしょう。

●関数定義

関数を使用するには，それを**定義**しなければなりません。最も単純な関数定義は，

関数定義（単純版）

```
func 関数名（）
    文1
    文2
    ……
    ……
endfunc
```

のように行います。func **関数名（）と** endfuncの間がサブルーチンのようなものです。関数名は，適当な名前で結構です。もちろんX-BASICの予約語は使えません。できるだけその関数の機能を連想させるものがよいでしょう。

●関数の呼び出し

関数呼び出し

定義した関数をメインルーチンで使うには

関数名（）

のようにその関数の名前と（）を記述すればよいのです。これによりちょ

うどサブルーチンの呼び出しのようにその関数で定義した内容が実行されます。

単純版関数の定義と実行例を**第5-19図，第5-20図**に示します。

第5-19図　func1．bas

```
 10 /*
 20 /* func1.bas
 30 /*    -- 引き数・戻り値なし
 40 /*
 50      hello()
 60      hello()
 70      end
 80 /*
 90 func hello()
100     print "X-68000"
110 endfunc
```

関数呼び出し

関数定義

第5-20図　実　　行

```
run
X-68000
X-68000
Ok
```

引数がある場合の関数呼び出し

関数は，サブルーチンと異なり

引数――関数の呼び出し側が関数に送信するデータ

を指定することができます。たとえば2数の和を計算してprintするadd（）という関数を定義したとします**【第5-21図，第5-22図】**。

その場合，メインルーチンは関数の呼び出し時に

```
add (10, 5)
     ↑   ↑
     └───┘
```

この場合，10と5の加算が行われる

のように（）の中に引数を並べるようにします。

第5-21図　func2．bas

```
 10 /*
 20 /* func2.bas
 30 /*    -- 引き数あり・戻り値なし
 40 /*
 50      add(10, 5)
 60      end
 70 /*
 80 func add(x, y; int)
 90      print x; "+"; y; "="; x+y
100 endfunc
```

引数を渡す

第5-22図　実　　行

```
run
 10 + 5 = 15
Ok
```

引数がある場合の関数定義

この場合，関数の定義は

関数定義
（引数付き）

```
func 関数名（引数定義）
    文1
    文2
    ……
    ……
endfunc
```

のようにします。つまり関数名の後の（ ）の中に引数の定義を書くのです。引数定義は，

関数名（x，y）――このままではまだ不完全

のように引数の数だけ適当な**変数名**（これを仮引数といいます）を並べます。するとこれらの変数に，メインルーチンで指定した引数の値が入ります。ただし，変数名だけではその型がわかりませんので

関数名（x；int，y；int）

のように変数の後に；（セミコロン）をつけてその**変数の型**を示します。なお同じ型の変数が続く場合は

関数名（x，y；int）

のように**型をまとめる**こともできます。

戻り値がある場合の関数

さらに関数は，**戻り値**を返すことができます。つまり引数をもらい，その引数に何らかの操作を加えた上で値を返す――ゆえに関数であるわけです。この場合の関数定義は

関数定義
（戻り値あり）

```
func 型 関数名（引数定義）
    文1
```

```
    文2
    .....
    return（戻り値）
endfunc
```

の形になります。関数名の前に**型**（intとかstrとかchar等）が入ります。これは何かというとその関数が返す値の型です。関数は，値を1つしか返すことができませんので，その型をここに示すのです。

関数の戻り値は，**return（ ）の（ ）の中**に書きます。**第5-23図**の第90行がそれです。この関数err（ ）は，文字列を返しますので関数の型にstrを指定していることに注意してください。

このように戻り値を返す関数を使う場合は，普通に式の中にその関数名を置けばよいのです。たとえばこのerr（ ）は，print文の対象となっています。

第5-23図　func3．bas

```
 10 /*
 20 /* func3.bas
 30 /*    -- 引き数あり・戻り値あり
 40 /*
 50      print err("ラベル未定義")
 60      end
 70 /*
 80 func str err(s; str)
 90     return ("エラー : " + s)
100 endfunc
```

戻り値をそのまま print

戻り値の型

戻り値は return（ ）で返す

第5-24図　実　　行

```
run
エラー : ラベル未定義
Ok
```

Q35 なぜサブルーチンを使ってはいけないのか？

A35

俺は，いままでのBASICでサブルーチンに慣れている。X-BASICにもgosub文があってちゃんとサブルーチンが使えるではないか。なぜサブルーチンではなく，面倒くさそうな関数を使わなければいけないのか？ ——この疑問に答えておきましょう。

出　発

いま，”口”という文字を7つ並べた

目標→

口口口口口口口 ……〈1〉

を表示したいとします。これをfor～nextを使って表現しますと

```
for i= 7
    print ”口” ;
next
```

となります。セミコロン；によりprint後の改行を抑えていますので，最後にprint行を付け

```
for i= 7
    print ”口” ;
next
print
```

とすればなお結構でしょう。

サブルーチン化する

せっかくこのようなプログラムの部品ができましたので，**サブルーチン化**してみます。最後にreturn文を付け

サブルーチン化

```
for i= 7
    print ”口” ;
next
print
return
```

のようにすればOkですね。このサブルーチンをたとえば200行から

```
200  for i= 7
210      print ”口” ;
```

```
220        next
230        print
240   return
```

のように入力したとします。するとダイレクトモードから直接

```
gosub  200↵
```

と入力することで<1>のような図形が得られるでしょう。もちろんメインルーチンから呼びだして使用することもできます。

サブルーチンでは困ること

これで<1>を表示するサブルーチンができあがりました。プログラムの部品として，使えます。もちろんX-BASICの上で動きます。

「何だ問題ないじゃないか！　カッコつけて何も関数を作らなくともサブルーチンだけで十分いけるじゃないか」

と思われるでしょう。

――ところが，ところが，これを呼び出すメインルーチンを作ったとします。すると，このサブルーチンを呼び出すには

```
gosub  200
```

とします。この200という**行番号**が問題なのです。なぜ問題なのかというと

行番号の問題

- **行番号200という数を見ても，直感的に何のサブルーチンかわからない**
- **1度入力したサブルーチンは，その行番号が固定されている**

の2つの問題があるからです。関数は名前で呼びだしますから，その関数名を見れば何をしているのかすぐにわかります。

X-BASICでは非構造化文を書きにくくしてある

1度入力したサブルーチンの行番号が固定されているのも（かなり）問題です。

行番号が固定されていると

- **メインルーチンをそのサブルーチンの行番号に合わせて作らなければならない。**
- **サブルーチンを他のプログラムで利用しようとしても，行番号を付け直さなければならないケースが多くなる。**
  また行番号を付け直すことは面倒である。

といったさまざまな不便が考えられるからです。さらに悪いことには，せっかくメインルーチンをそのサブルーチンに合わせ，動くプログラムができあがったとしても **renum コマンドによって行番号を付け換えることができません**。X-BASIC は，構造化プログラミングを推奨していますので，renum により goto や gosub の行番号を付け換えることはしません。ゆえに X-BASIC では，非構造化文である

goto

gosub

を使ってはいけないのです。使うと，ひどいめにあいます。またいま述べましたようにプログラム（部品）の再利用が難しくなります。

このプログラムの再利用ということに関しましては，もう１つ

ローカル変数

という大問題があります。それを次のQ36で説明します。

Q36 ローカル変数とは何か？

A36 サブルーチンが壊れた？

前のQ35で作成した

□□□□□□□□ ……〈1〉

を表示するサブルーチンを再び例に使います。

いまこのサブルーチンを使用して〈1〉の図形を

□□□□□□□□
□□□□□□□□
□□□□□□□□
□□□□□□□□
□□□□□□□□

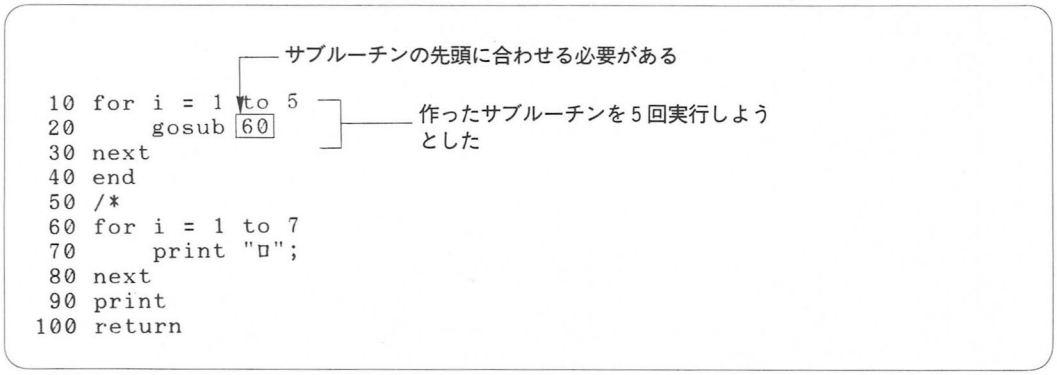
のように5回表示したいとします。普通に考えればfor～nextを用いて

```
for i= 1 to 5
    gosub  60
      (〈1〉を表示するサブルーチンを60行から入力したとして)
next
```

とすればよいわけです【第5-25図】。

ところが実際走らせてみると，第5-26図のように動きません。1回ならうまく動いていたサブルーチンが5回ならダメになってしまったのです。

第5-25図　サブルーチンを呼ぶ



第5-26図　実　　行

```
run
□□□□□□□□          ←―― 1回しか実行されない
Ok
? i
 9                  ←―― iの値を調べてみると9になっている
Ok
```

変数の衝突

理由は，**変数の衝突**です。つまりサブルーチンの中で使用していた変数iと，メインルーチンで使用していたiが干渉しあって，まずい結果を生んでしまったのです。なぜならサブルーチンが1回呼ばれた段階でiは8になっています(iが8になって初めて最後のto 7を抜ける)。次にメインルーチンに戻った時，iの値は30行のnextでさらに1が加えられ9になります。これは，10行のto 5を抜けるには十分ですからここでメインルーチンのfor文は終わってしまいます(もう1度**第5-26図**のiの値を見てください)。

この変数の衝突を避けるためには，メインルーチンとサブルーチンとで使用する変数名を変える必要があります。ですから**第5-27図**のように変更すれば，ちゃんとプログラムは動きます。

第5-27図　変数名を注意して

```
list
   10 for i = 1 to 5
   20     gosub 60
   30 next
   40 end
   50 /*
   60 for j = 1 to 7
   70     print "□";
   80 next
   90 print
  100 return
Ok
run
□□□□□□□
□□□□□□□
□□□□□□□
□□□□□□□
□□□□□□□
Ok
```

メインルーチンとサブルーチンで変数名の衝突をさける

やっと5回実行される

関数にすれば行番号に依存しない部品が作れる

このように従来のBASICでは，サブルーチンを再利用しようとしても

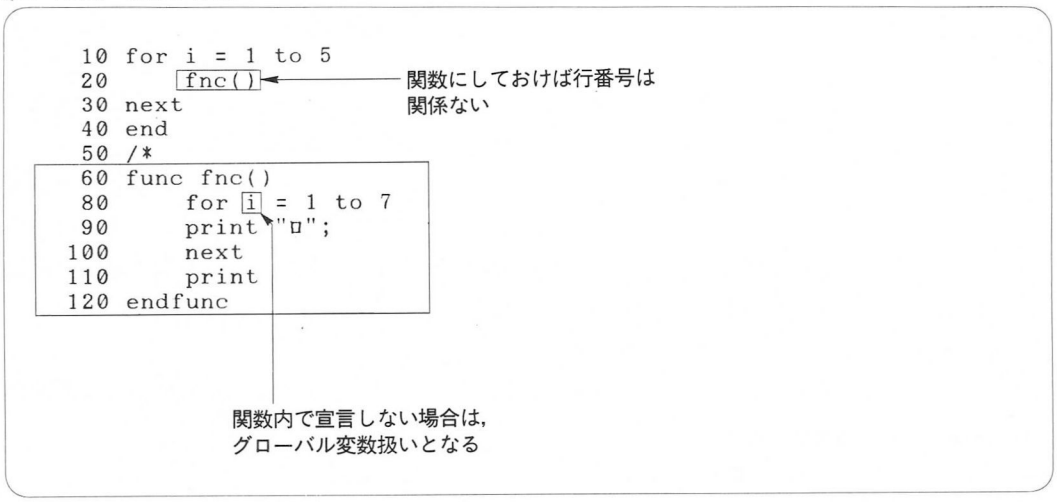
行番号を付け直さなければならない（Q35参照）

変数の衝突に注意しなければならない

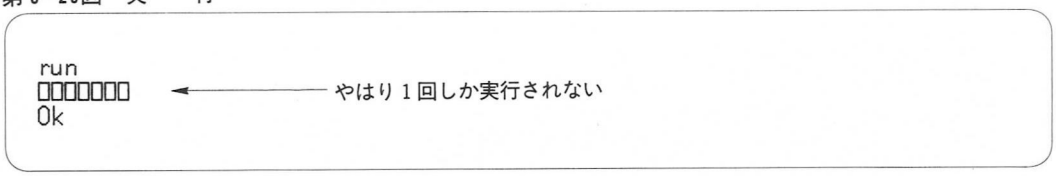
という問題があり，プログラムの蓄積を行いにくい構造になっていました。ところがX-BASICは，関数及びローカル変数が使えますので，これらの問題をすべて回避することができます。つまり，プログラムの再利用が可能な構造を持っているわけです。

そこで，**第5-28図，第5-29図**をご覧ください

第5-28図　関数に変更



第5-29図　実　　行



□□□□□□□ ……〈1〉

を表示する関数を60行～120行で定義しています。関数名をfnc( )にしていますが，もう少し機能を意味するものに命名した方がよいでしょう。いずれにしてもこの関数を呼び出すには，20行のように関数名だけでできることに注意してください。**行番号には依存していません。**

そこで関数版のプログラムを実行してみます。しかし，結果はサブルーチンの場合と同じです。〈1〉は1回しか表示されていません。

グローバル変数

これは，関数内で使用している変数 i が**グローバル変数**のままだからです。関数内で特に宣言しない変数は，メインルーチンと同じ変数と見なされます。ですからメインルーチンとサブルーチンとで共通の変数を使いたい場合は，このようにすればよいのです(何もしなければよい)。このようにプログラムすべての部分で共通に使える変数を

グローバル変数

グローバル変数

といいます。グローバル変数は，従来のサブルーチンと同じような使い方ができるわけです。

ローカル変数

メインルーチンの変数に影響されない関数内部で独立した変数を使用するには，関数の中で新しく変数を宣言する必要があります。

第 5 -30図をご覧ください。関数の中でメインルーチンと同じ名前の変数 i を使っていますが，変数の衝突が起きていません。今度はうまく動いています。このようにメインルーチンには影響を与えない関数内部だけで使用される変数を

ローカル変数

ローカル変数

といいます。ローカル変数を使用することで，メインルーチンには影響さ

第 5 -30図　ローカル変数の宣言

```
list
   10 for i = 1 to 5
   20     fnc()
   30 next
   40 end
   50 /*
   60 func fnc()
   70     int i          ← 変数宣言を入れる
   80     for i = 1 to 7
   90     print "□";
  100     next
  110     print
  120 endfunc
Ok
run
□□□□□□□
□□□□□□□
□□□□□□□   ← 今度は 5 回実行される
□□□□□□□
□□□□□□□
Ok
■
```

れない再利用の可能な関数を作ることができます。

もう1つのローカル変数――仮引数

ローカル変数を定義するもう1つの方法があります。それは，

仮引数

仮引数――関数定義の（）の中に記述し，メインルーチンからの引数を受け取るための変数

を使用することです。仮引数は，自動的にローカル変数になります。

第5-31図の150行～190行をご覧ください。local（）という関数を定義

第5-31図　local. bas

```
 10 /*
 20 /*  ローカル変数
 30 /*
 40      int x = 1, y = 2
 50 /*
 60      print "メインルーチンです"
 70      print "x ="; x, "y ="; y
 80      print
 90      local(10)
100      print
110      print "メインルーチンに戻りました"
120      print "x ="; x, "y ="; y
130      end
140 /*
150 func local(x; int)
160      int y = 20
170      print "関数 local() に入りました"
180      print "x ="; x, "y ="; y
190 endfunc
```

仮引数
関数内で定義された変数　} がローカル変数になる

第5-32図　実　　行

```
run
メインルーチンです
x = 1   y = 2

関数 local() に入りました
x = 10  y = 20

メインルーチンに戻りました
x = 1   y = 2
Ok
```

関数内で値を変更

メインルーチンに戻ると
変数の値は変わっていない

しています。localの中では，

x——150行で定義された仮引数

y——160行で変数宣言された変数

の2つの変数が使われています。これらは，いずれもローカル変数ですのでメインルーチンの変数とは独立しています。

実際，実行してみますと関数内でx，yの値を変更しているにもかかわらず，メインルーチンのx，yの値は変化していません【第5-32図】。

第6章

テキスト画面の制御

Q37　1行の表示文字数を変えるには？

Q38　X-68000のテキスト画面の表示可能行数は？

Q39　テキスト画面を32行×16行モードで使用するには？

Q40　スクロール範囲とは？

Q41　clsとchr$(12)の違いは？

Q42　画面を徐々に暗くするには？

Q43　カーソル位置を変更するには？

Q44　カーソルを消去するには？

Q45　現在のカーソル位置を知るには？

Q46　書式制御を使うには？

Q47　テキスト画面に色を付けるには？

Q48　RGB方式でカラーコードを作るには？

Q37 1行の表示文字数を変えるには?

A37 1行の表示モードを変更するには

X-BASICの**1行の表示文字数**は

1行の表示文字数

> **96文字**（漢字なら48文字）
> **64文字**（漢字なら32文字）

の2つのモードがあります。出荷状態のX-BASICでは，1行**64文字**モードで立ち上がるように設定されています。

起動時にX-BASICの1行の表示文字数を変える方法は，Q3で説明しました。しかし，1度X-BASICが起動した後でも，1行の表示文字数を変えることができます。それには，widthというコマンドを使用します。ダイレクトモードで

widthによるモードの変更

> width　96↵……1行96文字モードに設定する
> width　64↵……1行64文字モードに設定する

のいずれかを入力することで，1行の表示モードを変更することができます。なお，この時，

テキスト画面
グラフィック画面

のいずれもクリアされます。またグラフィック画面の初期化も解除されますので，グラフィック関係の命令を使用する場合は，初期化からやり直す必要があります。

width. bas

widthはコマンドですが，プログラムの中で使用することができます。つまりステートメントのようにして使用することもできます。widthを使ったプログラム例を**第6－1図**に示します。

実行例を**第6－2図，第6－3図**の通りです。画面上部に文字数を数えるための

インジケータ

を表示させるようにしましたので，自分のマシンの能力を試してみてください。1行96文字を表示できるマシンは，そうざらにはありませんので……。

第6－1図　width.bas――1行の表示文字数を変更する

```
 10 /*
 20 /* test <<width>>
 30 /*
 40 print "1 .. 96 文字モード"
 50 print "2 .. 64 文字モード"
 60 print "     画面モードを選んでください .. ";
 70 while 1
 80     switch inkey$
 90         case "1" : mode(96): end
100         case "2" : mode(64): end
110         default  : beep
120     endswitch
130 endwhile
140 /*
150 func mode(m; int)
160     width m                          ―― ここで画面モードを変えている
170     for i = 1 to m                   ┐
180         if (i mod 10) = 0 then {     │
190             print str$(i / 10);      ├ ここでインジケータを表示している
200         } else {                     │
210             print ".";               │
220         }                            ┘
230     next
240 endfunc
```

第6－2図　実行開始

```
run
1 .. 96 文字モード
2 .. 64 文字モード
    画面モードを選んでください ..
```

1か2を選択する
それ以外を押すと，ベルが鳴ってやり直しとなる

第6－3図　2を選択

```
.........1.........2.........3.........4.........5.........6....
Ok
```

画面上端にインジケータが現れる
1行64文字モードになっているのがわかる

Q38 X-68000のテキスト画面の表示可能行数は？

A38

X-68000テキスト画面の表示可能な最大文字数は，横に96文字でした。それでは，縦――つまり**表示可能な最大行数**はいくつでしょう？

それを確かめるために，**第6－4図**のプログラムを作ってみました。

実行させると，**第6－5図**のようにX-68000は，縦に32行表示可能なことがわかります。さらにファンクションキーの内容表示の行を加えれば

最大32行表示可能

なことがわかります。

《補　注》

最終行の第32行は，ファンクションキーの内容表示専用となっています。この行をユーザーが使用することはできません。

第6－4図　isline.bas

```
10 /*
20 /* 画面の行数を調べる
30 /*
35 cls
40 for i = 1 to 31
50   locate 0, i-1
60   print i;
70 next
80 goto 80
```

行数を表示

これは画面をこわさないための手段

第6－5図　X-68000のテキスト表示画面は？

max →

ファンクションキーの表示部分を除いて
31行あるのがわかる

Q39 テキスト画面を32行×16行モードで使用するには？

A39

X-BASICの画面モードは，通常なら

96行×32行モード（1行はファンクションキー用）
64行×32行モード（1行はファンクションキー用）

のいずれかに固定されています。この切り換えは，Q37で説明しましたようにwidthコマンドで行います。ですからwidthコマンドに固執する限り，これ以外の画面モードに設定することはできません。

しかし，好みによっては（たとえばおおざっぱなデザインを取りたいとき），

もう1つのモード　　32行×16行モード

も使うことができます。32行×16行モードに設定するコツは，screenステートメントを使用することです。本来，screenは，グラフィック画面を初

第6－6図　テキスト画面を32行×16行モードに

```
10 /* (32 * 16) mode */
20 screen 0, 2, 1, 1
30 print "(32 * 16) mode に設定しました"
```

第6－7図　32行×16行モード

```
(32 * 16) mode に設定しました
Ok

list
   10 /* (32 * 16) mode */
   20 screen 0, 2, 1, 1
   30 print "(32 * 16) mode に設
定しました"
Ok
```

1行32文字で改行している
漢字は2文字で1字と数える

期設定するためのものです。このscreenを用いてグラフィック画面を

256（ドット）×256（ドット）

モードに設定しますと，自動的にテキスト画面のモードが32行×16行モードになります。たとえば**第6－6図**のプログラムを走らせてみてください。**第6－7図**のようにテキスト画面のモードが32行×16行モードになるでしょう。かなり大きな字で表示されますので，ビックリしないでください。

Q40 スクロール範囲とは？

A40

スクロール範囲

まずは，**第6－8図**のプログラムを走らせてみてください。**第6－9図**のような画面が現れます。この状態で，カーソルを上下に移動させてみてください。第6行～第26行の間しか動けなくなっているのがわかるでしょう。また，文字の入力もこの範囲内しかできなくなります。

このようにカーソルが移動可能な範囲を

スクロール範囲

スクロール範囲

といいます。LISTコマンド等でプログラムのリストを取ると，リストはスクロール範囲にしか現れません。

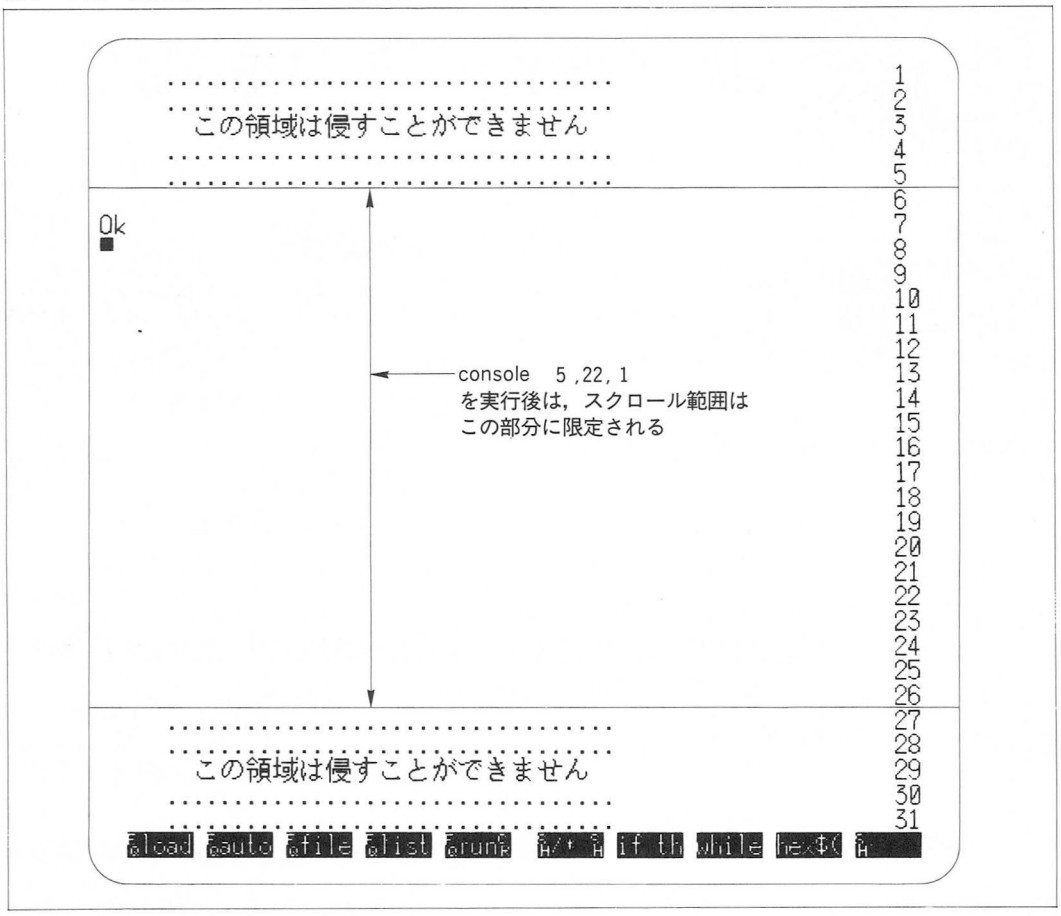
console

スクロール範囲は，ユーザーが自由に変更することができます。予めスクロール範囲を画面全体にしておき，必要な表示を行っておきます。その後，スクロール範囲をせばめることで，**書き換えては困る部分の範囲を保**

第6－8図　console.bas

```
 10 /*
 20 /* test <<console>>
 30 /*
 40 str t
 50 t = string$(5, chr$(&H1C))
 60 cls
 70 console 0,31,1 :cls
 80 for i = 1 to 31
 90   locate 59, i-1
100   print i;
110 next
120 locate 0,0  :msg()
130 locate 0,26 :msg()
140 console 5, 22, 1
150 locate 5, 5
160 end
170 /*
180 func msg()
190     print tab(5); "................................."
200     print tab(5); "................................."
210     print t + "   この領域は侵すことができません"
220     print t + "................................."
230     print t + ".................................";
240 endfunc
```

第6－9図　console　5，22，1を実行



護することができます。

このようなスクロール範囲の変更には，consoleというステートメントを使用します。consoleは，3つの引数を取ります。各引数には

consoleの引数

第1引数――スクロール開始行
第2引数――スクロールを行う行数
第3引数――ファンクションキーの内容表示を行うか否か
（0＝行わない，1＝行う）

を指定します。なお，スクロールの開始行は

1番上の行を第0行

と数えることに注意してください。第3引数は，省略できません。またスクロール範囲は，31行目までで，ファンクションキーの内容表示行まで拡張することはできません。

Q41 clsとchr$(12)の違いは?

A41 clsとchr$(12)

cls

chr$(12)

通常，テキスト画面をクリアするにはclsというステートメントを使用します【第6-10図】。

しかし，Q20で説明したコントロールコードの12を使用し

```
print　chr$(12)
```

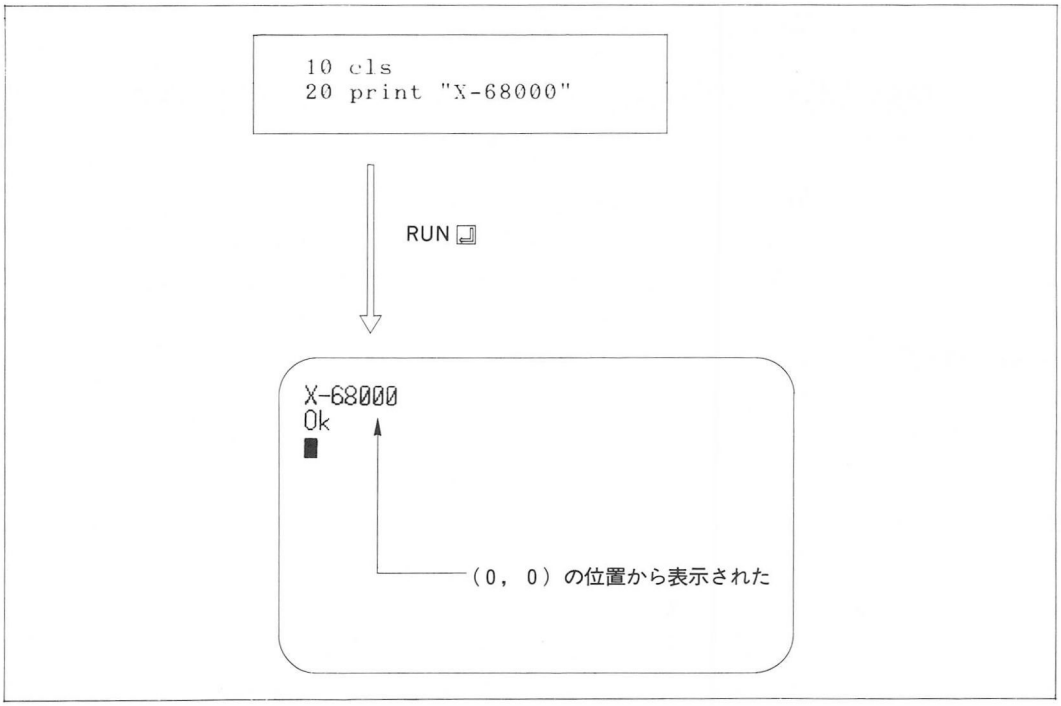
でも画面クリアすることができます【第6-11図】。

ただしこれらの図をよく比較すればわかりますように，

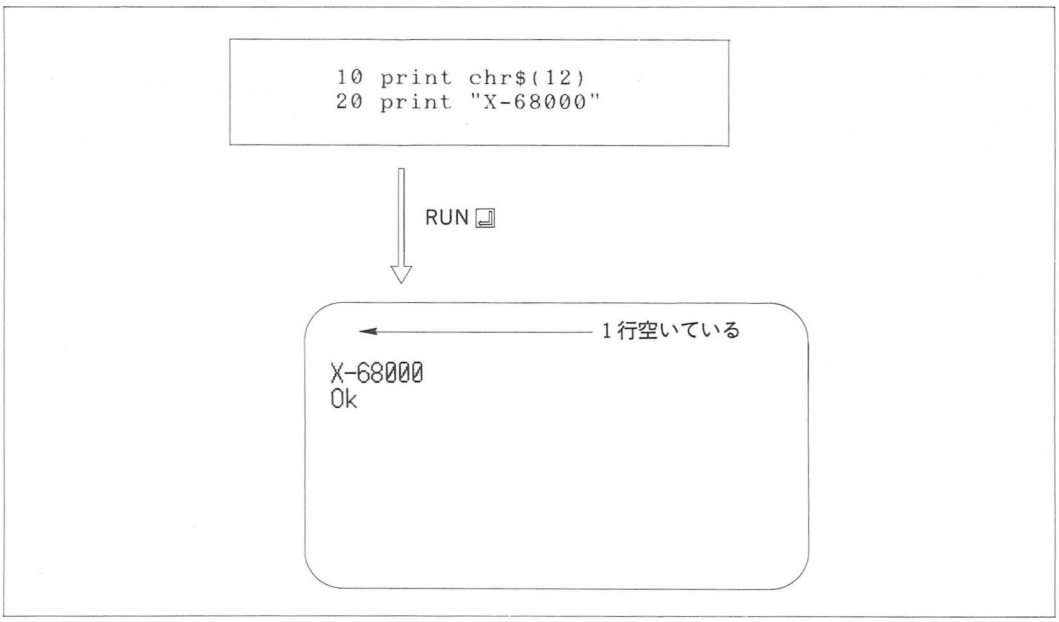
```
cls
print chr$(12)
```

は異なります。すなわちprint chr$(12)で画面をクリアしますと，画面クリア後に1行改行してしまいます。

第6-10図　clsによる画面クリア



第6-11図　chr$(12)による画面クリア

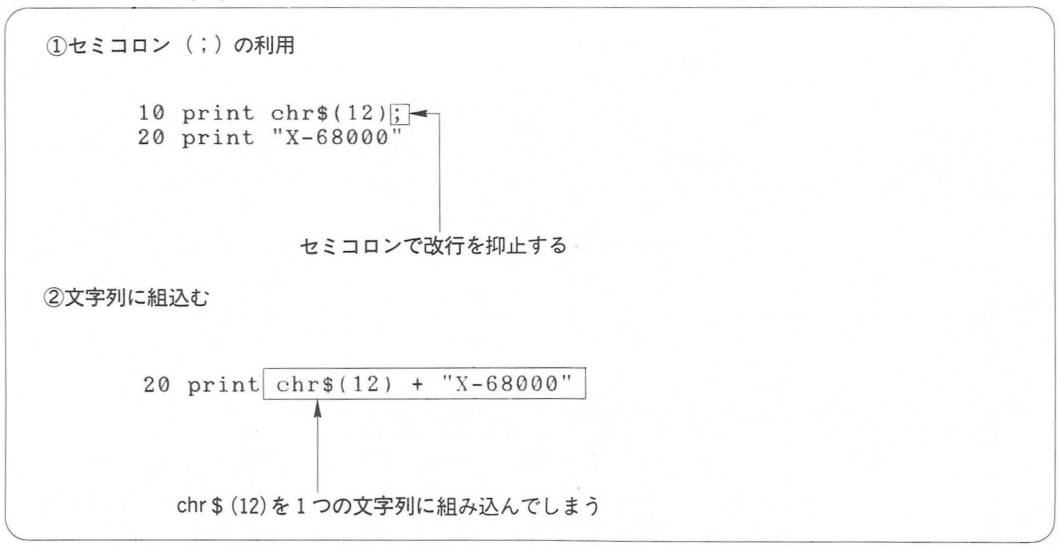


改行を抑止するには?

これは，12というコントロールコードをprint文で出力しているからです。print文は，最後に**セミコロン**；がない限り，改行してしまいます。逆にいえば，chr$(12)を用いて画面クリアする場合，改行させたくなければ最後にセミコロン；を付けてやればよいことになります。実際，

print chr$(12)；

第6-12図　chr$(12)で改行させないために



を実行すると，画面クリア後改行されません【第6-12図】。

なお第6-12図には，もう1つ改行させない方法が示してあります。それは，chr$(12)とそれに続く文字列を

```
chr$(12)＋ "X-68000"
```

のように**1つの文字列に結合してしまう**ことです。こうすればprint文は，この文字列を一挙に表示した後に改行するようにします。

Q42 画面を徐々に暗くするには?

A42 コントラストを変えるには?

X-68000のサブ電源を落としますと，**画面が徐々に暗く**なっていきます。このようにX-68000は，ディスプレイ画面の

コントラスト

コントラスト

を変えることができます。その制御は，もちろんX-BASICの上でできます。

X-BASICを使って画面のコントラストを変えるには

contrast( )

という関係を使います。contrast( )はGRAPH関係の登録関数ですから，あらかじめBASIC．CNFの中で

FUNC=GRAPH

のように登録しておく必要があります（通常はそのようになっています。Q24を参照）。

contrast( ) の書式は，

contrast

```
contrast(4)
```

のように( )の中に明るさを指定します。明るさは

0 （暗い）～15 （明るい）

の16段階が指定できます。0が最も暗く，数字を大きくするほど明るくなります。

サンプル

第6-13図にcontrast( )を使ったサンプルプログラムを示します。80行以下にコントラストを変更する関係contrast( ) を定義しています。この関係を

contrast(”暗い”)；

のように呼ぶと，画面がだんだんと暗くなります。また

contrast(”明るい”)；

のように呼ぶと，画面がだんだん明るくなります。

第6-13図　contrast.c

```
 10 /*
 20 /* test <<contrast>>
 30 /*
 40 contrst("暗い")
 50 contrst("明るい")
 60 end
 70 /*
 80 func contrst(s; str)
 90     if s = "暗い" then {
100         for i=0 to 15
120             contrast(15 - i)
130             for j = 0 to 400: next
140         next
150     } else if s = "明るい" then {
200         for i=0 to 15
210             contrast(i)
220             for j = 0 to 400: next
230         next
240     }
250 endfunc
```

"暗い"が指定されたら
だんだん暗くする

"明るい"が指定されたら
だんだん明るくする

Q43 カーソル位置を変更するには?

A43 カーソル位置を変更する——locate

テキスト画面に出力する文字は，**カーソル位置**から表示されていきます。ですからこのカーソル位置を自由に変更できれば，面白いことができそうです。

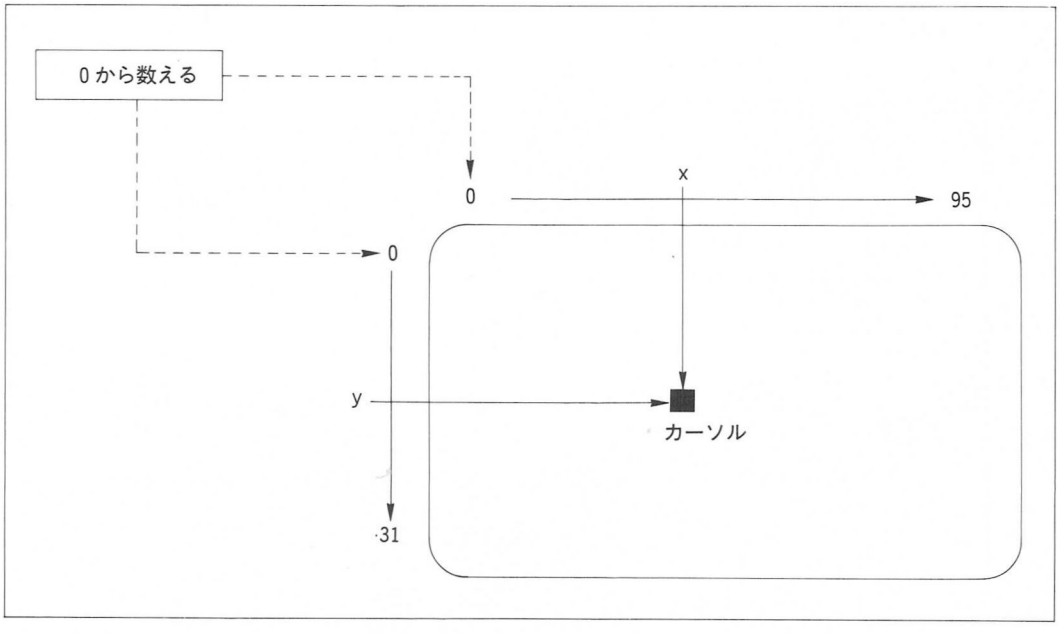
X-BASICでカーソル位置を変更するには，locateというステートメントを使います。locateは，

locate

```
locate  10, 20
        ↑   ↑
        |   |
        |   これが新しいカーソル位置のｙ座標
        |
        これが新しいカーソル位置のｘ座標
```

といった使い方をします。なおこの座標は，先頭を0と数えますので注意してください【第6-14図】。

第6-14図　X-68000のテキスト画面



サンプルプログラム

locateステートメントを使ったサンプルプログラムを**第6-15図**に示します。このプログラムを走らせますと，《X-68000》の文字が点滅します**【第6-16図】**。その仕組みは，

ブリンクのしくみ

座標（10，5）に《X-68000》を書く
↓
座標（10，5）にスペースを7つ書く

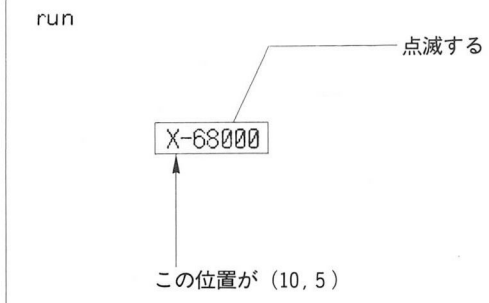
といったことを繰り返しているのです。スペースを7つ書くことで，《X-68000》の表示を消すことができます。

プログラムでは，座標（10，5）を指定するのにlocateステートメントを使用しています。

第6-15図　locate.c

```
 10 /*
 20 /* test <<locate>>
 30 /*
 70 while 1
 80      loc("X-68000", 800)      ――表示する
 90      loc("       ", 400)      ――消去する　} 交互に繰り返してブリンクさせる
110 endwhile
120 /*
130 func loc(s; str, t; int)
140      locate 10, 5: print s;
150      for i = 0 to t:next
160 endfunc
```

第6-16図　"X-68000"をブリンクさせる



Q44 カーソルを消去するには?

A44

通常，カーソルは，次に入力される位置の目安に使われます。しかし，ゲームを作っている時等，点滅するカーソルを消したい場合もあります。

もし必要なら**カーソルの表示を止める**ことができます。それにもlocateステートメントを使用します。普通locateは2つの引数でカーソルの座標を指定しますが(Q43参照)，もう1つ3番目の引数を指定することができます。この引数がカーソルの表示を制御するのに使われます。

第3引数

1——カーソルを表示する
0——カーソルを表示しない

カーソルの表示を制御する場合，locateの第1，第2を省略して

```
locate ,, 0
          └──────この場合は，カーソルを消す
```

のように使います。もちろん一緒にカーソル位置を指定することもできます。

Q45 現在のカーソル位置を知るには？

A45

システム変数を利用してカーソル位置を知る

locateステートメントを使うとき，**現在のカーソル位置**を知りたい場合があります。もちろんX-68000の中で現在のカーソルを知ることができます。それには，

カーソル位置を保持するシステム変数

pos ――― x座標
csrlin ――― y座標

という2つの**システム変数**を使用します。

システム変数というのは，文字通り変数ですが，ユーザーが勝手に使うことはできません。X-BASICインタプリタが特別な目的で使用します。たとえばposというシステム変数には，現在のカーソルのx座標が自動的に入るようになっています。ゆえにX-68000の中でカーソルのx座標が知りたい場合は，システム変数posの値を調べればよいのです。同様にy座標の方はシステム変数csrlinを調べればわかります。

サンプルプログラム

pos，csrlinを使用したサンプルプログラムを**第6-17図**に示します。これは，**第6-18図**のように画面上の

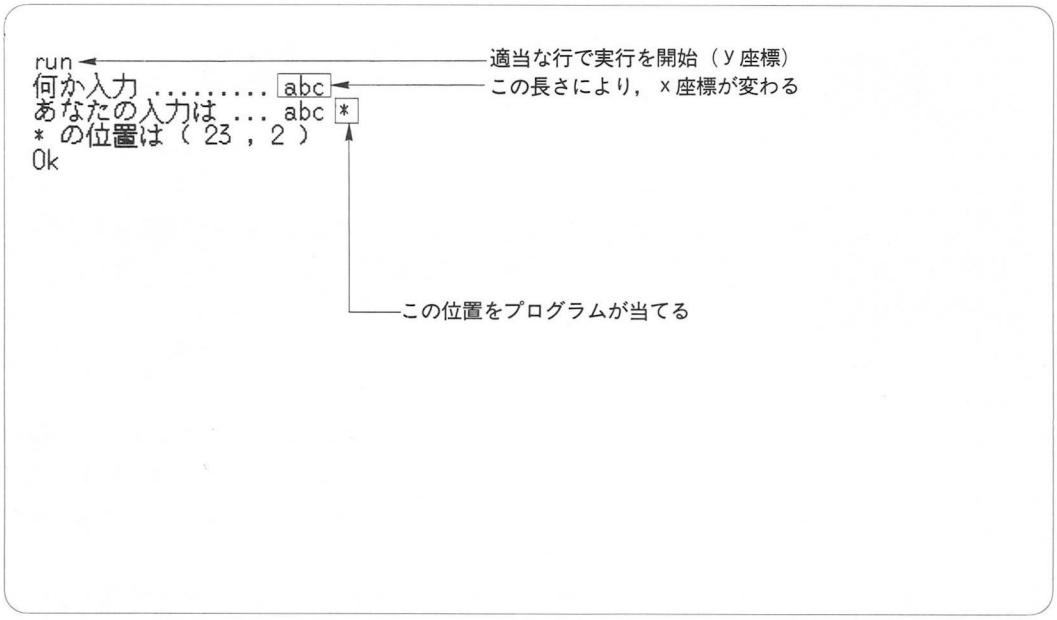
第6-17図　csrlin.c

```
10 /*
20 /* test <<pos, csrlin>>
30 /*
40 int x, y
50 str s
60 input "何か入力 ..... ", s
65 print "あなたの入力は ... "; s; " *";
70 x = pos - 1
80 y = csrlin
90 print: print "* の位置は ("; x; ","; y; ")"
```

ここで改行させてはいけない

すぐにカーソル位置を記憶させてしまう

第6-18図　カーソル位置を当てる



＊

の位置をプログラムが当てるものです。＊の位置は，ユーザーの入力により変化します。なお70行で，pos の値を－１しているのは，＊の位置がカーソル位置よりも１文字左にあるからです。

Q46 書式制御を使うには？

A46

テキスト画面に数や文字を表示するには，print 文を使用します。そのまま print 文を使用しますと，カーソル位置から左詰めに表示されますが，

- **表示桁数を決め，その中で右詰めに表示したい**
- **数の中に，（カンマ）を埋め込みたい**

といった書式制御を行いたい時があります。そのような時は，

using　　print using

により書式制御文字列を使用するのが簡単です。書式制御文字列は，最初はとっつきにくいのですが，慣れれば思い通り凝った表示をさせることができます。**第 6 -19図～第 6 -22図**のサンプルを参考にその表示効果を確かめてください。

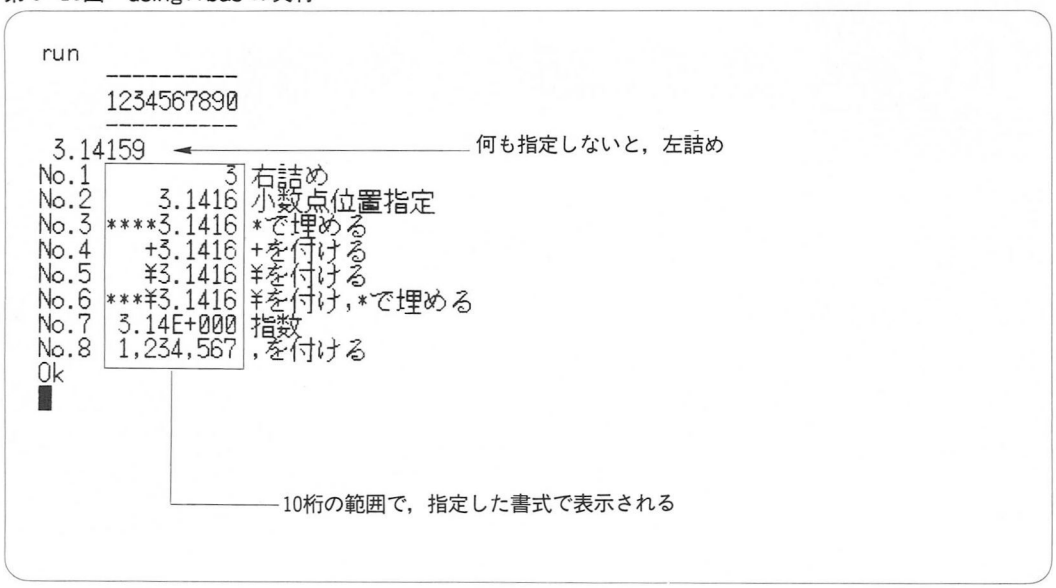
注意することは，数値用と文字列用の 2 種類の書式制御を間違えないことです。これらすべてを必要とすることはないでしょうから，気に入ったものを少しずつ身に付けていけばよいでしょう。

第 6 -19図　数値用書式制御――using1.bas

```
10 /*
20 /* print using num type
30 /*
40 float n = 3.14159#
50 print tab(5);"----------"
60 print tab(5);"1234567890"
70 print tab(5);"----------"
80 print n
90 print using "No.1 ##########  右詰め"; n
100 print using "No.2 #####.####  小数点位置指定"; n
110 print using "No.3 *****.####  _*で埋める"; n
120 print using "No.4 +#####.####  _+を付ける"; n
130 print using "No.5 ¥¥###.####  _¥を付ける"; n
140 print using "No.6 **¥##.####  _¥を付け,_*で埋める"; n
150 print using "No.7 ##.##^^^^  指数"; n
160 print using "No.8 #########,  _,を付ける"; 1234567
```

この部分が書式制御文字列
10桁に揃えてある

第 6 -20図　using1.bas の実行

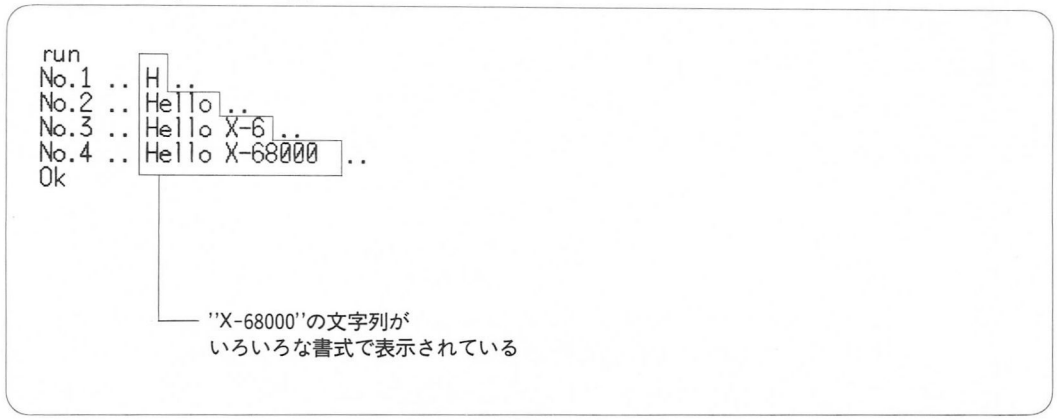


第 6 -21図　文字列用書式制御——using2.bas

```
10 /*
20 /* print using string type
30 /*
40 str s = "Hello X-68000"
50 print using "No.1 .. ! ..."; s
60 print using "No.2 .. &   & ..."; s
70 print using "No.3 .. &       & ..."; s
80 print using "No.4 .. &           & ..."; s
```

この部分が書式制御文字列

第 6 -22図　using2.bas の実行



Q47 テキスト画面に色を付けるには？

A47

パレットとパレット番号

X-BASICで色を扱うには，**パレット**という概念を理解する必要があります。ただし，最初はややっこしいので，以下の手順でその効果を少しずつ理解していってください。

パレット

パレットは，絵皿です。X-BASICのテキスト画面は，4つのパレットが使えます。ですから同時に4色の表示を行うことができます（グラフィック画面なら最大256種類のパレットを使用することができます）。

テキスト画面の4つのパレットには，それぞれ

0～3

の**パレット番号**が付いています。そして，最初は

パレット0——黒（背景と同じなので見えない）
パレット1——シアン（水色）
パレット2——黄色
パレット3——白

の色が使えるようになっています。

テキスト画面の表示色を変える——color

テキスト画面でこれらの色を使いたい場合は，colorというステートメントを使います。その書式は，

color

```
color〈属性〉
```

です。**属性**というのは，0～15までの数です。0～3については，まさにいま説明したパレット番号が対応しています。たとえば

```
color　2
```

とすれば，テキスト画面の表示色がパレット2に設定されます。パレット2は黄色ですから，これ以後print文で表示する文字は，すべて黄色で表示されます。ダイレクトモードで

```
color　1：print　"水色"↵
color　2：print　"黄色"↵
```

のように入力してみるとその効果がわかるでしょう。

その他の属性

それでは，残りの4～15の属性はどうなっているのかというと，パレットの標準値に種々の味付けをするようになっています。

その他の属性

パレット4――パレット0の強調
パレット5――パレット1の強調
パレット6――パレット2の強調
パレット7――パレット3の強調
パレット8――パレット0のリバース
パレット9――パレット1のリバース
パレット10――パレット2のリバース
パレット11――パレット3のリバース
パレット12――パレット0の強調リバース
パレット13――パレット1の強調リバース
パレット14――パレット2の強調リバース
パレット15――パレット3の強調リバース

たとえば

```
color　10
```

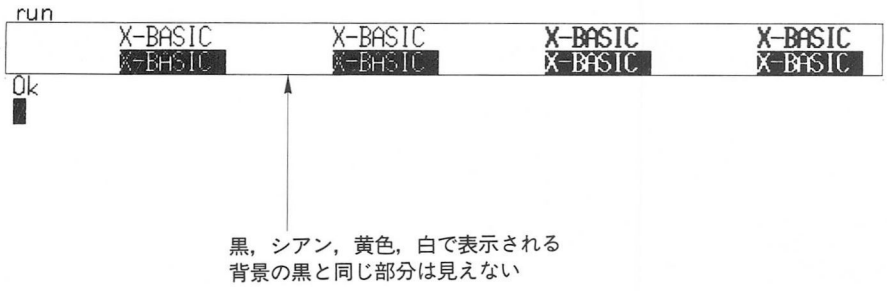
としますと，2（黄色）のリバース（反転）になりますので，以後print文で表示する文字は黄色の反転文字になります。実際の効果を見るために**第6-23図**のプログラムを作成しました。走らせて確かめてください**【第6-24図】**。

第6-23図　color.bas

```
10 /*
20 /* color.bas */
30 /*
40 for i = 0 to 15
50     color i
60     print "X-BASIC",
70 next
80 color 3
```

標準のパレットコード0～15で"X-BASIC"を表示する

第 6 -24図　color.bas の実行



（注）この画面は 1 行64文字モードでのものです

Q48 RGB方式でカラーコードを作るには?

A48

カラーコードを作る

Q47でパレットとその表現方法であるcolorステートメントの説明をしましたが，このままでは**シアン**と**黄色**の2色しか使えません。もっと他の色を使うには，自分で好みの色——すなわち**カラーコード**を作り，それをパレットに対応付ける必要があります。カラーコードは，

0～65535

までの数です。つまりX-68000は65,536色の中から任意の色を使うことができるわけです。

カラーコードの作り方は，X-BASICのマニュアルのP.28および本書の姉妹書である『X-68000活用研究』の第1巻に出ています。しかし，これらを見ますと少々面倒そうです。そこで，このQではもう少しわかりやすい**RGB方式**によるカラーコードの作り方を説明しましょう。

RGB方式

RGB方式というのは，**光の3原色**である

光の3原色

r——red
g——green
b——blue

の3色を混合することにより，任意の色を作りだそうというものです。そのためX-68000には

rgb( )

rgb（**第1引数，第2引数，第3引数**）

という登録関係（GRAPHで登録する）が用意されています。

rgb( )は，3つの引数を指定するようになっています。各引数には

第1引数——redの強さを0～31で指定する
第2引数——greenの強さを0～31で指定する

第 3 引数――blue の強さを 0 ～31で指定する

をそれぞれ指定するようにします。たとえば赤のカラーコードを作りたければ

```
c =rgb (31,  0,  0)
          |   |   |
          |   |   +---- blue は混ぜない
          |   +-------- grren も混ぜない
          +------------ red は最大の31混ぜる
```

のようにすればｃに赤のカラーコードが得られます。また黄色のカラーコードを作りたければ，赤と緑を同じ割合で混ぜればよいので，

```
c =rgb (31, 31,  0)
          |   |   |
          |   |   +---- blue は混ぜない
          |   +-------- grren は最大の31混ぜる
          +------------ red も最大の31混ぜる
```

のようにすればよいのです。その他，これら 3 色を適当な割合で混ぜることにより，いろいろな微妙な色を作り出すことができます。

パレットにカラーコードを割り当てる

カラーコードを作っても，それを**パレット**に割り当てなければその色を表示することができません。そのためには

color [ ]

color［第 1 引数，第 2 引数，第 3 引数，第 4 引数］

というステートメントを使います。この color [ ] は，前出の color とは，異なることに注意してください。X-BASIC は，color の後に[ ]が付くかどうかで両者を区別しています。

color [ ] の 4 つの引数には，それぞれ

第 1 引数――パレット 1 に割り当てるカラーコード
第 2 引数――パレット 2 に割り当てるカラーコード
第 3 引数――パレット 3 に割り当てるカラーコード
第 4 引数――パレット 4 に割り当てるカラーコード

を指定するようにします。

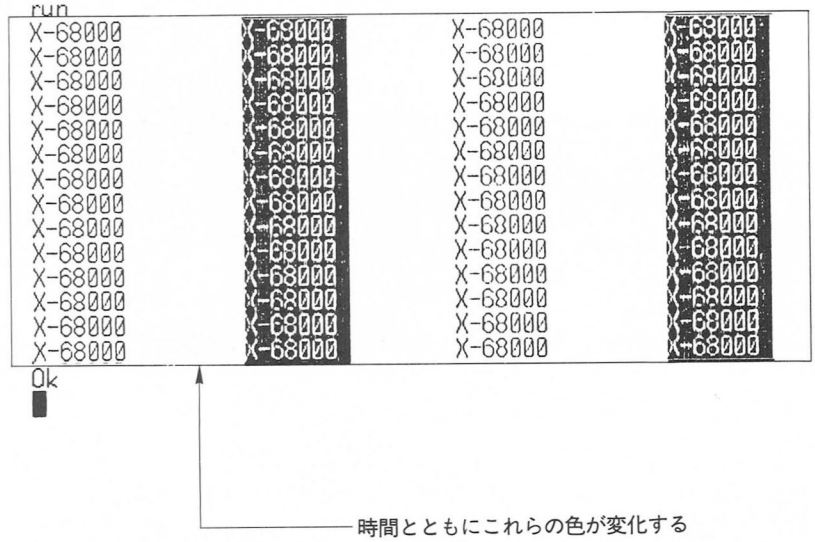
サンプルを**第 6 -25図，第 6 -26図**に示しますので，参考にしてください。

第6-25図　rgb.bas

```
 10 /*
 20 /* rgb.bas */
 30 /*
 40      black  = rgb( 0,  0,  0)
 50      blue   = rgb( 0,  0, 31)
 60      red    = rgb(31,  0,  0)
 70      purple = rgb(31,  0, 31)
 80      green  = rgb( 0, 31,  0)
 90      cyan   = rgb( 0, 31, 31)
100      yellow = rgb(31, 31,  0)
110      white  = rgb(31, 31, 31)
120      for i = 0 to 27
130          color  1: print "X-68000",
140          color  2: print "X-68000",
142          color  9: print "X-68000",
144          color 10: print "X-68000",
150      next
160      colpr(blue, red)
170      colpr(purple, green)
180      colpr(cyan, blue)
200      color 1
210      end
220 /*
230 func colpr(p1, p2)
240      color [black, p1, p2, yellow]
250      for i = 0 to 10000: next
260 endfunc
```

rgb( )を用いて各色を作るには
このように行えばよい

第6-26図　rgb.bas の実行



第 7 章

ディスクアクセス

Q49　ディスクファイルにアクセスする手順は？

Q50　具体的なディスクアクセスの方法は？

Q51　error off は何のために使う？

Q52　"w"モードと"c"モードの違いは？

Q53　既存ファイルの保護を考慮するには？

Q54　fseek( )は，何に使う？

Q55　ディスクの残り容量を調べるには？

Q49 ディスクファイルにアクセスする手順は？

A49

ディスクアクセスの手順

ディスクファイルにアクセスする手順は決まっていて，**第7－1図**のようになっています。<1>でファイルを**オープン**し，<2>でファイルに実際に**アクセス**します。そして，最後の<3>で使用したファイルを**クローズ**しておしまいです。

各手順の詳細は，次の通りです。

ファイルのオープン

ディスクファイルを扱うには，まず最初にそのアクセスしたいファイルを**オープン**しなければなりません。ファイルのオープンは

fopen（ ）

```
fopen( )
```

関数で行います。この時必要な引数は

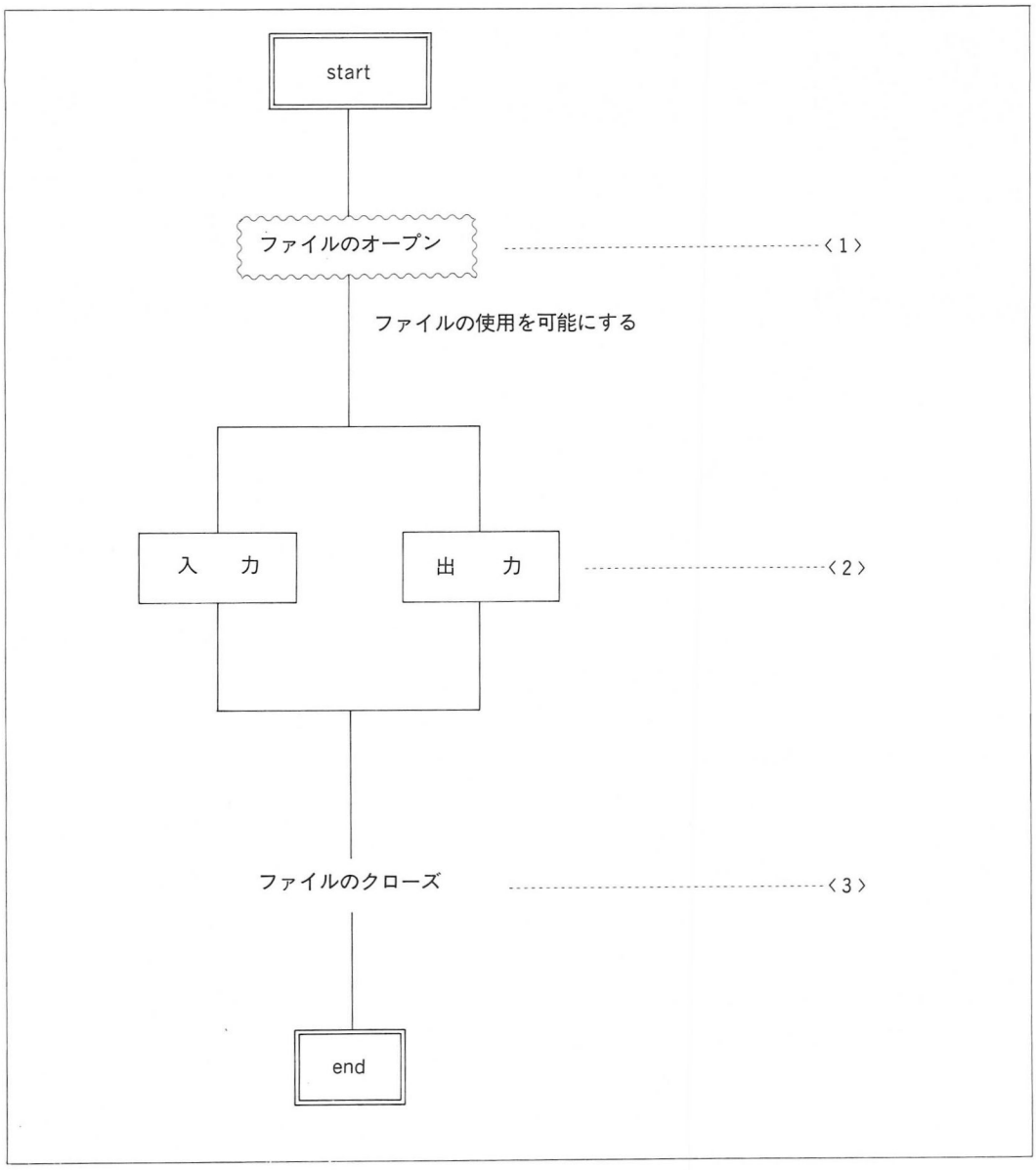
ファイル名
オープンモード

の2つです。**オープンモード**は，そのファイルを何の目的で使用するのかを示すもので，次の4つから選択します。

オープンモード

```
r(read)
    ：ファイルの内容を読む目的でオープンする
w(write)
    ：ファイルに上書きする目的でオープンする
rw(read／write)
    ：ファイルを読み書き両用の目的でオープンする
c(create)
    ：ファイルを新規作成する
    ：作成されたファイルは読み書き両用でオープンされる
```

第7－1図　ファイルアクセスの手順



ファイルのアクセス

オープンされたファイルは，**アクセス**することができます。アクセスとは，ファイルの内容を読んだり，ファイルに新しい内容を書き込んだりすることです。もちろんオープンモードによっては，そのいずれか一方しかできません。

ファイルのアクセスには，次の3つの方法があります。

ファイルアクセス関数

●1文字単位での読み書き
　fputc( )，fgetc( )
●1行単位での読み書き
　freads( )，fwrites( )
●数値型1次元配列の読み書き
　fread( )，fwrite( )

ファイルのクローズ

アクセスが終わったファイルは，最後に**クローズ**します。ファイルのクローズは，

クローズ関数

fclose( )
　：オープンされているファイルを1つずつクローズする
fcloseall( )
　：オープンされているファイルをまとめてクローズする

のいずれかの関数を使用します。

その他ディスクアクセスで必要な命令

以上の3つの手順を通すことによりディスクファイルを操作することができます。実際は，この他にも

その他の関数・ステートメント

err　off
　：外部関数エラーが発生してもプログラムを終了
　　させないためのステートメント
fseek( )
　：ファイルアクセスの位置を変更する関数
feof( )
　：ファイルの終わりをキャッチする関数

といった補助的な命令を使ってプログラムを組んでいきます。具体的な手順は，次のQ50以下で説明します。

Q50 具体的なディスクアクセスの方法は?

A50

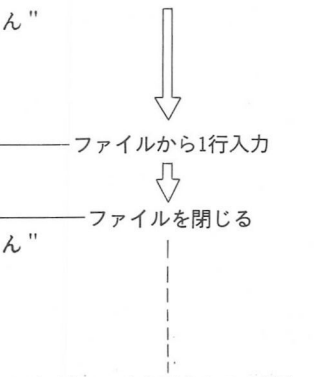
サンプル——type. bas

ディスクアクセスに関する概要は，前のQ49で説明しました。ここでは，もう少し具体的にサンプルプログラムを示し，実際にディスクファイルにアクセスしてみます。

例として**第7－2図**のプログラムを見てください。

第7－2図　type.bas

```
 10 /*-------------------------------------------
 20 /* basic のプログラムを画面に表示する
 30 /*      ファイル名の入力で .bas は省略する
 40 /*                                  87.10.21
 50 /*-------------------------------------------
 60 error off
 70 str fn, buf[255]
 80 input "ファイル名"; fn
 90 fp = fopen(fn + ".bas", "r")
100 if fp = -1 then {
110     print "ファイルがオープンできません"
120     end
130 }
140 print
150 while not feof(fp)
160     freads(buf, fp)
170     print buf
180 endwhile
190 if fclose(fp) then {
200     print "ファイルがクローズできません"
210     end
220 }
```



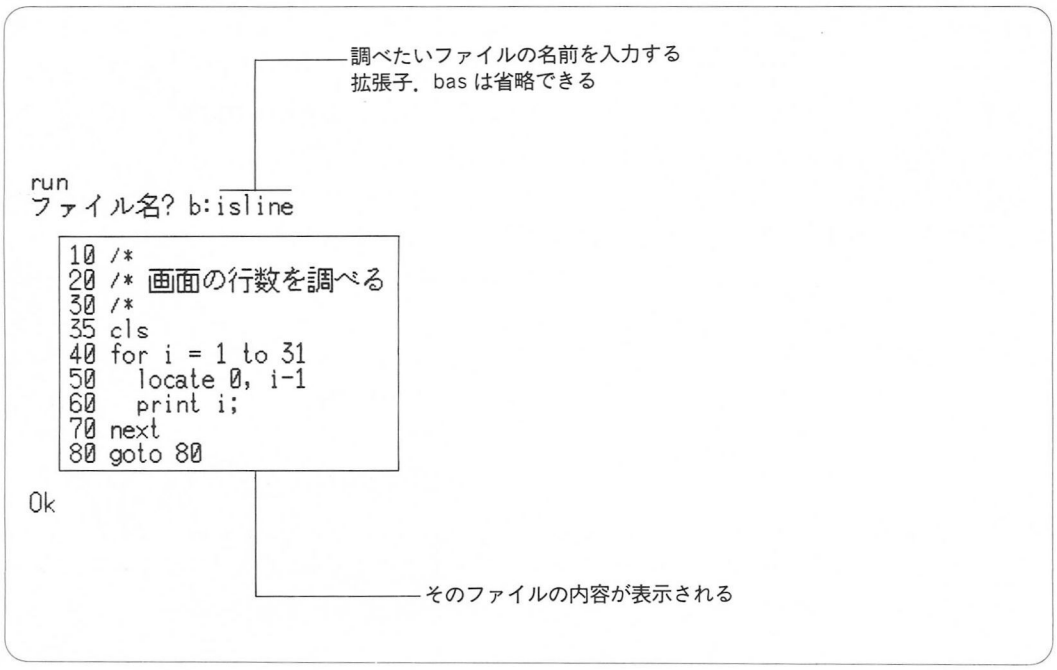
これは，ディスク上にファイルされているX-BASICのプログラムの内容を表示するものです。プログラムがたくさん増えてきますと，ファイル名だけでは何のプログラムかわからないときがあります。そのようなとき，通常はそのファイルをロードし，listを取ることによりその内容を調べます。しかし，そのたびにファイルをロードしていては，面倒です。そのようなとき，このプログラムが役立ちます。

実行例を**第7－3図**に示します。

run ↵

しますとファイル名を聞いてきます。内容を調べたいファイルの名前を入力します（このとき，拡張子．basの入力は不要です）。これだけの操作で，そのファイルの内容を読むことができます。

第7－3図　実　　行



ファイル番号

60行の err off については，次のQ51で説明します。

このプログラムのポイントは，ディスクファイルにアクセスし，その内容を読むことです。ですから，最初はファイルの**オープン**からスタートします。そのため，読みたいファイルの名前を60行の input 文により，外部（キーボード）から入力させています。入力したファイル名は，変数 fn に格納されます。実際のファイル名は，それに拡張子．bas を付けた

fn＋”．bas”

となります。オープンモードは，ファイルの内容を読むわけですから

”r”――read モード

です。よって，ファイルオープンの部分は

fopen（fn＋”．bas”，”r”）

ファイル番号

となります。fopen( )を実行しますと，**ファイル番号**とよばれる整数値が返されます。そこで，そのファイル番号を

fp＝fopen（fn＋”．bas”，”r”）；

のように適当な整数型変数に受け取るようにします（行番号90）。

以後，ファイルのアクセスはこのファイル番号 fp によって行います。

オープンエラーへの考慮

実際は，ファイルのオープンに失敗することもあります。たとえばこの場合でしたら，指定したファイルが存在しない場合にエラーとなります。存在しないファイルの内容を読むことはできません。fopen( )は，オープンに失敗しますと，－1を返してきます。ですから

オープンエラーの処理

```
if fp＝－1　then　{
    オープンエラーの場合の処理
}
```

により，オープンエラーが発生した場合の処理ができます(100行～130行)。

ファイルの読み込み

次にファイルのアクセスです。このプログラムでは，ファイルの内容を読んでは表示すればよいので，行単位のアクセス関数

```
fread (buf, fp)；
```

を使用します。この fp は，オープン時に得られたファイル番号です。この fp が指すファイルから１行読み込み，読み込みバッファ buf に格納してくれます。バッファ buf は，実際は文字列変数で

```
str buf；
```

のように宣言しておきます。ただし，ここでは32文字を超える文字列を読み込む可能性がありますので

バッファの宣言

```
str buf [255]；
```

のように，文字列の大きさも宣言しておきます(行番号70)。[255]の部分が，文字列の大きさを表しています。この場合，文字列変数 buf には，最大255文字まで格納することができます。

fread( )の読み込みに成功すれば，その内容は buf に格納されていますので

```
print buf；
```

でディスプレイに表示することができます。

ファイルの終わりを知るには？

まとめますと，ファイルの読み込み／表示は，

```
fread (buf, fp)；
print buf；
```

のように行えばよいわけです。これが1行分の処理に当たります。後は，この処理をファイルの終わりまで繰り返せばよいのです。それでは，その

ファイルの終わり

をどのように知ればよいのでしょう？　fread( )は，

読み込みに失敗した
ファイルの終わりに達した

のいずれかに遭遇しますと－1を返します。ですからfread( )の結果を調べ，それが－1であればファイルの終わりに達したと考えていいでしょう。ただし，もしかしたら読み込みエラーが発生している可能性もありますが，実用上はそれほど問題ないでしょう。

もし完全にファイルの終わりに達しているかどうかを知りたいなら

feof ( )

```
feof (fp)；
      └──この場合ファイル番号fpのファイルについて調べる
```

という関数を使います。feof( )は，まだファイルの終わりに達していなければ**0**を返します。0は，X-BASICでは偽に相当しますので

```
while not feof (fp)
    .....
    .....    ファイルの読み込みを行う
    .....
endwhile
```

の書式でファイルの終わりまでの処理を行うことができます(150行～180行)。

ファイルのクローズ

ここまでわかれば，最後のファイルのクローズは簡単でしょう。

```
fclose (fp)
       └──これがクローズするファイルのファイル番号
```

のように処理します（190行～220行)。

Q51 error off は何のために使う？

A51

導　入

error off

Q50の**第7－2図**のプログラムでは，最初に

error off

というステートメントが使われていました【**第7－4図**にその部分を再掲】。

error off を正確にいいますと，**error** がステートメントで **off** はその引数です。X-BASIC のマニュアルを見ますと，error は

「外部関数でエラーが発生……」

と書かれています。この意味，わかりましたでしょうか？　**第7－4図**のプログラムで，error off がないと何かまずい点があるのでしょうか？

第7－4図　error off を削除すると

この行がなければどうなるか？

```
 10 /*-------------------------------------------
 20 /* basic のプログラムを画面に表示する
 30 /*      ファイル名の入力で .bas は省略する
 40 /*                                  87.10.21
 50 /*-------------------------------------------
 60 error off
 70 str fn, buf[255]
 80 input "ファイル名"; fn
 90 fp = fopen(fn + ".bas", "r")
100 if fp = -1 then {
110     print "ファイルがオープンできません"
120     end
130 }
```

error off がないと

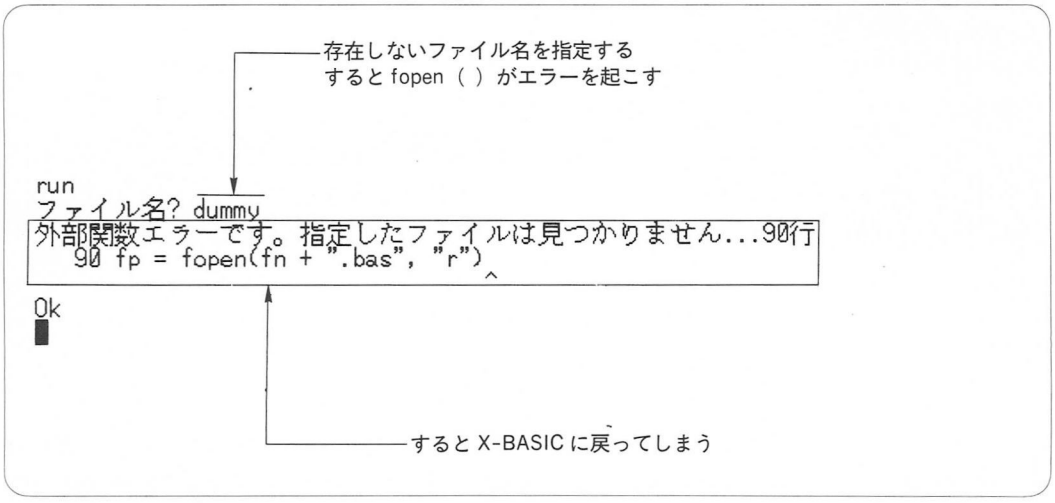
試しにこのプログラムの60行を削除して，**error off を削除**してプログラムを実行してみます。そして，わざとエラーを発生させるため，最初のファイル名の入力のところで**存在しないファイル名**を入力してみます。こうすれば，もちろん90行のところでオープンエラーが発生するはずです。そして，110行により

ファイルがオープンできません

のメッセージが現れるはずです。ところが実際走らせてみますと，fopen

( )がエラーを起こしたところでX-BASICに戻ってしまいます。せっかく用意した110行～120行の部分は，実行されません【第7－5図】。

第7－5図　わざとエラーを起こす



エラー処理ルーチンを可能にするために

X-BASICの関数の仕様はそうなっています。つまりエラーが発生したら，プログラムの実行を打ち切ります。そして，

X-BASICに戻ってしまう

のです。ですからエラー発生時に，プログラムの方でその面倒を見ることはできません。

しかし，予め**error offを実行しておけば**――この事態は改善されます。エラーが発生してもX-BASICには，戻りません。そして，関数は予め決められたエラーを表す値を返します。プログラマは，この値によりエラー処理ルーチン（この場合なら110行～120行）を書くことができるのです。

このQの最初の仮定であるerror offを削除すると――もちろん**まずい**のです。

Q52 "w"モードと"c"モードの違いは?

A52

Q49で説明しましたようにディスクファイルにアクセスするには，まずそのファイルをオープンしなければなりません。ところで，ファイルにデータを書き込む場合，ファイルを**書き込みモード**でオープンする必要があります。その場合，モードの選び方は

"w"**モード**でオープンする
"c"**モード**でオープンする

の2つの方法があります（外にも"rw"モードがありますが，これは書き込みという動作から見ますと，実質"w"モードと同じ機能と考えることができます)。もちろんマニュアル上は，

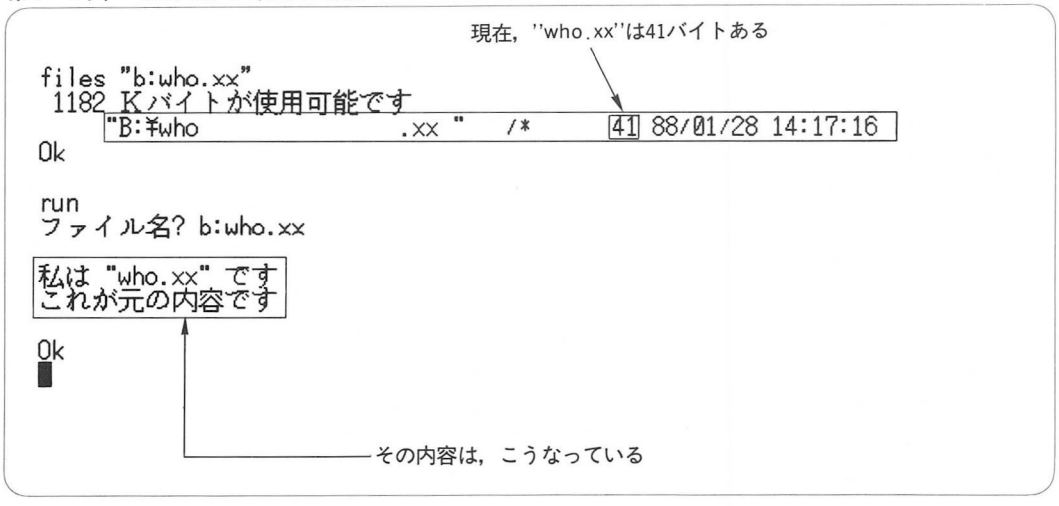
"w"モード——既存ファイルに書き込みを行う
"c"モード——新規ファイルに書き込みを行う

のように明確に分かれています。このことは頭ではわかるのですが，具体的にはどのような違いがあるのでしょう？

"w"モードの動作

そこで，まず**第7－6図**をご覧ください。ここに"who．xx"というファイルがあります。その大きさは，41バイトで，内容はご覧の通りです。

第7－6図 "who.xx"の最初の状態



このファイルに対し，まず"w"**モード**でデータを書き込んでみます。**第7－7図**のようなきわめてシンプルなプログラムを作り，実行してみます。結果は，**第7－8図**の通りです。ファイルの大きさが41バイトで，実行前と変わっていないのがわかるでしょう。これは，"w"モードが

元のファイルに対し，新しい内容を上書きする方法で書き込む

からです。書き込まれない部分は，元の内容がそのまま残っています。ですから前のファイルよりも小さな内容を書き込んだ場合は，ファイルの大きさは変わりません**【第7－9図】**。

第7－7図　"w"モードで書き込む

```
10 int fp
20 fp = fopen("b:who.xx", "w")
30 fwrites("上書きします", fp)
40 fclose(fp)
```

"who.xx"にこの文字列を書き込む

第7－8図　実　　行

```
run
Ok

files "b:who.xx"
 1182 Kバイトが使用可能です
    "B:¥who                .xx "    /*        41 88/01/28 14:22:44
Ok
```

実行して書き込む

しかし，ファイルの大きさは変わらない

第7－9図　もう1度"who.xx"の内容を見る

```
run
ファイル名? b:who.xx
上書きします" です
これが元の内容です

Ok
```

たしかに上書きされている

しかし，元の内容はそのまま残っている
これがファイルの大きさが変わらなかった理由

"c"モードの動作

それに対し，第7－7図の第20行の"w"を"c"に変えてプログラムを実行してみます。すると，今度はファイルの大きさが変化します【第7－10図】。

第7－10図 "c"モードで書き込む

```
run
Ok

files "b:who.xx"
 1182 Kバイトが使用可能です
    "B:¥who                .xx "    /*       12 88/01/28 14:27:24
Ok
```

今度は大きさが変わった

第7－11図 実　行

```
run
ファイル名? b:who.xx
上書きします
Ok
```

今度は新しく書き込んだ内容だけが残っている

ファイルには，新しく書き込んだ内容しか格納されません【第7－11図】。

まとめ

このように両モードには，

"w"と"c"の違い

> "w"モード——前のファイルに上書きする
> "c"モード——新しく書き込んだ内容だけをファイルする

といった明確な違いがあります。通常のプログラミングにおいては，後者がふさわしい姿でしょう。ですからファイルに書き込むといっても実際は，"w"モードではなく，"c"モードを使用する機会が多いと思われます。

Q53 既存ファイルの保護を考慮するには?

A53 ファイルの書き込みは危険な処理である

ディスクファイルへの書き込みを行う場合,

既存ファイルの保護をどう設計するか?

という問題があります。これを**第7-12図**のプログラムを例に考えてみます。

これは, X-BASICのプログラムをコピーするプログラムです。まずは, その動作を見てみましょう。**第7-13図**をご覧ください。2つのファイルがあります。ここにある"isline.bas"の内容をもう一方のファイル"new.bas"

第7-12図　copy.bas

```
 10 /*-------------------------------------------
 20 /* basic のプログラムをコピーする
 30 /*      ファイル名の入力で .bas は省略する
 40 /*                                  87.10.21
 50 /*-------------------------------------------
 60 error off
 70 str fn1, fn2, buf[255], a
 80 int fp1, fp2
 90 input "どのファイルをコピーしますか"; fn1
100 fn1 = fn1 + ".bas"
110 fp1 = fopen(fn1, "r")
120 if fp1 = -1 then {
130     print fn1; " がオープンできません"
140     end
150 }
160 input "コピー先のファイル名は "; fn2
170 fn2 = fn2 + ".bas"
180 fp2 = fopen(fn2, "r")
190 if fp2 <> -1 then {
200     print fn2; " は既に存在しています"
210     print fn2; " に書き込んでも良いですか(y) ";
220     input a
230     if a <> "y" then end else fclose(fp2)
240 }
250 fp2 = fopen(fn2, "c")
260 if fp2 = -1 then {
270     print fn2; " が作成できません"
280     end
290 }
300 while freads(buf, fp1) <> -1
310     fwrites(buf, fp2)
320     fwrites(chr$(&HD) + chr$(&HA), fp2)
330 endwhile
340 fwrites(chr$(&H1A), fp2)
350 if fcloseall() = -1 then {
360     print "ファイルをクローズできません"
370 }
```

にコピーしてみようというわけです。このとき，注意しなければならないのは，もしそうしたなら

”new．bas”の元の内容は消えてしまう！

ということです。これは，大変危険な行為です。ですからすでにコピー先のファイルが存在している場合，**第7-14図**のようにそれを消してもよいかを確認するようにしています。

第7-13図　コピー前の状態

```
files "b:isline.bas"
 1185 Kバイトが使用可能です
    "B:¥isline              .bas"    /*      152 87/12/05 15:05:14
Ok

files "b:new.bas"
 1185 Kバイトが使用可能です
    "B:¥new                 .bas"    /*      660 88/01/26 22:02:20
Ok
```

コピーする（大きさの変化に注意）

第7-14図　実　　行

```
run
どのファイルをコピーしますか? b:isline
コピー先のファイル名は ? b:new
b:new.bas は既に存在しています
b:new.bas に書き込んでも良いですか(y) ? y
Ok

files "b:new.bas"
 1185 Kバイトが使用可能です
    "B:¥new                 .bas"    /*      152 88/01/26 22:03:08
Ok
```

コピー先のファイルがすでに存在している場合は，
書き込んでも良いか注意してくれる

コピーされている

2つの設計方法

このようにファイルへの書き込み動作が伴うプログラムでは，すでに存在するファイルに注意する必要があります。そのような配慮をまったく行わないというのも1つの設計方法です。Human68kのcopyコマンドは，そのように設計されています。逆に書き込んでよいのか確認を取る方法もあります。後者については，多くの日本語ワードプロセッサにその例を見

ることができます。

以上のことはどちらが良いという問題ではなく，アプリケーションの性格とプログラムの設計者の考え方によるでしょう。

ファイルの存在を調べるには?

この後者の例のように，ファイルに書き込みを行う前にそのファイルの存在を調べ，そこに書き込んでよいかを尋ねるようなプログラムの作り方を考えてみます。指定したファイルが存在するかどうかは，**ファイルのオープン時**にわかります。オープンモードのうちcを除く，r，w，rwモードでは，指定されたファイルが存在しないと，

存在しない場合⇨　　－1

を返します。ですからこれら3つのうちの適当なモードでそのファイルをオープンし，－1であればそのファイルは存在しないので安心して書き込んでよいということになります。このプログラムでは，とりあえず

fp＝fopen（……，　"r"）；

のように読み込みモードでオープンしています（これから書き込みを行うのに"r"モードとは変な感じですが，ただファイルの存在を確かめるだけですのでモードは何でもよいのです）。そして，

```
if fp <> －1 then {
    ファイルがすでに存在していることを表示
    そのファイルに書き込んでよいかを確認
    書き込んではまずい場合は end
}
```

のような処理を行っているのです。

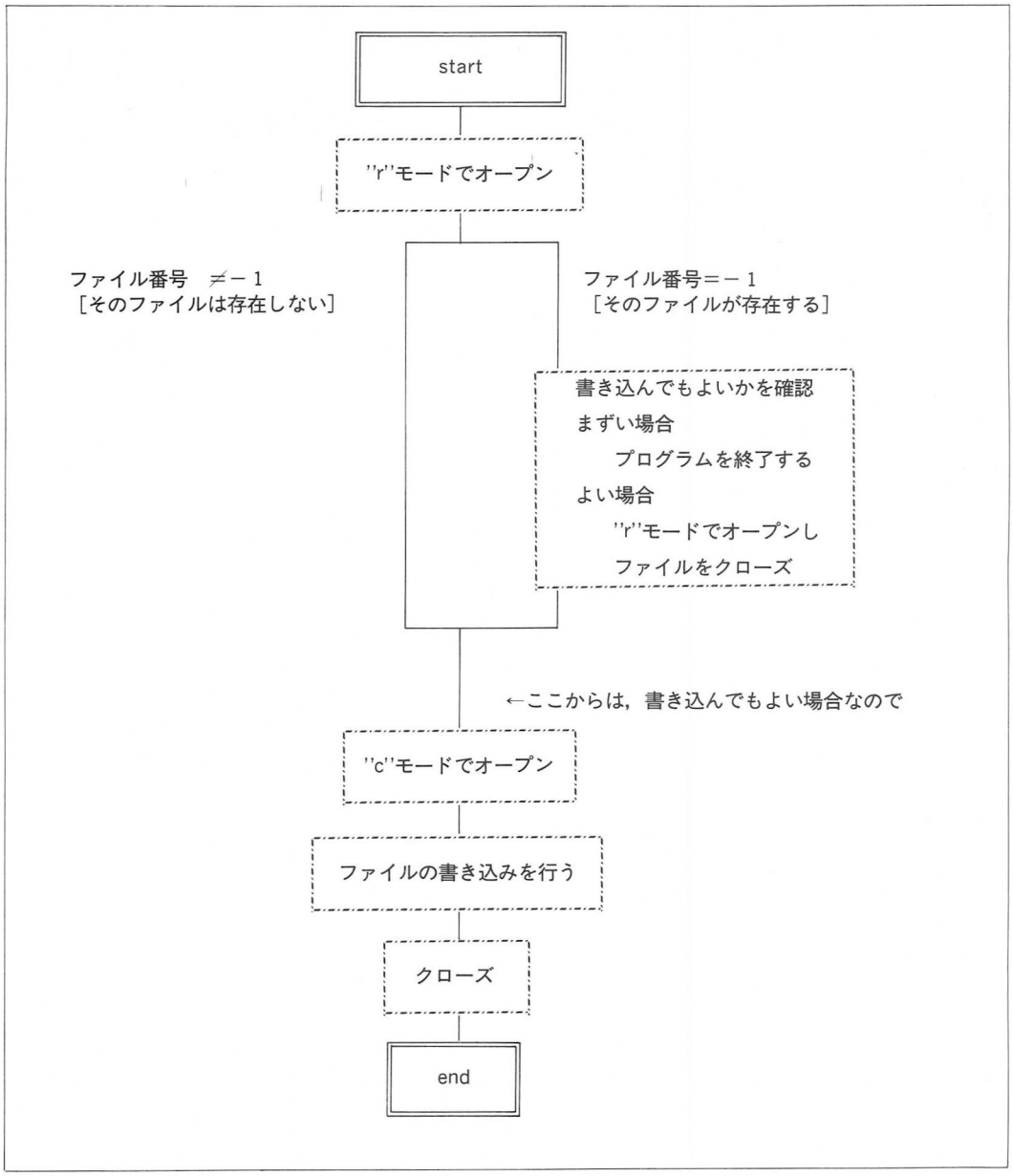
オープン時の注意

ここで書き込んでもよい場合，重要な注意があります。それは，この"r"モードでオープンしたファイルを

fclose（fp 2）

のように，1度**クローズしなければならない**ということです（行番号230）。なぜなら今度は，書き込みのためにもう1度オープンしなければならないからです。そのような手間をかけるなら最初から"w"モードでオープンしておけばいいと思われるでしょう。しかし，次の2つの理由により，"w"モードではダメなのです。

第7-15図　ファイルの書き込み処理



・"w"モードでは，既存のファイルに上書きするように書き込む。このことは元のファイルよりも小さい内容を書き込んでもファイルの大きさが元のままに残ってしまうことを意味する
・ファイルが存在しなかった場合，"w"モードではオープンできない

そこで，ファイルが存在していてもいなくても新しく

"c"モード——新規ファイルを読み書き両用でオープンする

でオープンするのがよいのです。これなら既存のファイルが存在しても，そのファイルを自動的に削除した上でオープンしてくれます。

まとめると，あるファイルに書き込みを行う場合の処理は，**第7-15図**のようになります。

Q54 fseek( )は，何に使う？

A54

fseek( )が必要な場面

fseek ( ) は，

データポインタ

の位置を変更するのに使用します。通常のディスクアクセスでは，あまりデータポインタを意識する必要はありません。ディスクアクセス用の関数の位置を変更するのに使用します。通常のディスクアクセスでは，あまりデータポインタを意識する必要はありません。ディスクアクセス用の関数の方で自動的にデータポインタを変更してくれるからです。ですからfseek ( )が必要なのは，意識的にデータポインタの位置を変更したい場合——具体的には，

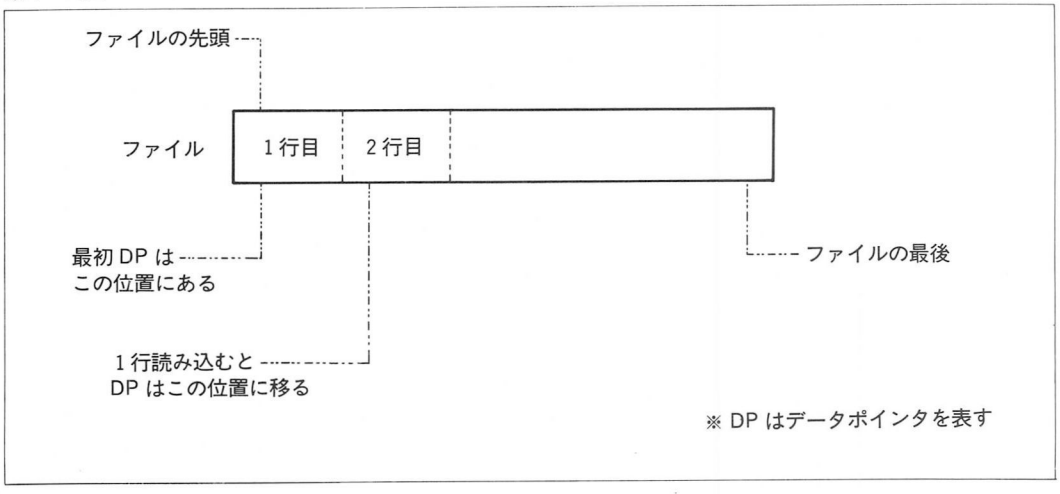
- ランダムアクセスを行いたい
- ファイルの終わりにアペンドしたい
- ある行の途中から読み書きしたい

といった場合等に使用します。

データポインタの移動

データポインタ　　データポインタとは，

第7-16図　データポインタの移動



次に読み書きするファイル位置

を覚えているポインタです。ファイルをオープンした時，データポインタはファイルの先頭に初期化されます。そして，ファイルの読み書きを進めるうちに，自動的にファイルの終わりに向かって進んでいきます。たとえば1行読み込めば，ファイルポインタは次の行の先頭に位置するように自動的に更新されます【第7-16図】。

サンプルプログラム

fseek（）を使用したサンプルプログラムを示します【第7-17図】。

第7-17図　cutpr.bas

```
 10 /*-------------------------------------------
 20 /* basic のプログラムをプリンタに出力する
 30 /*      行番号はカットする
 40 /*      ファイル名の入力で .bas は省略する
 50 /*                                  87.11.10
 60 /*-------------------------------------------
 70 error off
 80 str fn, buf[255]
 90 input "ファイル名"; fn
100 fp = fopen(fn + ".bas", "r")
110 if fp = -1 then {
120     print "ファイルがオープンできません"
130     end
140 }
150 print
160 while not feof(fp)
170     fseek(fp, 6, 1)          /* 行番号を読み飛ばす
180     freads(buf, fp)
190     lprint buf
200 endwhile
210 if fclose(fp) then {
220     print "ファイルがクローズできません"
230     end
240 }
```

これで先頭の6バイトをスキップできる

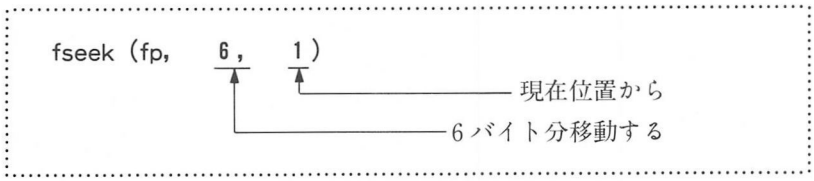
このプログラムは，BASICのプログラムファイルを行番号をカットしてプリンタに出力するものです。BASICのプログラムをプリンタに出力するには

freads (buf, fp)

で1行読み込んでは，

lprint buf

でプリンタに出力すればよいのです。ところがこのままでは，目的の行番号をカットすることができません。そこで，fread( )で1行読み込む前に



で行の先頭の6バイト分をスキップするようにします（行番号170）。

Q55 ディスクの残り容量を調べるには?

A55

dskf ( )

dskf ( ) という関数を使用して，ディスクの残り容量を調べることができます。dskf ( ) は，

dskf（ドライブ番号）

［ドライブ番号は
　0――カレントドライブ
　1――ドライブA
　2――ドライブB
のように指定する］

の書式で使うことができます。

第7-18図に，ディスクの指定したドライブのディスク容量を調べるプログラムを示します。実行例は，**第7-19図**のようになります。

第7-18図　dskf.bas

```
 10 /*---------------
 20 /*  test dskf()
 30 /*---------------
 40 str n
 50 print "---- 残りのディスク容量を調べます ----"
 60 print "どのドライブを調べますか？"
 70 print " 0 --ｶﾚﾝﾄﾄﾞﾗｲﾌﾞ   1 --ﾄﾞﾗｲﾌﾞA   2 --ﾄﾞﾗｲﾌﾞB .... ";
 80 n = inkey$
 90 print n
100 print dskf(val(n)) / 1024; "K byte free"
```

第7-19図　実　　行

```
run
---- 残りのディスク容量を調べます ----
どのドライブを調べますか？
0 --ｶﾚﾝﾄﾄﾞﾗｲﾌﾞ   1 --ﾄﾞﾗｲﾌﾞA   2 --ﾄﾞﾗｲﾌﾞB .... 0
243 K byte free
Ok                    カレントドライブのディスク残量

run
---- 残りのディスク容量を調べます ----
どのドライブを調べますか？
0 --ｶﾚﾝﾄﾄﾞﾗｲﾌﾞ   1 --ﾄﾞﾗｲﾌﾞA   2 --ﾄﾞﾗｲﾌﾞB .... 2
1183 K byte free
Ok                    ドライブBのディスク残量
```

補章

ファンクションキー活用テクニック

1．簡単なテクニック

まずは，簡単なテクニックからまいりましょう。

●ディスクのイジェクト

CTRL ＋ f・1　　CTRL ＋ f・1
ドライブAのディスクをイジェクトする

CTRL ＋ f・2　　CTRL ＋ f・2
ドライブBのディスクをイジェクトする

たとえば CTRL ＋ f・2 で，ドライブBのディスクをイジェクト――すなわち取り出すことができます。

●シフトモードの内容を表示する

CRTディスプレイの最下行には，通常 f・1 ～ f・10 の内容が表示されています。ところが

CTRL ＋ f・7　　CTRL ＋ f・7
SHIFT キー併用のファンクションキー内容を表示する

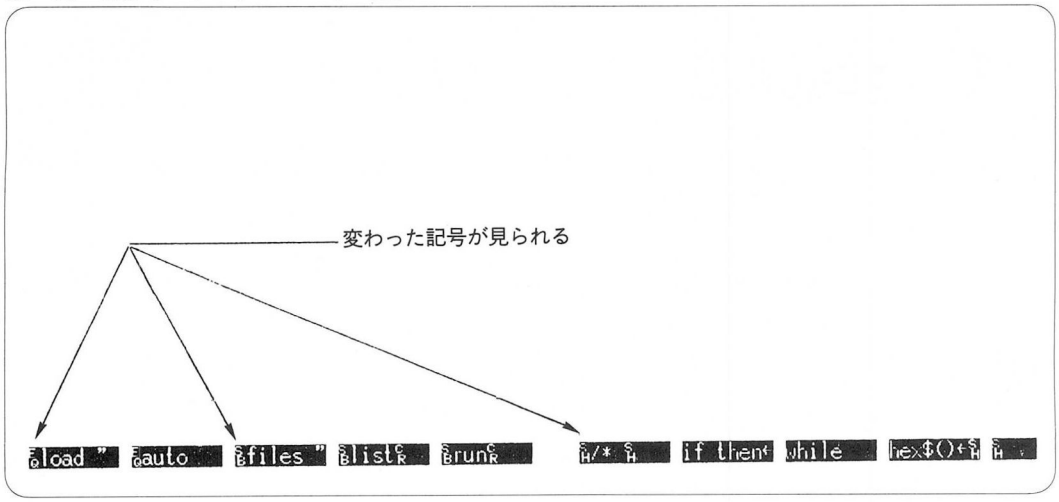
を押すことにより，SHIFT キーと併用させた場合のファンクションキー内容を表示させることができます。

2．こんなテクニックもある

私がX-BASICで作業をしているところを見ると，たいていの人はびっくりします。次のように，とんでもないファンクションキーの設定をしているからです**【第1図】**。

事実，私のX-BASICにおけるファンクションキーの設定は，**第2図**のようになっています。

第1図　奇妙なファンクションキー定義



第2図　定義内容

```
key list
key  1,"@Eload @@"
key  2,"@Eauto "
key  3,"@Zfiles @@a:@M"
key  4,"@Zlist@M"
key  5,"@Zrun@M"
key  6,"@A/* @A"
key  7,"if then@]@]@]@]@A"
key  8,"while "
key  9,"hex$()@]@A"
key 10,"@A    @A"
key 11,"@Esave @@"
key 12,"@Erenum@M"
key 13,"@Zfiles @@b:@M"
key 14,"@Zlist -@]@A"
key 15,"@Esystem@M"
key 16," = "
key 17,"else "
key 18,"endwhile@M"
key 19,"&h"
key 20,"@A    @_@]@]@]@]@A"
Ok
```

変な記号を導入している

【補　注】

key list コマンドにより，現在のファンクションキーの設定内容を表示させることができます。

とんでもない設定内容になっていますが，これがなかなか便利なのです。

なぜこのような設定をしているのか？

また，そのしくみはどうなっているのか？

それをこれから説明することにします。これを参考に独自の設定をくふうしてみてください。

テクニック1

関連内容をSHIFTモードに設定する

あたり前といえばあたり前のことですが，ノーマルモードとSHIFTモードでは，なるべく関連ある内容に設定しておくと便利です。これは，憶えやすいこととキー操作がしやすいことの2つの意味合いがあります。たとえば私は

f・7 ――――――――――――if～then
SHIFT ＋ f・7 ――else

のような設定をしています。

テクニック2

@M　改行は@Mを指定する

たとえばrunコマンド等は，ただちに改行させたいものです。そのような場合は

run@M

のように最後に@Mを付けるように設定すると，そこで改行してくれます。

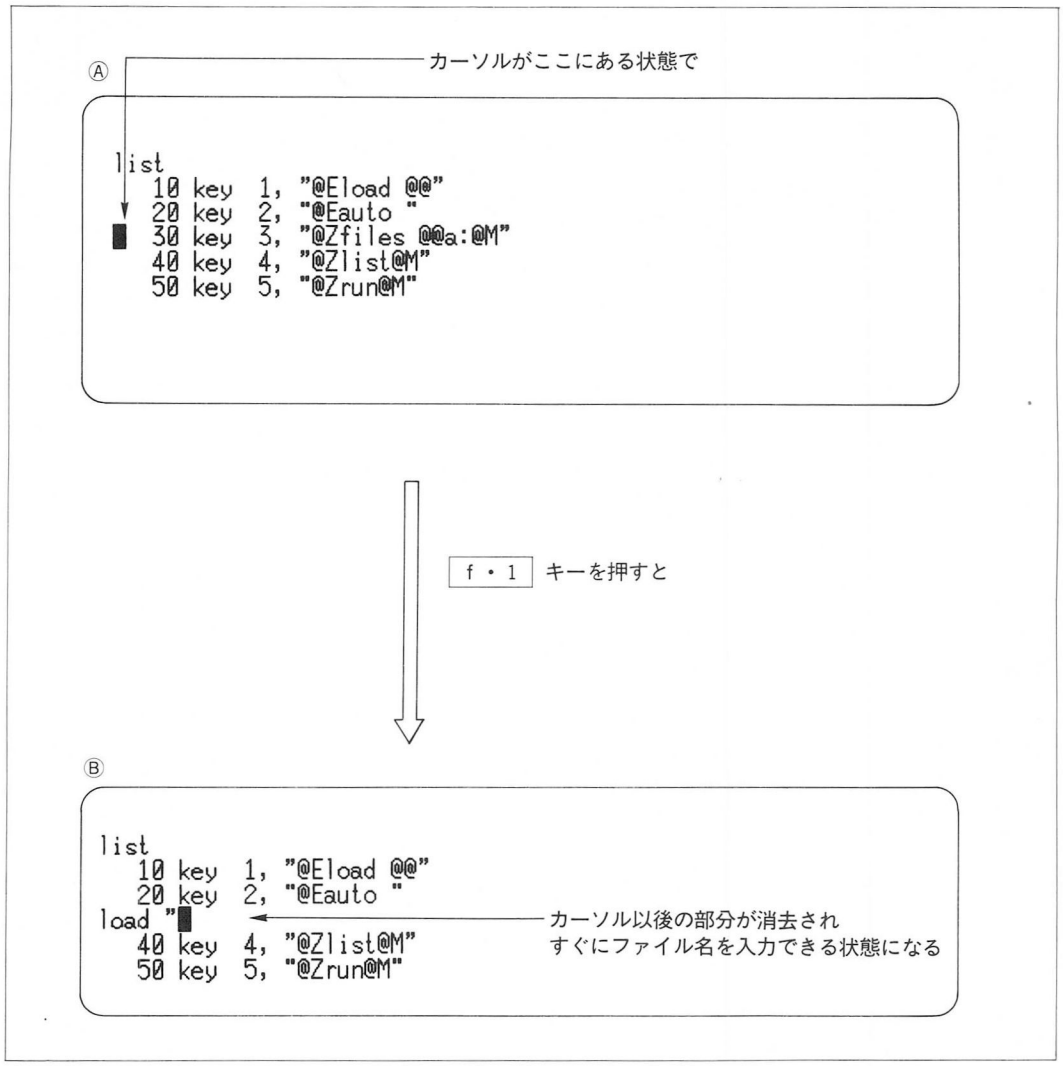
テクニック3

@E　カーソル行をクリアするには@Eを指定する

いま**第3図**Ⓐの位置にカーソルがあるものとします。この状態で，loadコマンドを使いたいとします。loadコマンドは，f・1に登録してあります。このような場合，(元の内容がじゃまになりますので)通常は画面をクリアしてから実行するものです。ところが私の設定では，このような場合でもいきなりf・1を押すことができます。最初にその行をクリアしてからloadを表示してくれるからです。

このように@Eを付けてファンクションキーの設定を行いますと，最初にその行をクリアしてから表示してくれます。

第3図　@Eの効果



@Z

テクニック4

カーソル以後をクリアするには@Zを指定する

@Eに似たテクニックに@Zがあります。たとえばfilesコマンド等を実行する場合に便利です。すでに何かが表示されている上にカーソルがあるとき，通常なら1度画面をクリアしてからfilesコマンドを実行します。さもないと前の表示が残ってしまいます。しかし，

@Z

をファンクションキーの先頭に定義しておきますと，あらかじめカーソル以後をクリアしてから実行してくれます**【第4図】**。

第4図　@Zの効果

```
list
   10 key  1, "@Eload @@"
   20 key  2, "@Eauto "
   30 key  3, "@Zfiles @@a:@M"
files "a:
  243 Kバイトが使用可能です
     "A:¥CONFIG              .SYS"   /*      208 87/10/20 09:14:16
     "A:¥AUTOEXEC            .BAT"   /*       40 87/11/02 10:44:56
     "A:¥X68K_M              .DIC"   /*   625664 87/03/15 12:00:00
     "A:¥X68K_S              .DIC"   /*    14336 87/03/15 12:00:00
     "A:¥SYS                 .   "   /* --DIR-- 87/10/05 10:55:42
     "A:¥BIN                 .   "   /* --DIR-- 87/10/05 10:55:46
     "A:¥BASIC               .   "   /* --DIR-- 87/10/05 10:55:50
Ok
```

カーソル以後，画面の下までクリアしてから
files が実行される

@A

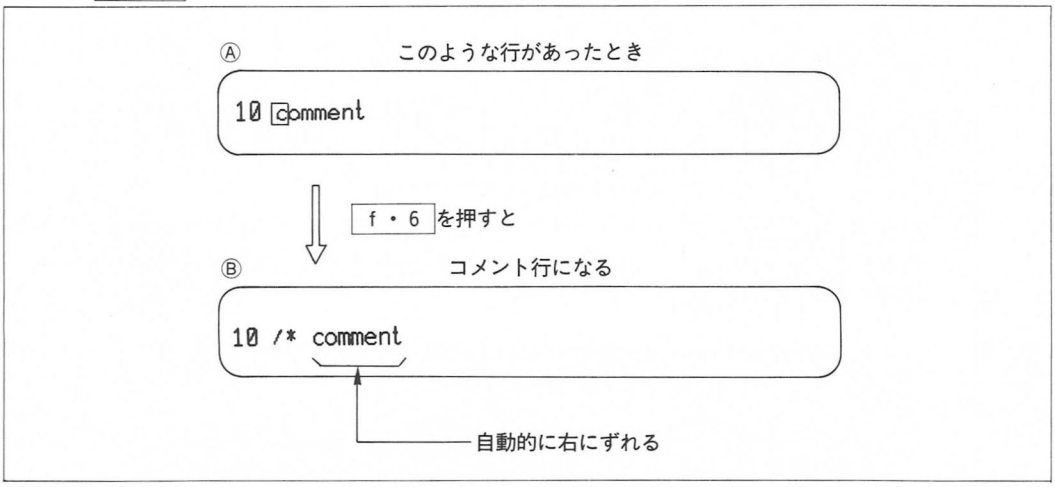
テクニック5

自動的に挿入モードにするには，@Aを指定する

いま**第5図**Ⓐの位置にカーソルがあるとします。この行をコメント行にする場合，通常なら INS キーを押して**挿入モード**にしてから

／＊

第5図　f・6



を挿入します。しかし，ファンクションキーに@Aを定義しておきますと，自動的に挿入モードにすることができます。ですから私の定義では，[f・6]を押すだけで簡単にコメントを作ることができます【第5図Ⓑ】。

なお@Aで挿入モードにした場合は，最後にもう1度@Aで上書モードに戻しておくとよいでしょう。

テクニック6

@[

カーソルを左に戻すには，@[を指定する

私の定義では，[SHIFT]＋[f・4]を押しますと，

list　－

という表示が現れ，カーソルが'－'の上に位置し，かつ自動的に挿入モードに入るように設定されています【第6図Ⓐ】。ですから行番号を入力するだけで

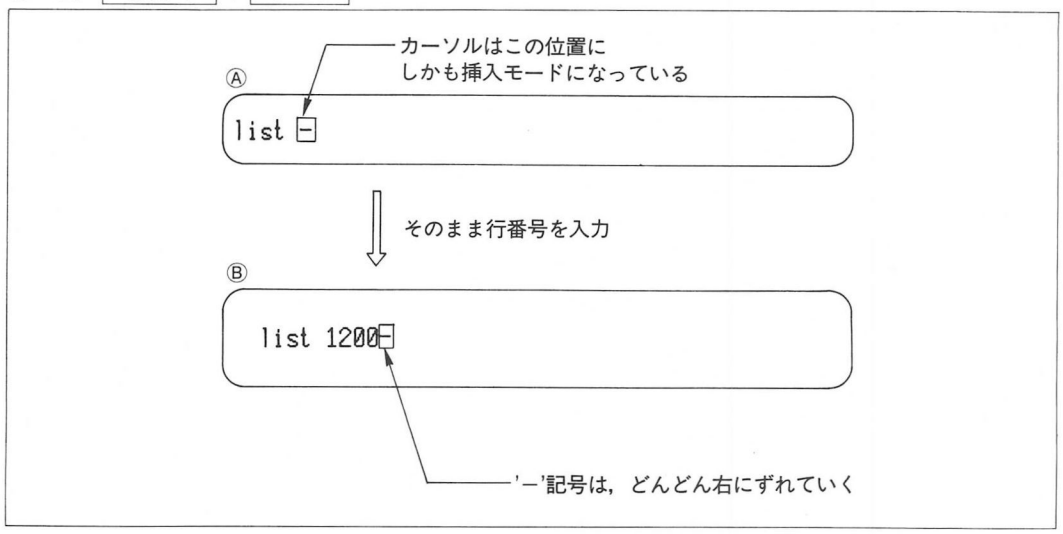
list　1200－

といった表現が簡単にできます【第6図 Ⓑ】。

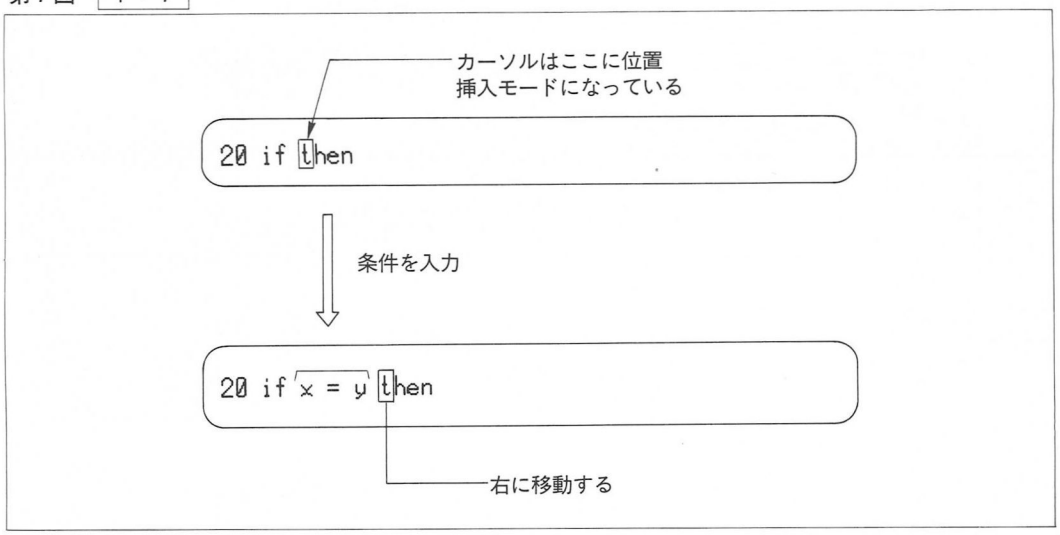
ファンクションキーにこのような内容を定義するには，「list －」を表示した後，1文字**カーソルを戻す**必要があります。その定義を@[ですることができます。

同様のテクニックは[f・7]の「if～then」【第7図】や[f・9]の「hex$( )」【第8図】にも取り入れられています。

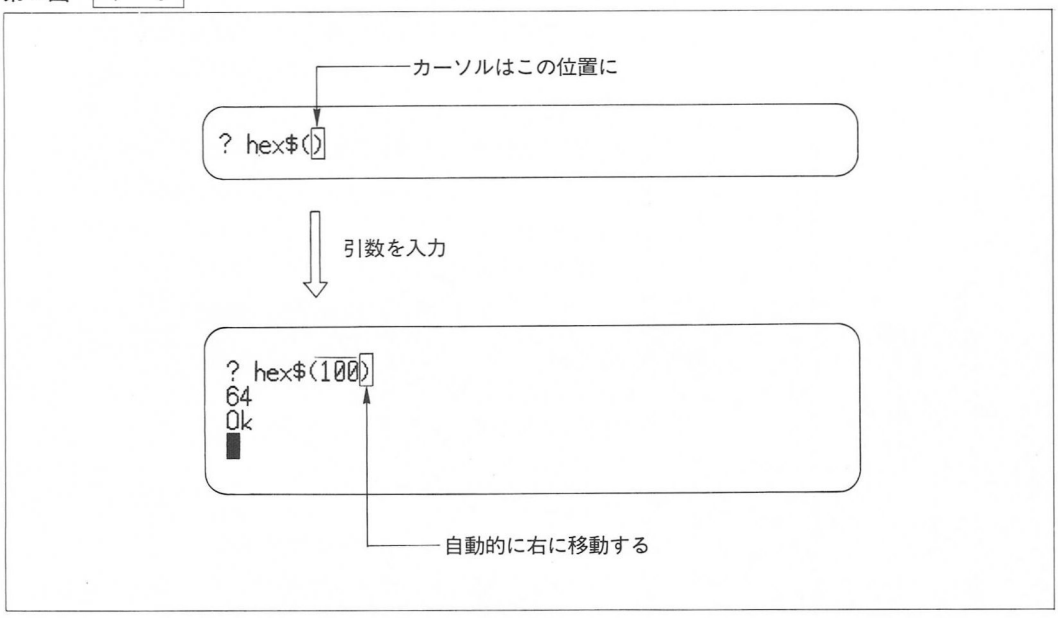
第6図　[SHIFT]＋[f・4]



第7図　f・7


カーソルはここに位置
挿入モードになっている
20 if then
条件を入力
20 if x = y then
右に移動する

第8図　f・9


カーソルはこの位置に
? hex$()
引数を入力
? hex$(100)
64
Ok
自動的に右に移動する

3. ファンクションキーの設定例

最後に，私のファンクションキー定義例を示しておきます。

ファンクションキーの定義例

f・1	load ” ・最初にその行をクリアしてから表示 ・’ ”’を表示させるには，@@とする
f・2	auto ・最初にその行をクリアしてから表示
f・3	files ”a：改行 ・最初にカーソル以下をクリアしてから表示
f・4	list 改行 ・最初にその行をクリアしてから表示
f・5	run 改行 ・最初にその行をクリアしてから表示
f・6	／＊ ・コメントを挿入する ・自動的に挿入モードにしてから実行
f・7	if then ・実行後，カーソルは’t’に位置する ・自動的に挿入モードに入るのですぐに条件を入力できる。
f・8	while
f・9	hex $ () ・実行後，カーソルは’) ’に位置する ・自動的に挿入モードに入るのですぐに式を入力できる。

f・10	スペース を4つ挿入する ・4字毎のインデントを行うのに便利 ・自動的に挿入モードにしてから実行される ・挿入後は上書きモードに戻る
※ f・11	save　” ・最初にその行をクリアしてから表示
※ f・12	renum 改行 ・最初にその行をクリアしてから表示
※ f・13	files　”b：改行 ・最初にカーソル以下をクリアしてから表示
※ f・14	list　－ ・実行後，カーソルは’－’に位置する ・自動的に挿入モードに入るのですぐに行番号を入力できる
※ f・15	system 改行 ・最初にその行をクリアしてから表示
※ f・16	スペース ＝ スペース ・C言語を意識するなら’＝’の前後に スペース を挿入するのが望ましい
※ f・17	else
※ f・18	endwhile 改行
※ f・19	& h
※ f・20	スペース を4つ挿入後，カーソルを1行下げる ・ f・10 に似ているがカーソルを1行下げるところが異なる ・連続的に4字毎のインデントを行うのに便利

※ f・11 ～ f・20 はSHIFTキー併用を表す

このような設定を行うためのプログラムを**第9図**に示しておきます。これを

AUTORUN.BAS

の名前でファイルしておきますと，X-BASIC起動時に自動的にファンクションキーを設定することができます。

第9図　AUTORUN.BAS

```
 10 key  1, "@Eload @@"
 20 key  2, "@Eauto "
 30 key  3, "@Zfiles @@a:@M"
 40 key  4, "@Zlist@M"
 50 key  5, "@Zrun@M"
 60 key  6, "@A/* @A"
 70 key  7, "if then@]@]@]@]@A"
 80 key  8, "while "
 90 key  9, "hex$()@]@A"
100 key 10, "@A    @A"
110 key 11, "@Esave @@"
120 key 12, "@Erenum@M"
130 key 13, "@Zfiles @@b:@M
140 key 14, "@Zlist -@]@A"
150 key 15, "@Esystem@M"
160 key 16, " = "
170 key 17, "else "
180 key 18, "endwhile@M"
190 key 19, "&h"
200 key 20, "@A    @_@]@]@]@]@A"
```

付　録

X-BASICクイックリファレンス

1. X-BASICの特色

●構造化プログラミングスタイル

構造化文の整備

- func～endfnc
- switch
- ブロックif

ローカル変数

再帰呼び出し

●外部仕様は言語仕様から切り離されている

外部関数

- グラフィック，サウンド，スプライト等のハードウエア機能が外部関数として定義されている
- 外部関数は別売のC，アセンブラにより作成できる
- これによりオリジナルBASICを構築できる

強力なハードウエアを反映している

- intが4バイト整数である
- その他

2. 起　動　法

起動法

BASIC　[〈オプション〉][〈自動実行ファイル名〉]

●起動時オプション

／c 〈コンフィギュレーションファイル名〉
／f 〈フリーエリアサイズKB〉
／w 〈画面サイズ〉

●フリーエリア

テキストエリア＋変数エリア＋配列エリア

●コンフィギュレーションファイル（拡張子＝．CNF）

beep＝off
ビープ音を止める

caps＝on
CAPS スイッチを押した状態で起動する

free＝160
フリーエリアを160KB 確保

func＝graph
外部関数ライブラリの GRAPH.FNC をリンクする

width＝96
1 行96文字モードにする

●外部関数

・使いたいライブラリをコンフィギュレーションファイルに指定する
・必要なものだけをリンクすればよいから BASIC をコンパクトにできる
・将来の拡張が簡単である
・ユーザーライブラリが蓄積できる
・標準ライブラリ

GRAPH.FNC ……グラフィックの制御
MUSIC.FNC …… FM 音源の制御
SPRITE.FNC ……スプライトの制御
STICK.FNC ……ジョイスティックの制御
MOUSE.FNC ……マウスの制御
AUDIO.FNC …… ADPCM の制御

●自動実行

BASIC 起動時に BASIC のプログラムを自動実行することが可能
方法は 2 つあり，AUTORUN.BAS が優先される
1〉『BASIC 〈自動実行ファイル名〉』で起動
2〉自動実行ファイル名を AUTORUN.BAS にしておく
カレントディレクトリまたは PASS の設定されだディレクトリに置く

3．オペレーション

● X-BASIC の文（命令文）の実行形態

ダイレクトモード

先頭に行番号があればスクリーンエディタ
なければ文（命令文）を即時に実行

プログラムモード

メモリ上のプログラムを実行するモード

●スクリーンエディタ

入　力

改行 を押す

追　加

新しい行を入力
行番号順に整列される
ゆえに10おきくらいにする
間隔が足りなくなったら renum

削　除

削除行の行番号だけを入力して 改行
delete 命令

行の置き換え

同じ行番号で新しく入力すればよい

コントロールキー

マニュアル巻末 P.174「コントロールコード一覧」

●プログラムの実行開始

おもな実行開始の方法

RUN
GOTO

プログラムの途中から実行を開始する時

GOSUB

ダイレクトモードでサブルーチンのテストを行う場合

4．プログラム

●プログラム

プログラムは，行の集合である

行は，次の構成要素を持つ

行番号　文　［：文……］

※１行は255文字以内

コメント

／＊以降の部分

●マルチステートメント

１行に：（コロン）で区切ることにより複数の文を置くことができる

●行番号

１～65535の整数

0や65536以上を使用すると『行番号が異常です』エラー

負数を使用すると『文が実行できません』エラー

5．言語仕様

●文

・命令文のこと

・大文字，小文字のいずれでも入力可能だが，リストを取ると小文字になる

・文字列，変数名，関数名はそのままの形で記憶される

●文（命令）の種類

コマンド

主としてプログラムに対する命令（本体はOSのコマンドに相当）

ダイレクトモードで使用

プログラムモードでも使えるものもある

ステートメント

プログラムの中で核となる命令

関　数

与えられた引数に対する答えを返す

●データ型

数値型

整数型

char 型

1バイトで表現される符号なし整数

0～255の値を取る

文字コードの表現に便利

int 型

4バイトで表現される符号なし整数

−2，147，483～2，147，483，647の値を取る

実数型(float 型)

8バイトで表現される倍精度実数

MSB が符号を表す

残り52ビットが仮数部，11ビットが指数部

絶対値は，1.1125369292536E−308～3.595862697246E＋308

文字型(str 型)

255文字以内

" "で囲む

内部構造的には，最後に文字コードの 0

文字列の大きさを指定しないと32文字まで

●整数型定数の表現

10進形式 ……負数は−，＋は付けても付けなくとも可
8 進形式 ……&O（0で入力してもリストを取ると O）
16進形式 ……&H
2 進形式 ……&B
1 文字形式 ……'A'

●実数型(浮動小数点)定数の表現

・int の範囲を超えた整数

・小数を含む数
・最後に．または#を付けた数
・指数表示：仮数（14桁以内）E指数（±308）
・リスト時
　実数型定数は必ず最後に#が付く
　14桁を超える表現は，自動的に指数表示になる
　小数点を含む15桁以上の表示は四捨五入される

●型変換の規則

数　値

代入先の型に変換される
演算の中では型の高い方へ
実数が整数に変換される時，小数以下は切り捨て，整数の範囲を超えたらエラー

論理演算

整数型に変換され，結果は int

●変　　数

変数名

64文字までの英数字
先頭は英字
途中に予約語を含んでよい
大文字・小文字は区別される

変数の宣言

型宣言子　変数名　［＝定数式］［，……］

変数に代入する（整数型のみ）
宣言だけを行った場合の初期値は0，またはヌル
宣言子:＝ char ， int ， float ， str
同じ名前の変数を別の型に宣言し直すことは不可

char　変数名［＝定数式］［，……］

例› char a='g'
　変数aには，gのASCIIコードが1バイトで格納される

str　変数名［文字バッファサイズ］［＝文字定数］，……

文字バッファのサイズは１～255文字まで
文字バッファサイズを指定しない場合は，32文字分取られる
バッファを超えた分は，切り捨てられる
文字列の最後には０が入るので，実際は指定したサイズ＋１の領域が確保される

1991年１月１日より東京都内局番が変わります。現在使われている３ケタの局番の前に３がつきますのでご注意下さい。
尚この本に掲載されている広告及び内容も同様です。

月刊マイコン別冊
X68000 活用研究Ⅲ
X-BASIC活用Q&A
塚越一雄著
昭和63年７月１日発行
1990年12月20日第２刷発行

定価2,060円（本体2,000円）

発行人　平山哲雄
発行所　株式会社電波新聞社
〒141　東京都品川区東五反田1-11-15
電　話　03(445)6111(大代表)
振　替　(東京)5-51961
印　刷　大日本印刷　株式会社
製本所　(株)堅省堂

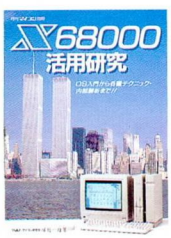
X-BASIC プログラミングの決定版 ついに登場!!

第１章　X-BASICの概要
1－1　X-BASICと一般のBASICの違い
1－2　X-BASICの特長
1－3　これからのX-BASIC
第２章　X-BASICの基礎
2－1　X-BASICプログラミングの基礎
2－2　変　数
2－3　関　数
2－4　制御構造とステートメント
2－5　ファイル管理
第３章　標準関数と外部関数
3－1　標準関数
3－2　システム変数
3－3　外部関数
3－4　定義関数
第４章　外部定義関数
4－1　外部定義関数の書式　その1
4－2　外部関数定義の実習
4－3　外部定義関数の書式　その2
4－4　システムコールとIOCSコール
4－5　外部定義関数の書式　その3
第５章　X-BASICで役立つ外部定義関数・事例集
5－0　事例集の書式
5－1　テレビコントロール関数「TVCTRL」
5－2　スプライト・グラフィック・テキストのプライオリティ設定関数「PRI」
5－3　マウスカーソルの定義関数「MSCSET」
5－4　使用したいマウスカーソル番号を設定する関数「MSCURSOR」
5－5　複合関数「XLINE」「TLINE」「TCLS」
5－6　XORタイプのCIRCLE「XCIRCLE」
5－7　検索機能付FILES関数「FFILES」
5－8　ポップアップメニュー関数「POP-UP」
付　録　応用プログラム TEXT-DRAW

月刊マイコン別冊

X68000 活用研究II X-BASIC マスター編

宮原哲也/深沢幸三　共著
B5判316頁
定価2,000円（〒300円）



OS入門から各種テクニック・内部解析を解説した

X68000 活用研究

好評発売中!!　B5判364頁　定価2,200円（〒300円）

電波新聞社　出版販売部
東京本社　〒141 東京都品川区東五反田1-11-15　☎(03) 445-6111
大阪本社　〒530 大阪市北区中之島3-2-4 朝日新聞ビル　☎(06) 203-3361
西部本社　〒812 福岡市博多区博多駅前2-13-23扇寿ビル　☎(092)431-7411

SHARP

New-Life People

あふれるクリエイティブマインド——NEW Z-BASIC搭載。

ADVANCED TURBO

NEW Z-BASIC搭載

多色グラフィック、カラー画像デジタイズ、ステレオFM音源、バンクメモリ対応などクリエイティブワークを強力にサポートするAV指向の高水準BASICです。グラフィック用関数、X68000と命令コンパチの拡張MMLをはじめ使い込むほどに凄さがわかるパワフルなBASICを搭載しました。

先駆のAVアート機能

量子化、モザイク、反転などトリック取り込み処理をサポートしたカラー画像デジタイズ機能標準装備。さらに、クロマキー合成、インターレーススーパーインポーズ、4,096色対応ニューテロッパ機能、8重和音のステレオFM音源。先駆のZアビリティがパソコンクリエイターを魅了します。●メインメモリ128KB標準実装(NEW Z-BASICで最大576Kバイトまでサポート)した大容量設計●1Mバイトフロッピー2基搭載●JIS第1/第2水準準拠漢字ROM、「システム・ユーザー辞書」標準装備●マウス標準装備●X1ターボシリーズの豊富なソフト資産が活用できるコンパチブル設計●多彩な通信ツール*のサポートでパソコン通信に対応●ドットピッチ0.31mmの高精細カラーディスプレイテレビ* *別売

パソコンテレビ X1 turbo Z II

本体＋キーボード	CZ-881C-BK(ブラック)	標準価格 179,800円
14型カラーディスプレイテレビ	CZ-880D-BK(ブラック)	標準価格 109,800円
14型カラーディスプレイテレビ	CZ-830D-BK(ブラック)	標準価格 98,000円
チルトスタンド	CZ-6ST1-B(ブラック)	標準価格 5,800円

※写真のディスプレイはCZ-880Dです。

シャープ株式会社 ●お問い合わせは…シャープ㈱電子機器事業本部システム機器営業部〒545 大阪市阿倍野区長池町22番22号☎(06)621-1221(大代表)
電子機器事業本部テレビ事業部第4商品企画部〒162 東京都新宿区市谷八幡町8番地☎(03)260-1161(大代表)

電波新聞社 〒141 東京都品川区東五反田1−11−15 定価2,060円(本体2,000円)

雑誌68469-4