Aristo Tacoma

The Art of Thinking

Vol. I: G15 PMN Programming for Kids

==========
BOOK WRITING IN PROGRESS.
UPDATED UNTIL IT IS COMPLETE.
==========

****
Publication info:
This is publically available in various forms, while it
is written, and after it is completed as well, confer
links at norskesites.org/fic3/fic3inf3.htm
When completed, it is a book that constitutes the first
volume in a five-volume series called The Art of Thinking.
This is also published in paper, in 2018, by
Yoga4d Avenuege, with ISBN number for National Library
of Norway assigned to the printed edition.

Author: Aristo Tacoma

Published by Avenuege Library, Yoga4d:VRGM, Oslo, 2018.
****

Spelling takes are by the angels.
--Anonymous

Note to the youngest readers:
I have tried to avoid the longest words but sometimes I have forgot myself and they are there! But then, just read on. Usually everything that is important is mentioned more than once. And as you read it a second time, you'll find that words that made little sense to you the first time, makes more sense then, as you get used to them, and see how the words are used. Play with your thoughts around that which you want to learn--play and experiment, try out things, and you'll pick it up soon enough. And good luck!

Note to the not youngest readers
I say 'for kids' because I mean to write to anyone who can read English texts, also slightly more complicated texts, but who do not have lots of experience with computers; and I try to use a language that is simple. But of course this may just happen to make it interesting to adults, as well. Wasn't it the physicist Albert Einstein who once said that "only if you can express it simply, have you understood it?" So this is not merely for kids,--it is about making computer programming simple.

Foreword

As you know, most forewords to books (when written by the author of the book) are written long after the book itself was written, and contain various excuses for why the book is the way it is and hope you like it anyway and so on.

This foreword ain't so.

The day, today, as I'm writing this, is sunny, and I'm sitting in a park and crowds, tiny crowds of people are around and children are playing and screaming and shouting. And, chewing on my pen, I'm wondering how to write this book.

I'm asking myself, what could possibly be the point of suggesting to these children and youngsters and young adults--and any adult--to spend time with something as technical and abstract as a "programming language"?

Why not just spend time with the concrete? And with natural language--such as getting good at English, develop humour and wittiness there? And to exercise physically, and to train elegant dance motions and postures and draw and paint and indeed do what's life is all about? So, to repeat my own question, why should anyone bother about programming?

Okay--some might answer--it could be a bit useful. To program probably means, for instance, you get better about handling numbers. Which again may help you to think better about money and money-making and that's helping everything else.

Yet--I would object--most other fields of learning of any kind are more truly useful than programming, in that most other fields somehow leads to concrete skills in dealing with life and opening up to life, perhaps.

But hang on a moment. What was that phrase we just used? "Opening up to life"--that phrase!

I will now suggest a real answer to why you--yes, you--should learn G15 PMN programming if you haven't already done so. And indeed any one. And I will use that phrase--"opening up to life". Here is what I claim--and please consider it:

To learn programming, and to work on making programs--and looking at them--opens your mind and heart to a greater experience of the orders of nature and life, rather as learning to draw can help you to look at the beauty of another's face.

The sun is shining very brightly now--after days of

clouds and rain--and I forgot to bring my sunglasses. If I try to look straight at the sun, I have to close my eyes four-fifth or so. And as I do it, bursts of radiant golden bows as if emerging from the golden-yellow-slightly bluish disk of the hot sun flicker in synchrony with any slight motion of my eyelids. It's quite a display. Life is full of harmonies, orders, patterns, and experiences, which can be read like a symphony of coincidences, synchronicities. And, like waves on the ocean, nothing in experience stands still but the music, the orders, flow on and on.

In all this vast movement, we have to relate to, and often change, our own expectations, plans, hopes, desires --and yet when the day is suddenly very beautiful, the mind as if orders itself. Intuition--your capacity to get things right without quite knowing how or why--comes more easily then, in what we can call "moments of harmony" or "moments of meditation".

"Meditation", the word, comes from an ancient greek word relating to good and proper "measures", as does "medicine" and "moderate".

So, in ancient traditions, words of deep meaning and with nice sounds to them, and patterns--called "mandalas", for instance--have been used to calm the mind of devotees of many religions. And in all these traditions, the experience of the connection between inner peace and silence, on the one hand, and a sense of beauty and love, on the other, has been commonly reported.

The simple pattern, the good sound--gently, effortlessly repeated--or the ritualistic action, done with attention for the sake of the energy of attention--these things, then, are part of a life where experience goes deeper, where the sense of art of living becomes possible.

And so, can it not be, then, that in the nice little repetitive patterns and words involved in a great programming language like G15 PMN (it is great, although, as creator of it, it is immodest that I claim it)--can lead to a more meditative life? A life in which you are constantly more open to the beauty of all experience? The sharpened sense of numbers you get by working with what is called "32 bit numbers" in this language is also no impediment to deep experience of the order of nature. Inside numbers there are beautiful patterns--they are not just heaps, but highly organised.

To conclude this foreword, I would suggest that whatever usefulness we may or may not find connected to programming, even a little bit regular touch, weekly or

perhaps daily, with something like G15 PMN is a connection to something of the inner workings of the very fabric of existence. If that sounds too grand, let us bear in mind that notable thinkers in ancient Greece thought the universe was "made up of numbers". Add to that a little (or a lot) movement, and you get algorithms, functions. And yet, the universe may have this inside its core, and yet be beyond it: for the universe is no machine, but alive, like you and me.

As motivator, then, to look into G15 PMN code, and do your little experiments with it--or make full programs on your own--is the idea that you are sort of cleansing your mind of the confusions of past experience and readying your mind to experience life anew. As if for the first time, and yet with a sense of constantly learning at your soul and spirit level.

A.T. April 10, 2018

PART A

Part A, chapter 1: SAY "HI" TO G15 PMN

Why is it called "G15 PMN"? It's just a name, of course. Ideally, the name of the language should tell you something about it, and has a nice and unique sound to it as well.

First, "G15". It is a name for the core. The core is meant to be part of something generous, graceful, etc. "G" on its own sounds rather like "gee", and has a bit of rhyme with "fifteen". This core, nearest the electronics that makes up a personal computer, is about numbers in movement. So a number being part of its name makes sense.

Fifteen is a neat number; it is, as you know, 3 times 5,

and 3 together with 5 is a combination we come back to in discussing "the golden ratio". And here we have a pair of numbers that can help you in drawing and painting. It's also possible to say much more about how 15 is meaningful, --it being small, yet big enough so that when we count from 0 to 15 we have what we will discuss later as "four bits". And more so.

Then "PMN". Mostly, it's just a good, short phrase, pee-emm-enn, again a rhyme within itself and a bit same type of sound as G15.

Where G15 on its own is mostly about numbers, PMN is a sort of layer on top or around G15 to make it more easy for us, more natural, more meaningful. Perhaps the "MN" is for "meaning", and "P" for "primary" or first. Perhaps M is for "matrix"--something we discuss later--or for "match" or for a big word about all existence, "multiverse". Then "N" may be as in "net"--something we'll briefly touch on when we say something in a chapter about robots and then "FCM"--or by a word like "noetics", meaning mind or the like.

But it is not the machine that has mind when you use G15 PMN--you have mind, and you get more activated in your mind as you use it. In a way, it is a language for the mind to speak to itself in a different way than what a natural language like English can offer.

In having a keyboard and a mouse and a monitor, all connected to an open personal computer, or PC, with G15 PMN in it, you have a kind of tool for thinking at your fingertips. The machine won't do any thinking for you, but like a camera in a way, it can give valuable impulses--what we call "feedback". (Some people have sometimes claimed that they can make machines think--don't believe these people.)

Most things in life are a bit gradual, on the move, not quite one way or another way. A big word for this is "analog". In doing something with the G15 PMN programming language, we find that things typically are either one way or another way. A word for this is "digital".

So we also say, "digital computer", "digital electronics". (Technically, we can say that the digital electronics of a G15 PMN computer has in it a G15 PMN CPU, or Central Processing Unit, of a 32-bit kind, which is not made by means of a 'microprocessor' but as a combination of many things we call "intraplates". But this is not necessary to know in order to learn and enjoy this language. By the way, if your computer has a key on the

keyboard named "DEL" or "DELETE" it is a different type of CPU in it, and G15 PMN is running on top of it; the DEL key is then used to connect to this other CPU.)

Sometimes there are bridges between the analog and the digital--fortunately! For instance, we can set up a digital computer to work together with machinery that is not digital but more analog, so as to produce analog, normal music we can listen to and dance to or relax to.

In each of our meetings with the programming language G15 PMN, we will either type something into the language itself (and we always use the keyboard, so it is time well spent to learn to touch-type on it without having to look at the letters on the keys all the time!)--or we will put it into bright green cards withy the two columns of large, easily redable letter. And then, after putting the cards in order, we make the PC go through one or more of them.

Then, if we have typed something wrong, we can go back to the card (the PC will tell us which number it has, so we'll find it easily, each time). There, we'll fix it and we can try again. This is called "program correction". It can take even more time than making the first form of the program--and all in all this is what we call "programming". Even the best programmers set aside lots of time for program correction. Sometimes, though, there's hardly anything to correct, and the program just works. That's fun, but it's not a goal to do away with program correction. Program correction is part of our thinking about what we really want the computer to do. Often we discover that we haven't entirely thought it through in this phase, and must lean back or go for a walk or come back to it next day with some fresh ideas. So program correction isn't only about correcting spellings and such: it is as often to look at our own thoughts and think even better thoughts--and then to change the program and see if the PC does what we thought it would do after this.

So, we either type something straight into the programming language--into what we also call "the terminal"--or we type it into cards that we save, one by one.

The terminal is a fun place to begin and it's also very useful in checking how a program works. The terminal we use, normally, in doing whether simple or very advanced G15 PMN programming, is also called "the TF terminal". Sometimes we just say "TF" or "tf". This is short for a long phrase which sums up stages in how PMN was shaped. The full phrase is "The Third Foundation" (it isn't

necessary to understand just why it is called that, just remember that when we say "terminal" or "tf" we usually mean the same type of working with the language as if somebody says "third foundation").

Have you ever played a game, such as TexasStars, on a PC with G15 PMN?

Then you have possibly done it this way--which is very similar to how we start the TF terminal--so I suggest you start up TexasStars this way if you aren't already familiar with this way of starting up that game of ours.

1. You switch on the G15 PMN PC (and if it is already up and running with G15 PMN, you take it off first).

2. You type the three letters

```
mnt
```

and you press lineshift. Some call this large button to the right of the letters for the ENTER-button, and we sometimes say it one way, sometimes the other way. The old way of saying it--from the times of the mechanical type-writers--was "carriage return", so on occasion you'll see the two letters CR to mean just the same type of thing. (G15 PMN often wants you to press that lineshift button, so when it appears to be waiting for some input and you don't know what to press, it is often correct to press that button.)

The three letters mnt are short for "mount", and it means, let's load some stuff into the PC.

3. The menu shows and you press the digit one:

```
1
```

4. Then you type the number of the program--also called "application", or "app", for short, which in the case of TexasStars shooting game are these seven digits:

```
101010
```

As you press lineshift after these then it should say something about SUCCESS. (If it doesn't, get somebody to help you set up the computer so that it has both this TexasStars app and the TF app in it.)

Press lineshift again and the menu will open.

5. At the menu, press the CTR button (also sometimes named CTRL)--which means 'control'--and hold it in, and, while holding it in, also give the W-key a quick press. Then release your hold on the CTR button. This we can write as:

```
CTR-W
```

This activates the mouse, so you can use the mouse to start programs. (A click on the right of the mouse makes switches off the menu mode again. Then the menues can be

changed, what we call the 'edit' mode. To get the mouse back to work if you hit the right side of it, press CTR-W again.)

6. Now you can use the mouse to start the program. Look for an arrow or flower-like type of thing, in this case after a big F and before a big number 1. Find this place with the mouse pointer, and click on the left side of the mouse to start the program. The whole app starts now, and a press on the large lineshift button is all it should take to get you going with the game! You shoot at the mysterious objects in the sky by a click on the mouse, and move by a push on left and right arrows. ESC to leave the game.

7. When done playing, type

```
qu
```

and press lineshift to get to the menu.

8. To leave the menu and the other cards--we also call this the CAR editor--press

```
CTR-Q
```

in the same way you pressed CTR-W above. Then type

```
reb
```

and press lineshift, and that allows the G15 PC to reboot and freshen itself and its socalled "memory" or RAM, so you can start it anew.

To start the G15 PMN language in its TF terminal, all you have to do is to go through exactly the same motions again, only choose to type, not 1010101 as app number, but 3333333 (you can just type as many 3's as the field allows, so it fills up to seven digits). Do the CTR-W and the mouse-click to start it.

Done it? Great!!!

Now that you have the Third Foundation, or TF, terminal, clear and present on your Personal Computer's display, you see there a sort of greeting graphics--a square made of a lot of lines and a bit of a triangle made out of tiny squares, and so on. As you press lineshift several times, you kind of carve a bit of black in that bright green graphics each time. If you press lineshift many times, you make the whole column black.

Let's connect to G15 PMN, say hello to it. Type the two letters:

```
hi
```

and press lineshift. You might enjoy the greeting it gives you. It was actually pretty much the first thing I got PMN

to do, when I made it. It wishes you good luck with large bright green letters on black. You find that this bright green makes the world around it look quite rosy--it is something called 'afterglow'--just like a yellowish sun makes the sky around it blue, so does a bright green computer monitor makes any white near it look a bit pink or red or something like that. It's called "complementary colors". But bright green, or yellow-green, is also found to be both relaxing and positive whether you work with it early in the day or late in the day or at night.

Some might ask, "Isn't it best to have a color PC? For then you can look at such as photos more like things are in real life." But think about it: do you really wish the PC to go so near real life it looks like it? Is that the point of computing machines? Or is it the point that they are going to be there for this and that and the other task --including calming our minds--but then we want to go out and leave the PC and experience the full colors of real, analog life. We don't want to make the PC into something we get addicted to, so we get lazy and do nothing but steer at it all day. So the PC is a good thing only if we don't allow every desire to have its say but keep back a little, keep it back a little, how much we develop it. Nobody who is sane want the machines to take over life or to melt with the machine somehow, and so the machines should not, must not get over-developed. And if they have been over-developed in the past--as they have--let us not contribute further to this over-development, but go beyond it and focus on what is good.

So you have typed in "hi" and you can do that again and again and you get the same greeting over and over again. Let's have the PC type your name in nice letters. It's time to use some of the other things on the keyboard than letters and numbers. For instance, above the digits you find many other things--we call them "characters" or "special characters". Over the digit 6 you get to ^ and over the digit 7 you get to &, if you press SHIFT key and hold it in while you also give the digit a quick press. The SHIFT key is sometimes written as "SHIFT" on the keyboard, at other times it is a sort of very broad arrow pointing up. It is the same key you use for getting big letters in when typing a text. (At some keyboards, what is above the digits may be moved around a little bit, but experiment and you'll find all you need.)

But first, let's clear the screen. Type

```
ce
```

and the screen gets all black. You can think of it as c as in "clear" and "e" as in "entire screen" or "everything". This happens when you press lineshift.

Type it in using small letters. (In the TF terminal, the big and small letters are rather the same, though, only that a dot appears on top of the letters that are meant to be thought of as big.) Don't put any blanks before these little commands. But if you do, just type it in once more.

So, what's your first name? Type this into G15 PMN:

```
&yourname&
```

only that you type whatever your name is inside the & and the &. Then press lineshift, and if you did it right, there is no message from G15 PMN. It just stores that name and you can get it out again, this way:

```
b9
```

Nice, right? It gives you your name, but it uses more smooth letters than what it usually uses for programs, because you said "b9". Try the same but slightly different --like this:

```
&yourname&
pp
```

So you type your name again, with the & around them, and then you use pp instead of b9. The "p" means "print", and "pp" simply means that it is a very simple and often used print command, and two p's are easy to type in. "B9" sounds like "benign" which means "nice, good, healthy", when you read it out. So we call the type of pp letters one type of "font" and the b9 type of letters for another type of "font". And so you have a B9-font and you have another font, that we sometimes call "robot-font", because --like robots--it is a bit square and machine-like. It is also easier to use when you do much programming in G15 PMN than any other font, this robot-font--because it is shaped so that this language is easy to work with using it.

Ready for another experiment with G15 PMN? The computer is nothing if not repetition! So let's get it to repeat your name ten times. Ready for it? A bit of typing we shall do. Can you manage to type this exactly? Try it. Then do it again if it didn't work. If all gets funny, type 'qu' and exit the G15 PMN terminal, press CTR-Q, type the 'reb' command to let G15 PMN refresh itself. As you start it up again, you'll find that it is entirely clear and it has no scars or memories of the last session--which is one of the wonderful things about programming--you get infinitely many new chances!

So, let's type in something that allows us to create a

new word in the language. That word will exist just for this session, for we do not store it on a card. So almost any name will do. Shall we call it "myname"? So we are going to say myname= something something. This "=" is usually high up on the keyboard, around the area where the lineshift button is. It is the "equal sign". We type in some stuff--at least one line, maybe several--and then at the last line in this little program, we put a dot, ".". Okay? So what we will type in is what we did above, but we are going to say--do this ten times--and the way we say it to the PC is this way: LL:10. Or with small letters, ll:10. This means, let's have a "loop". A loop is another word for "repetition". It wants to know what--how many lines to repeat--and so each time we type in LL: something then it wants, on a lower line, to have the letters LO, or --with small letters to be exact, "lo". So here we go, type in this as exactly as you can (just be sure to type your first name where it says 'yourname'):

```
myname=
ll:10
&yourname&
b9
lo.
```

If you typed this exactly--being sure to type the letter "l" rather than the digit one where it says "ll" and where it says "lo"--it will say nothing at all. The principle is "no news is good news"!

If it says nothing, it means that it perfectly handled what you typed. It means you succeeded, most likely, in making a new word in G15 PMN for this session. (If not, just restart the PC the way we said, and try over again. This is a very normal thing to do for any professional, advanced programmer--get to love the restart of the G15 PMN PC!)

So to check out your new command, your new word, type this:

```
myname
```

and as you press lineshift, you should get the PC to give you ten copies of your own first name on the computer display. I'd say that's enough for a first session, and let's complete the session while the game is going good. So type

```
qu
```

and press lineshift. At the menu, press

```
CTR-Q
```

and then type

```
       reb
```
and at lineshift, the G15 PMN will refresh itself. Next time, we will see how we can use cards, so that it can store something between our sessions. For what we type in during one session in the TF terminal isn't there the next time we take that terminal up, but that which we store in cards is more easy to get up the next time on the same PC. Of course, it may be that you are using a PC where the cards are constantly cleansed in order to keep the PC in top shape for the use of many people--but even so you can learn to use cards while you sit at the PC. Then later on you can get to work with a PC that is more your own, and use the same skills you build up here.

Part A, chapter 2: CONTROL YOUR G15 CARDS

The Personal Computer, at its best, is a sort of greatly advanced paper-and-pen type of thing. Similar to paper and pen work, you can gather yourself, compose your thoughts, by work with a Personal Computer. And it is calm and it is easy to get the same responses each time, so you get a nice sense of overview. How important isn't that in a world where sometimes our meetings with each other are full of openness and lovely--and sometimes not so lovely--uncertainties! Just like sleep is in contrast to wakefulness, and a starry night is in contrast to sunlight, so is working with pen and paper, or with a Personal Computer, or with reading a book, or making a drawing or making a painting something that provides the right type of calm in between our social life with friends. We can't be social all the time without getting exhausted--and it's much harder to be friendly when we are exhausted, especially if we are much exhausted. So a healthy good life needs these contrasts.

So, the Personal Computer is not supposed to be fluid and unpredictable and moody and someone you have to talk convincingly too. No, it is supposed to be just like pen

and paper but with this wonderful capacity it has to provide some feedback, some machine-movement, in relation to what you type into the machine.

And then there are some things the machine allows us to do easy--like sending information across distances, and keeping tracks of numbers if you run a shop, and so on: this is all part of the Personal Computer.

But, as you perhaps know, the Personal Computer doesn't do a thing unless it is given a program. And that program it carries out to the letter.

If you put one letter wrong, it can't do that program. Is that stupidty? It is neither exactly stupidity nor its opposite, smart,--for a machine isn't ever smart. We are not trying to make it smart--at least, not if we have a little education and know a little bit about what's what in life.

So, I'm not saying that these are the opinions of every one who has ever worked with computers. Some people try to define words in a different way, and argue that some machines, or programs, somehow can be smart or intelligent --and I know their arguments fairly well, I think, and I also think I understand their arguments, perhaps even better than they do. And I think they are just plainly wrong about it, but it's something that requires lots of time to think clearly about. Years, even. So I mention this in order for you to be aware that I am talking you through a view of Personal Computers that I think is right and scientific and true but I'm not claiming that this is, or always has been, the universal opinion.

Right? So I give you my take on the computer, and I think you can go a long, long way with it, and gets lots of fun with computers.

Let's get going, then. Last time I think we got the PC to display your name ten times. We typed some stuff into the terminal and when we rebooted the PC--if you rebooted it afterwards--it would have 'forgotten' all about it. Let's this time do something that stays in cards even if we reboot the PC, turn it off and on again.

So, perhaps you start up the PC just like last time, all the way into the terminal, into the TF terminal. Once you have done it, you see that bright green square and so on. It's a good way to check that you have control over the machine.

So let's do that.

Then, instead of typing in anything at once to the

terminal, we'll leave it, store something to a card or two, then get back into the terminal and tell it to 'have a look' at these cards.

Now, when you read this, you may notice that when I said 'forgot' just above, and 'have a look at' here, I put some quote signs (' or ", single or double) around these phrases. Why do you think that is?

Because otherwise we might start talking about machines as if they were humans and alive too easily. It's nice to be able to say, oh, the machine 'forgot' the program. That has a nice easy tone about it. But to be precise, we can say, that program wasn't stored on the harddisk. Such precise talk is a little boring sometimes, especially if we do it all the time. So to loosen up and be a little playful about it, we use the quotes, and when we use quotes around a phrase, we remind us that we aren't quite saying it precisely--such as when we talk of letting the machine 'have a look at' something. Even if we put a camera to the PC and makes a program go through the camera input, it doesn't mean that the PC is looking at anything at all. It is never 'looking'. It is perhaps matching patterns against patterns.

So did you quit the terminal, after you started it up? This you probably did this way. You typed

```
qu
```

and pressed lineshift. Right?

Next press a key--when we write something like <CTR> or <PGUP> or <HOME> we usually mean, there is a button on the keyboard with such letters on it, find that key and press it. So now we say, press

```
<HOME>
```

And that gets you to a card that has the menu for G15 PMN. It is, of course, card 15 in disk G, which we can also write as G15 or G:15 or, with lowercase letters, g15.

Inside texts like this, it is a good thing to write it with that : colon inside, because then we'll quickly know what type of thing it is. I'll try to remember to write such as g:15 and i:1 in this book, even though when you are typing it into the PC, then almost always you must write it without colon. But the text gets more readable this way. In some places where we have long listings, we might add something around these, like <i:1>.

Anyhow!

We are next going to put some stuff to card i:1 and i:2.
I hope that's okay with your PC? Or has somebody else, or you earlier on, stored something on i:1 & i:2? If so, I'm sure you'll find out how to put it on another place; but if you have a 'clean' machine, set up just to learn G15 PMN programming, with the TF terminal app put into it, then the i:1 and i:2 cards are usually ready for anything you like them to have. As are i:1000 and i:1000000 for that matter. And so also k:1, k:2, l:1 and l:2. Just to mention some of the many cards you can use for various things from disk C to disk L.
There are disks A and B also, but they are not available except when the PC starts up and fetches its startup stuff. So C to L are available, and disk i, j, k, l are very often used during programming and writing, and often you find that a finished program is put to disk f, right after the Terminal, which often may occupy a couple of thousand cards on disk f.
Not that you have to remember all this.

So let's go and have a look at i:1. Press
<CTR>-L
that is to say, press <CTR> button and, while holding it in, give the L button a quick push and then let go of the hold on the <CTR> button.
Then the PC should ask you about which card, or "card-id" you wish to see. "Card-id" is sometimes written just "CID", by the way, if you see those letters somewhere --you probably will, sooner or later, when you learn G15 PMN programming. Card-id means a letter and then a number, and it should be typed in with lowercase letters (even if it shows quite like capital letters in robotfont; remember that robotfont puts a dot above and to the right of a letter it considers to be a capital or big letter).
So you type into the PC, usually without colon as said,
i1
and press lineshift. Now the screen should shift and at the lower left part of it, it says i:1 or so, doesn't it? Right before that it says MENU or EDIT mode. That has to do with the <CTR>-W stuff.
Now it may be that the i:1 card is perfectly blank, but suppose it isn't--or suppose you put something there now, and you want to put something different there the next time, so you should want to know of a really quick way to cleanse--not just card i:1, but also several more, i:2, and so on, let's say up to i:15. Right? So this is a

===>USEFUL THING TO KNOW: CLEANSE CARDS FAST
To cleanse a number of cards that you wish to program on. Here's how to cleanse 15 cards starting with i:1, but you can use it to cleanse any quantity of cards, anywhere.
  Click
       <HOME>
key and get to the main g:15 menu card. If it says {EDIT} at the lower left part of it, click
       <CTR>-W
so it changes to {MENU}. Then click on the :colon where it says D:10. You get up a different menu. (The colon is used to jump quickly between cards. The arrow- or flower-like / type of thing is used to start programs.)
  Then find the place it says E:5 and click on that colon.
  Then find the place where it says E/5000 and click on the arrow or flower-like thingy between E and 5000.
  You get up a cleanser program. Be careful to type right next, or you may remove more stuff from the PC than you want to (so that the PC has to be 'reinstalled' with G15 PMN). All right? So, with attention, type
       15
for that's the number of cards. Press lineshift.
       i
for that's the disk. Press lineshift. Type
       1
for that's the start card on that disk. Press lineshift.
  It should then say DONE! Press lineshift and you can press
       <HOME>
to get back to the main G15 menu card.

So such "Useful things to know" are meant to stand out in the chapters where they occur, so you can fairly easily find them by leafing back and forth in the book. That's why there is an arrow in front of the words and the words are all in capital letter. Sometimes there will be even more than one such "Useful things" in a chapter.
  To cleanse cards doesn't mean that they become totally empty. The cards are there, and there has gotta be something on them. Even a blank, a " " type of thing, is something rather than nothing, right? You type it in and although a blank isn't shown as anything, it is still a blank. And yet, when we cleanse cards the way we just did we don't put blanks into them, but something slightly more mysterious, which is called a "nil". Think of it as more

or less the same as "null", but even less of a thing than null or zero. A nil (sometimes called a "nil character" or a "nilchar") is a highly useful thing. When you write in a program on cards, it may take more than one card. But how is the PC to know that you don't want it to 'look at' more than just a handful of cards, or however many are in your program? The answer is that it stops 'looking' for cards when it comes across this guy "nil".

So let's see. Let's go to the card we cleansed and see what it looks like. Press

<CTR>-L

and type, without any colon, and in lowercase as usual,

i1

and press lineshift.

So you see these funny signs--not quite letters, not quite numbers, more like a drawing of a square, or a piece of paper, or perhaps a folder with papers.

Now we are going to get these nil-chaps, these nil-chars completely away and fill up that card i:1 just with blanks In that way, we can begin to put our program there, the program that displays your name ten times.

So, do a

[RIGHTCLICK]

on the mouse. When we want the mouse to be used, we can put the [ ] signs around, but when we want a particular key like <HOME> on the keyboard, we use the < > signs around, okay?

The [RIGHTCLICK], that is to say, the click on the right side of the mouse, allows you to change the card--also called EDIT the card. So the word EDIT appears on the screen somewhere.

Now you can erase all of the screen quickly. First use

<ARROWS>

to get to the upper left corner, if that half-dark line, the text marker, isn't already on the upper left corner. (When we say <ARROWS> we mean any one of the four slim arrows that are together on the keyboard, often to the right of it--these we can also call <ARROW-UP>, <ARROW-RIGHT>, <ARROW-BENEATH> and <ARROW-LEFT>.)

Then press

<TABL>

also called "tabulator" or "indent" key, the key on the left on the keyboard--usually to the left of "Q", where it says eg TAB or TABL or -->| or something like that. I like to call it just <TABL>.

When you press it, you'll see that it clears away a

handful or two of those peculiar nil-chappies. Press it several times. Then press

<ENTER>

or what we call lineshift (I will sometimes call it <ENTER>).

You see that <ENTER> is really practical here, it has the same effect as <ARROW-BENEATH> followed by many <LEFT-ARROW>. Continue to use <TABL> to cleanse that line. See how fast you can cleanse every bit of the card!

Two other keys it is very handy to know about when you do anything with these cards.

The first is <END>, which moves to the end of the line. Try it.

The second is the combination <CTR>-R, which moves to the right column--about the middle, perhaps a touch to the right of the middle. This is used all the time when we program because we have room for both a left column and a right column when we put our programs on the cards, since each line is quite slim.

What else? Oh, I almost forgot. We should now save the blank card. It's ready to be used. So, when you are satisfied that there are blanks everywhere on the card, press

<CTR>-S

and type

i1

and press lineshift. The card is now saved with blanks instead of the nil's. Try press

<PGDN>

to go to the next card, card i:2. Then press

<PGUP>

to go one 'page' or card up, that is to say, to the card just before i:2, namely the i:1 that we were working at. If you used <CTR>-S correctly, you should now see a perfectly blank card there (apart from the usual text at the bottom of it which tells which card it is and so on).

So now it's time to put our program there. I suppose learning is also about repetition, so I think that we should begin by putting exactly the same program as we made in chapter 1 on the card here, and add no fancy stuff at all. Then we can adjust it a little bit and see what more can be produced with little effort.

Just one little thing we'll add. Remember that Clear Entire screen command, those two letters CE? That's really handy to put in first. So here's what I suggest we'll do next. Press <PGDN> or <PGUP> to get back to i:1.

In case the card says MENU mode rather than EDIT mode, do a

[RIGHTCLICK]

and in case the text marker line isn't at the beginning of the first line, press a lineshift then <ARROW-UP> to get it right at the beginning of the first line. Ready? Then let's type! With lineshift after each line:

```
sayit=
ce
ll:10
&yourname&
lo.
```

Be sure it's all lowercase, even though, when you type it into the card, it looks more like LL and LO than ll and lo. If it is uppercase, these letters get a dot above them. That's how uppercase looks in the robotfont!

This time we call our new word 'sayit', if that's fine with you. The rule for choosing new such words is that it is short, more than two letters, lowercase, without funny signs, and that it isn't already been given a meaning in the TF terminal (we'll look into a way to check that fast). But you can call your new word abla or difta or progeny or what you want, as long as you remember what it is when you are going to start it up. Note that digits are okay in these little program names, when not in the first position. So hi3 is fine, 3hi isn't.

Be sure now to save the program card back to the i:1 position very beautifully. Here's how:

<CTR-S>

then, as it asks where to save it, type, without colon,

i1

and that's it. Ready to try the program? Then we must, the way we have set it up now (we'll look at ways to set it up for even faster starts), first tell TF where the program starts--at i:1--then we must write 'sayit' or whatever you called the program. Ready? Super. Then press

<HOME>

and click

<CTR>-W

and click on the :colon between H and 1 on the top there. That gets you to the h:1 menu, where I think is where you start the TF terminal, though we can set it up in many other ways. At the menu H:1, click on the F/1 thingy as you by now have done several times if you have done all what we have said so far in this book, and the terminal starts up with its little graphics squares and so on.

Now you have the terminal up, and we're now going to tell the terminal to go through the cards--or the card, in our case, since it's just one card. You must then find, on the keyboard, the ^ sign which is usually found by the combination <SHIFT>-6, since it is usually above the digit 6. If you press other things, just press the <== key, which we also call <BACKSPACE> or just <BSPACE>. Ready? Then type:

```
^i1
cc
```

with lineshift after each line. If you have done it right, it will say a large YES! In case it says something else, just go back and have a look at the program and try again. (Then you press lineshift, type qu and press lineshift, and do a <CTR>-L where you type i1 and press lineshift. Check the program, go back to the terminal and type it carefully once more. You'll get it right! And keep spirits up no matter what the PC says. It forgives instantly, and forgets all mistakes completely and forever.)

If you called the program gringring or propanol or slartibart type it in, otherwise, if you called it just "sayit" as I suggested, type it in:

```
sayit
```

and the screen should get completely blank and the name you love the most of all appears ten times in a nice font.

Restart the PC completely and go straight into the terminal and type the ^i1 and the cc again, and when you type 'sayit' or whatever you called it, it does it again.

And unless this is a PC that somebody cleanses afterwards, it will work tomorrow also: for cards don't require electricity to keep on their data.

And in getting the PC to do this, you have learned loads of things--loads of them. It's time to congratulate yourself and be in a mood of celebration and take a break and feel the joy of the full command you are having over your personal computers, your G15 PMN computers.

Part A, chapter 3:

Before each programming lesson--if 'lesson' is the right word for these chapters--let's open our minds a little bit and just relax and take in the broader picture.

Sometimes, when you program, you may feel that all the world is in order--because you find that the computer responds perfectly to what you type into it, and what you type into it reflects perfectly what you think. And maybe you get a kick out of that. A bit of self-confidence. And when you then leave the machine and go out and enjoy yourself with people, you may feel that you have a glow inside, a kind of 'carrier wave' or inner force. Yes, programming can give you this.

Now there is a really interesting type of study done by an austrian named Asberger a long time ago,--he found out, put very simply, that some people aren't much tuned into other people's intentions--what they want, you know, and why they move about the way they do, why they gesticulate the way they do--and that just these people may be great at doing something which requires great concentration to detail. That's a simplification and I blend in some later studies in my summary there.

Instead of--as maybe Mr. Asberger did--thinking of this as some kind of trouble, I would call it a personality mode, or a person mode,--just as we can switch between Edit mode and Menu mode with the cards. Sometimes you just ignore other people's intentions and at other times you are really tuned into other people's intentions--you seem to know what they want in a flash. So, the question I think is cool is this: what if somebody is a little bit stuck in one mode and find trouble switching to the other mode? You follow? And how can programming help?

First, let's imagine somebody who is a little bit stuck in not tuning to other people. That person will probably have an easy time concentrating on the computer. How can he, or she--let's say "she", it's more positive--how can she learn to understand other people's behaviour more, by programming?

Well, just think of what it is to program: you put in a program and if it is a bit long, chances are great that the PC will do something other than what you planned. And to fix it, you must look at the behaviour of the PC, go back to the program and see and look and understand and then you fix it. So the PC has a behaviour and you get to learn to think through what lies behind that behaviour. As

you get good at programming, you get self-confidence, and that--as said--helps you to be great social, too. So that takes care of that. Admittedly, it's a long way from thinking about the behaviour--or activity, to be more correct in language--of a machine, and that of a human being with a living mind. But it can be seen as a beginning--not all things about why people do what they do is so complicated, right?

The other way, those who are great at understanding other people's behaviour--they catch their plans without having to have them explained, all that--they may however find difficulty concentrating. So when the PC is in front of them, demanding nothing, it can help produce calmness. And their understanding of other's motives, plans, goals, intentions, means that they are good at thinking through this kind of stuff--and then all they have to do to learn programming is to translate plans into a series of steps that fits what the PC can 'make sense' of. So he, or she, --again, let's say "she"--she will learn to program by admitting to herself that she's already thinking through quite complex things on behalf of other people. The PC is much more simple. And the value of being patient with the PC and pick up its language is the great sense of calmness and the self-confidence and relaxation it can produce, among other things.

Then, for both types of persons--those who are mostly social, and those who are more, shall we say, self-centered--work around a PC can very easily become social. It can become a bridge, to sit near the PC and help each other and sometimes show off to one another.

Anyway, let's begin on some work, to get on with the next step. Last time we actually got a program put to cards--some would say to 'file'--and we got the PC to 'have a look' at these cards and follow the instructions on them.

Some would call this 'to compile' cards. We got the PC to 'compile' the program. That's a way to put it. If you like, the two 'c's we used--remember we wrote, into the terminal, something like ^i1 and then 'cc' on the next line? We could maybe think of this as Compile Cards.

These two-letter, and sometimes one-letter, words or what we should call them--commands, maybe--are also called pre-defined words, or PD words. "Pre-" means 'already'. And "define" is what we do when we tell the meaning of a word. It isn't as though we cannot make new PD words. We can--but that's hardly necessary before we are doing

really huge programming projects, like an entirely new type of game that nobody has thought about before. You can do huge amounts of programming without making new two-letter commands. But to make new words of three or more letters, that's what you have already begun doing.

Let's keep it freshly in our minds what we've just been through--when it comes to putting stuff to cards and getting the PC to 'have a look' at it. So could you get up card i:1 again? You start up the PC, mount (with mnt) the 3333333 thingy--if it is not already mounted in the way it is set up for you--and you press <CTR>-L and type in

```
i1
```

Got it? And, to be sure it is in Edit mode--so we can change what's on the card--do a [RIGHTCLICK] on the mouse. (In case somebody else have been messing about with your cards, go back and do the cleanser thing that we explained in the previous chapter--'useful thing to know' was the heading inside that chapter, right?)

So let's take away what's on that card, use <ARROWS> and put the text marker line on the uppermost left position, and then press <TABL> key to blank out a lot of them at a time. (Remember that key? The -->| type of button on the left side of the keyboard.)

I have in mind that we first get some numbers out of the machine, then do something with those numbers to change them into bright little green stars on the screen. How about that? So let's have a look on this first--just numbers--let's call it 'ournums' or something:

```
ournums=
ll:15
i1
nn
lo.
```

Now before you type this in, let's see how similar it is to our earlier program. It begins with a new word, which then has an equal-sign (=) in it. The last word finishes with a dot after it (.). The dot, when typed into the cards, is a whole little block of black on the screen, but when inside a book it's typically just a little spot. So you see the cards editor--also called CAR or CAR editor (because we drive the cards a little like we can drive a car)--is really friendly towards our programming. It will be very clear at once whether we have typed in, or for-

gotten about, such a thing as the final dot in each little program or, as we also call them, our 'functions'.

Anyway. More similarities? Yes, you can see that again we have the two letters ll in the first line after the new word, and lo on the last line. That means we're into repetition in this little program also. How many reptitions? 15 this time, not just 10.

Actually, last time we had a 'ce' to clear the screen on the first line, just before the ll: stuff. But it isn't really necessary here. Let's keep it all, this time, as simple as we can.

So we'll first get this program to produce the numbers 1 to 15. Then we'll calculate them up to become 10 to 150. Having done that, we'll produce some bright-green stars on the screen using these numbers--just a very simple line of very tiny dots, nothing fancy or varied this time (we'll come back to how we can vary it). We're gonna make a vertical line of bright star-like dots.

You see there is a two-letter word, or command there--that we haven't used before--nn. Just as both pp and b9 can be used to print tiny texts on the screen, so can nn be used to print a small number. To 'print' here is just a way to talk about putting something to the PC screen. Why we sometimes say print is, I suppose, because in the first days of computing there weren't any displays, just printers making a lot of noise putting some digits and letters to paper each time the computer was going to give some results. So 'to print' means the same as 'to show', when we speak compu-lingo (unless we are really talking about connecting a printer to the computer and using it).

Just before the nn, by the way, is a new little command which has nothing to do with cards and all to do with the ll: and lo type of looping or repeating something, and that's i1. This means, let's have the count, or what is also called the "index"--for the first repetition. (So if we have several on top of each other, we use i2, i3 and so on up to i9 and then, the tenth, ix. We'll come back to that.)

So the 'i1' on one line puts a number 'on the table' as it were--also called, 'on the stack'--and then the 'nn' picks it up again and shows it on the screen. We'll see enough examples of this soon, so you'll understand it fast enough, don't worry about it. It will become simple.

Alright. Do you mind typing this in? It would be nice if you can train your fingers to get used to this type of stuff more and more; that's part of programming--not just

to think with the head and your heart, but let your fingers learn to dance on the keyboard.

For the moment, just type it in from the top left of the card. We'll look into ways of typing with blank lines and the use of the second column soon enough, so that it gets easier to change the program.

When you're satisfied, save the card as last time: Click

<CTR-S>

and, with a lineshift after, type in

i1

and then, to be sure, press

<PGDN>

to have a look at i:2. That card should be empty. (If not, get it empty by the way we showed last time. There are other ways we can do that also, we'll come to that.) And it's almost certainly empty if it's full of these small squares with a sort of 'thumbs up' on its right side (some stuff can 'hide' inside such signs, though, and we'll come to a way to check that out).

Then you could press

<PGUP>

and have a look at i:1 again. Does it look fine? No extra blanks? No sudden uppercases? Letters where there should be letters--the small l (L, not the digit 1), for instance --before the colon : and before the digits 1 and 5, on the second line?

When you're satisfied with how the program looks--the compu-lingua word for it--that is to say, the computer technical word for it--is "syntax"--a word used also in grammars when it comes to spellings and such--when you're satisified, then, go into the TF terminal and start it up. So, press

<HOME>

and press

<CTR>-W

to get to Menu mode. Click on the colon inside H:1 on top there. In that h:1 card, click on the line in F/1 there. That starts it up, as you know. In the terminal, type

^i1

cc

How did it work? If you typed it perfectly and saved it, the computer will say a big bright YES! in upper left corner (something we didn't clear away with a 'ce'). Then you can type whatever we called our little program this time, ournums. So type:

ournums

and press lineshift. What the screen should show, then, is

```
  1
  2
  3
```

and up to 15.

Does the program work? If not, you have the fortune of learning some program correction. For all I know the PC might have stopped completely because something was typed in differently. Suppose you left out the dot. Suppose you didn't put in the 'lo'. Stuff like that. Perhaps you have to type some rubbish and press lineshift many times to get any response from the PC again. Maybe you even have to push the power button (perhaps even hold it in for a good many seconds) in order to restart the PC. That's something you should do in excellent spirits and with good humour.

The golden rule of thumb of program correction is this:

THE MORE ABSURDLY WRONG THE PROGRAM IS, THE EASIER IT IS TO FIX iT.

Whatever is the case, you should at some stage be able to type qu or so, and get back to i:1 card and fix it up and get it to run. Have you done it? Then let's multiply the number by ten. For this we just put in the number '10' and the multiply command, which is--you can just about guess it from our earlier commands, pp and nn,--yes, it is 'mm'.

So, get back to card i:1 again if you don't mind, and rewrite it to this,--and we'll soon enough produce some star-like dots--or 'pixels' as they are called--on the screen from these numbers. Here we go:

```
       ournums=
       ll:15
       i1
       10
       mm
       nn
       lo.
```

Save it to i:1, start up the terminal again in its h:1 card, and do the ^i1 and the cc and type

```
      ournums
```

and now we get:

```
  10
  20
  30
```

and all the way up to 150. Note that you could write, if you like, a million instead of 10 there. And you could

write ll:100 instead of ll:15, and the PC is equally happy. (I should say, 'happy', sometimes I forget to put in the quote-signs.) But take care once a number gets larger than twice a thousand million--that is to say, take care when numbers get bigger than two billion. That's about the size of what the PC 'likes' to handle. Up to 2,000,000,000 (usually without those commas although we can instruct it to put in those commas), and a little more, and the numbers can go into the minus region as well.

So though we begin learning programming with really small and neat numbers, it doesn't mean that we aren't also learning how to put the PC to hard and complex work with digits that are thousands or even millions of times bigger. When you learn to handle inputs from keyboard and mouse and output to the screen, and storage to cards and getting stuff from cards into a program again, and also sometimes connect to other machinery from a PC, then you can, by equally simple commands--though usually many of them--get a PC to do anything that a computer can do. So these little things are the nuts and bolts, the alphabet, we might say, that we need to go through to build up our programming language competence.

There are many ways to program, and there are many ways also to make programming languages. A guy named Charles "Chuck" Moore worked with simple computers in the 1960s to control motors connected to such as telescopes to look at the stars. And he, more than anyone else I know of in computer history, pioneered the work on using a 'stack' of numbers for the simple commands of a programming language to 'speak together'. This enormously elegant way shines all the way back to the work he did in the 1960s, although many things had to be made better for a really good and useful and easy programming language to result form it in the long run. In the making of G15 PMN we have learned from many places, but the heart is with the impulse from Mr Moore.

Let's next remake the program so it puts dots on the screen. Now I'm typing this into the B9edit editor, and I have the benefit of just copying what we have already programmed earlier as 'somenums' and putting it next here, and modifying it. Have a look at this:

```
ourstars=
ll:50
800
i1
```

```
       10
       mm
       px
       lo.
```

So I changed the ll:15 to ll:50. That's exactly the same type of command, just 50 repetitions instead of 15. So we get some more of these star-like dots we want to produce.

Then there is the number 800. That can be any number you like from about 0 to about 1023, and it goes from the entire left of the screen on the G15 PMN machine to the entire right. 800 means much to the right but not at all on the edge of it. In fact, it's about where we usually type stuff into the terminal.

What else is new? The name, 'ourstars'. (Call it what you want, of course, as usual, as long as it is unused.)

And then the 'px'. Now the 'px' command means 'pixel'--in other words, a pixel should be put on the screen at the place where we want it. It expects that 'on the table', as it were, --or on the 'stack' of numbers, we have two numbers. The first tells the left-right position and the second--the one which we calculate by means of our '10' and 'mm'--the multiply by ten thingy--is the vertical position. Left-right numbers go from 0 to about 1023. The vertical numbers go from 0 to about 768. That's how a G15 PMN screen is defined. Once you know that, you can compose the screen as you like it. Some programming languages try to handle any type of screen but it is my experience that the stage of the 'dance' of the program must be known for our programs to be looking good and easy to think with and work with.

So px expects two numbers. We give it 800--that means on the right side, about--and then we give it a number that varies from 10, via 20, 30 and so forth, up to 500. So both of these numbers make sense to give to px. Try the program!

First, you put it to card i:1 the way you know--get up the card by <CTR>-L and then type i1 and press lineshift. Press [RIGHTCLICK] on the mouse, wipe out what you want to whipe out with such as <TABL> key, and type it in. Then press <HOME> and <CTR>-W and get to card h:1 and click on the place you click there to start the terminal. In the terminal, type

```
       ^i1
       cc
```

and then type

```
       ourstars
```

with lineshift after each line. With luck, you get a row of bright little dots, vaguely like shining stars, right where you wrote 'ourstars' and underneath it, in that column.

Fix the program if it needs to get fixed. And now you have entered the world of graphics programming and you deserve to have that glowing feeling which comes from having set a goal and achieved it.

[STAY TUNED!!! MANUSCRIPT UPLOADED AS IT GETS FINISHED!]