

CROS COMPILATEUR DE TEXTE.gco

Par Nulentout : Mardi 6 octobre 2020.

Soyons honnête : Ce projet un peu fou n'a vraiment été motivé que par le plaisir de programmer. Quand la pyrograveuse décrite sur ce site a été achevée et le didacticiel mis en ligne, je me suis retrouvé un peu tristounet, comme c'est souvent le cas quand on vient de passer de très agréables vacances et qu'il faut reprendre le joug. Une sorte de "vide", car ce n'est pas les activités ludiques qui me manquent. Mais ... la programmation est une drogue, une activité qui oblige à faire des efforts intellectuels sans lesquels, à mon âge le cerveau se rouille. Bref, je cherchais un prétexte pour rebrancher un Arduino sur le P.C, et des raisons de me creuser les méninges.

Pyrograveuse achevée, la petite machine a été mise à contribution pour divers petits travaux. En particulier, tracer comme décrit dans le didacticiel des pénoms sur des livres pour en préciser le propriétaire. Changer de prénom devient rapidement en une routine un peu (*beaucoup !*) indigeste. Sur une feuille de papier quadrillé on trace les lettres, on positionne l'origine, on détermine les coordonnées des points. Scongregneugneu, c'est d'un fastidieux !

Imaginez qu'à Noël vous avez prévu pour les dix personnes qui seront à table une petite étiquette en bois pour situer leur place dans la salle de séjour. Plus un petit mot gentil sur chaque plaquette. Par exemple "Joyeux Noël" exige déjà 136 lignes de codes, soit environ 230 valeurs à déterminer pour les divers points à tracer. Une grandeur fausse, un oubli de faire déplacer au lieu de tracer, et il faut reprendre l'analyse du programme. Une rapide évaluation prévoit pour nos dix invités environ 3000 à 4000 points dont on devra calculer les positions.

Domage, l'idée était séduisante, mais il faut se rendre à l'évidence, c'est proprement impossible, complètement déraisonnable.

Youpiiiiiii, le voilà le prétexte pour tortiller du code C++, et y consommer des heures par paquet de beaucoup ! Une idée toute simple qui consiste à frapper du texte sur un clavier d'ordinateur, puis de le valider. Et prouitchhh, un programme avale et gloutonnement ce verbiage et recrache du code gco qui sur la pyrograveuse tracera le texte en question. Génial non ?

Il reste à définir sur quel système sera hébergé ce logiciel miracle, et sur quelle machine le loger. Comme je ne suis plus séduit par la programmation en LSE, BASIC, PASCAL, FORTH, PROLOG, tous ces langages que j'ai trop longtemps pratiqué, et que je désire surtout conserver mes connaissances en C++, c'est tout naturellement que j'ai pensé à créer un "CROS COMPILATEUR" à base d'une petite carte Arduino NANO. Une homogénéité parfaite avec la pyrograveuse qui sert de cible. La petite carte dialoguera de façon classique avec le Moniteur de l'IDE, transformera le texte qui lui est soumis en Image.gco et restituera le résultat dans la fenêtre du Moniteur. Ensuite, par du simple Copier / Coller on recopiera le code dans un fichier de type ImgN.gco et vogue la galère.

Fig.1



T'as vu ce container marin de la société
Compilateur ? Ils doivent être très écolo, car il est
tout vert leur conteneur.

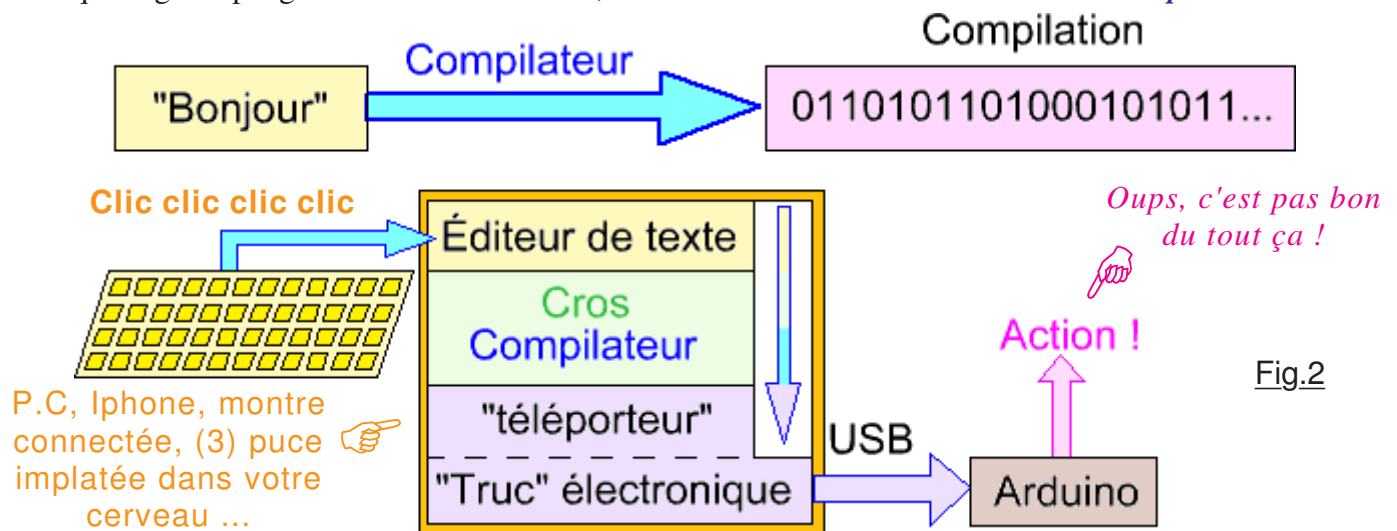
1) Compilateur, cros compilateur.

S'il est un mot technique qui n'a pas eu de chance dans sa vie, c'est bien le terme **compilateur** bafoué depuis des années à tel point que rares doivent être les personnes qui seraient capables actuellement d'en donner une définition précise. Et bien non, la compile, ce n'est absolument pas la sélection musicale effectuée par le "didgé" pour animer en "laïve" une soirée dansante.

Un petit retour en arrière s'impose. Remontons aux tout début de la naissance de ces monstres que l'on nommait des ordinateurs. Ils se programmaient en binaire, une sorte d'enfer qui se résume à des zéros et des uns. Les ingénieurs logiciels se sont rapidement rendu compte qu'il serait "impossible" de continuer à bégayer binaire toute leur vie. Aussi, ils ont donc rapidement compris que ces grosses machines pourraient un jour lire du "bon français" compréhensibles par les humains, et transformer ce "texte" en des séries de "0" et de "1" assimilable par les circuits électroniques des grosses machines.

- On définit un langage très rigoureux que peut pratiquer l'humain **facilement**, (1)
- On loge dans une machine informatique binaire un programme qui traduira ce texte "humain" en binaire pur directement "électrifiable" par le truc ordinatOMICROprocesseur, puis la machine cible **effectue le travail prévu par le programmeur**. (2)

C'est précisément **ce programme qui translate du texte en programme binaire pour un processeur cible** qui **se nomme un compilateur**. **Le résultat binaire** qui par un moyen quelconque sera téléchargé sur le processeur cible **se nomme une "compilation"** ... et ça n'a rien de musical ! Si c'est un ordinateur dont le processeur est totalement différent de celui de la machine cible qui loge le programme de translation, alors ce dernier se nomme un **cros compilateur**.



Par exemple, sur la Fig.2 l'**IDE** (*Ensemble jaune, vert et violet.*) est un système de développement de programme qui vise comme cible un microcontrôleur spécifique nommé ATmega328 logé sur une petite carte électronique connue sous le vocable **Arduino**. L'**IDE** comporte un petit traitement de texte intégré, un analyseur syntaxique pour vérifier que vous avez "parlé avec rigueur" et qu'il peut comprendre votre phraséologie. Puis, votre texte est alors compilé, et la suite de "0" et de "1" qui en résulte est téléversé par des intermédiaires électroniques dans l'ATmega328. Le microcontrôleur est alors "branché" automatiquement sur votre logiciel qu'il va exécuter avec une fidélité absolue "vos lignes de code". C'est là que deux remarques s'imposent :

- (1) : Complètement faux, ce n'est jamais facile de créer un **programme SOURCE** !
- (2) : Le microcontrôleur effectuera avec rigueur les instructions du **programme OBJET**, toutefois, on constate très très très souvent que ce n'est pas du tout ce que prévoyait le programmeur !

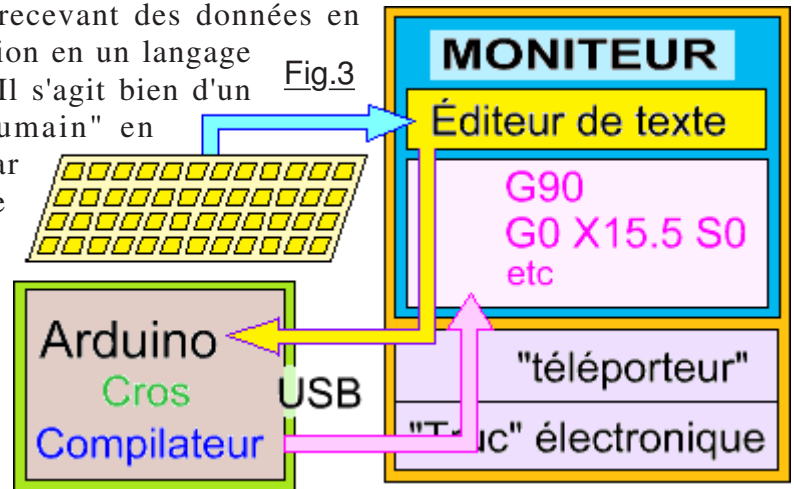
(3) pas pour Mômôa la puce électronique dans ma petite cervelle !



Il est vraiment comique l'auteur de ces lignes. Chaque fois qu'il a une idée simple, il croit que ce sera plié en deux ou trois heures. Et puis, quand il veut la concrétiser ça devient démentiellement diabolique !

2) Architecture logicielle de notre petit projet.


Banalement ordinaire, le programme sur décision arbitraire sera écrit en C++ et sera logé dans les circuit interne d'un ATmega328. La carte électronique sera une Arduino NANO pour des raisons de faible encombrement et de disponibilité. C'est pour avoir le plaisir de programmer avec l'**IDE** que nous avons décidé de loger le compilateur à l'extérieur du P.C. sur lequel nous aurions aussi bien réalisé le programme en langage Python par exemple. C'est un choix totalement arbitraire. Nous allons réaliser un programme qui recevant des données en langage "humain" effectuera la transposition en un langage "machine" que l'on nomme langage *gco*. Il s'agit bien d'un compilateur car il traduit du texte "humain" en "programme" directement assimilable par une machine pyrograveuse. Comme ce programme n'est pas écrit un *gco*, c'est donc un *cros* compilateur. La chaîne de programmation subit une petite mutation montrée en Fig.3 car le compilateur n'est plus interne à la machine (*Ici il s'agit d'un ordinateur P.C.*) qui établit le dialogue avec l'utilisateur.

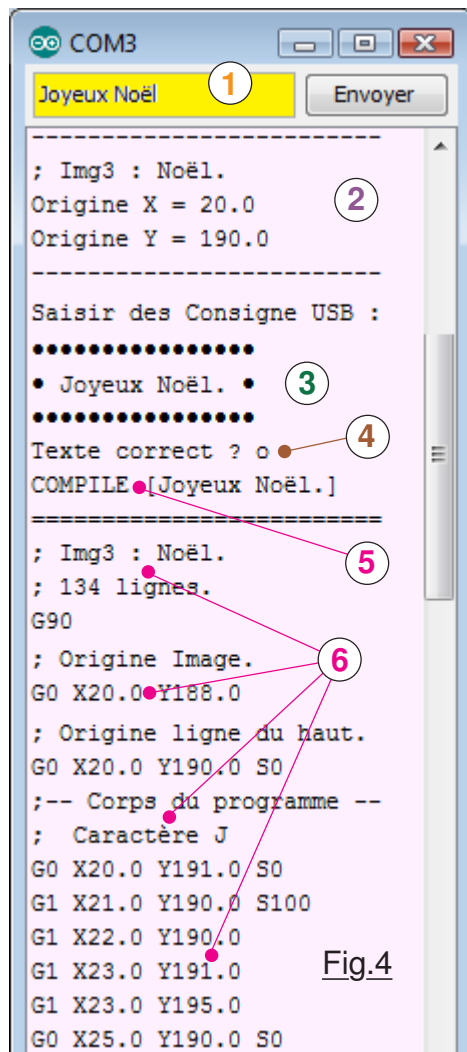


➤ Protocole d'utilisation du système.

Dialogue Homme / machine oblige, on ne limitera pas lors des échanges entre le P.C. et Arduino à fournir du texte et attendre en retour un résultat de type "lignes de code *gco*". Les possibilités opérationnelles seraient d'une pauvreté tristounette. Le dispositif présentera des options séduisantes comme le choix de différentes tailles de texte, possibilité d'encadrer etc. Qui dit options, implique

la faculté d'envoyer des consignes à Arduino, ce dernier accusant réception, comme tout système électronique dialoguant avec un utilisateur, signalant des erreurs, des problèmes potentiels ...

L'opérateur a branché la carte Arduino NANO contenant le programme d'exploitation du compilateur sur une ligne USB de l'ordinateur. Puis, on active l'**IDE** en ouvrant un nouveau "Sketch" ou un programme *.ino* quelconque, le but étant d'activer le **Moniteur** avec sa commande . Vitesse de transmission correcte sur la ligne USB, le dialogue commence et peut ressembler à ce que montre la Fig.4 qui résulte d'une copie d'écran retravaillée pour en minimiser la largeur. Dans la ligne de saisie 1 qui constitue un éditeur de texte élémentaire, nous avons proposé un texte à compiler. Suite à chaque saisie un petit rappel des caractéristiques de l'image *gco* qui sera générée est affiché en 2. Comme nous avons proposé un texte à traduire, le programme d'exploitation affiche en 3 ce qu'il compilera et nous demande en 4 de confirmer. Comme nous avons accepté avec la lettre 'o' pour OUI, le logiciel très bavard nous précise en 5 ce qu'il va tenter de faire. Puis en 6 le compilateur est activé et retourne le résultat de son travail, à savoir des lignes de code *gco* qui sur la pyrograveuse traceront les lettres qui constituent notre texte. (*Lettres, chiffres, ponctuation, accentués, tout ce que la police de caractère est capable de générer.*)



RÉSUMÉ : Après avoir invoqué l'**IDE**, on active son **Moniteur**. Puis, dans la petite fenêtre de saisie qui se résume à une ligne en 1 on impose des consignes ou on frappe un texte à traduire. Le **Moniteur** accuse réception dans sa grande fenêtre de dialogue et y affiche ses comptes rendus, ses réponses etc.

3) Matériel investi dans le compilateur gco.

Mise à part la carte **Arduino Pro Mini** qui est légèrement plus petite, et encore en longueur, mais légèrement plus large, le petit circuit imprimé Arduino NANO est le plus modeste en dimensions de toutes les références Arduino. Comme la NANO est disponible à des tarifs similaires ou inférieurs, il n'y a pas à hésiter, et ce d'autant plus que la **Pro Mini** ne dispose pas de prise USB. L'agencement global contribuant à minimiser le volume qui sera occupé par l'ensemble de l'électronique, au final le coffret du petit compilateur montré sur la Fig.5 ne mesure que 62mm de long, 26mm de large et 35mm de hauteur, feutres situés sur le dessous et bouton de RESET compris. C'est presque la ligne USB qui se branche sur l'ordinateur de dialogue qui prend le plus de place. Les photographies de la Fig.1 et de la Fig.5 sont trompeuses, en apparence le boîtier semble gros. Dans la pratique ce sont des images saisies en MACRO et le coffret (*Voir la Fig.35.*) est vraiment peu encombrant. Il suffit de savoir que les vis visibles sur la Fig.5 sont au diamètre nominal de 2mm, donc très petites. Le schéma électronique retenu et présenté en Fig.6 est très complexe puisqu'il n'ajoute à la carte NANO qu'une résistance et un BUZZER actif. Vu le peu de composants mis en œuvre il serait possible de se passer de circuit imprimé support. Ceci dit, comme une toute petite carte de prototypage sera suffisante, cette dernière assurera l'immobilisation de l'ensemble électronique dans le coffret. La sortie **D4** a été choisie pour

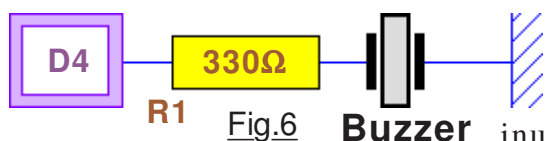
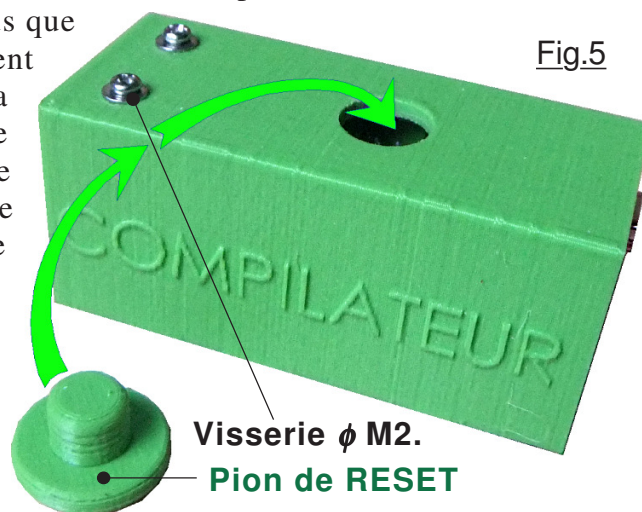


Fig.6

Buzzer

homogénéité avec la carte de la pyrograveuse. (*Critère faible.*) Le BUZZER est de type actif, c'est à dire qu'une tension d'environ 5Vcc génère directement un signal sonore, inutile sur Arduino d'avoir recours à de la PWM. Si il est

directement branché sur **D4** je trouve que les BIPs d'alerte sont bien trop agressifs, la résistance **R1** diminue le courant envoyé au transducteur lorsque la sortie est à l'état "1". À vous de modifier à votre guise cette valeur, sans toutefois dépasser le seuil pour lequel le fonctionnement ne serait

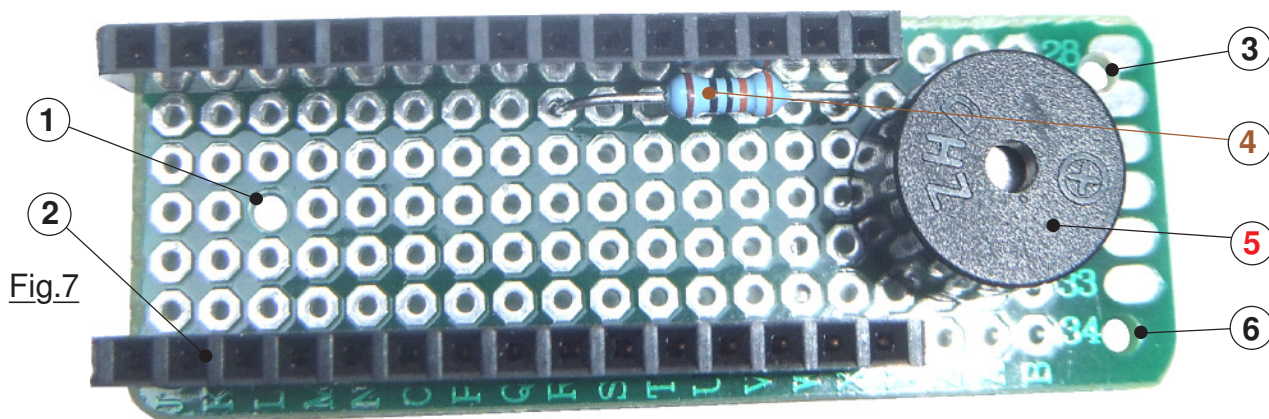
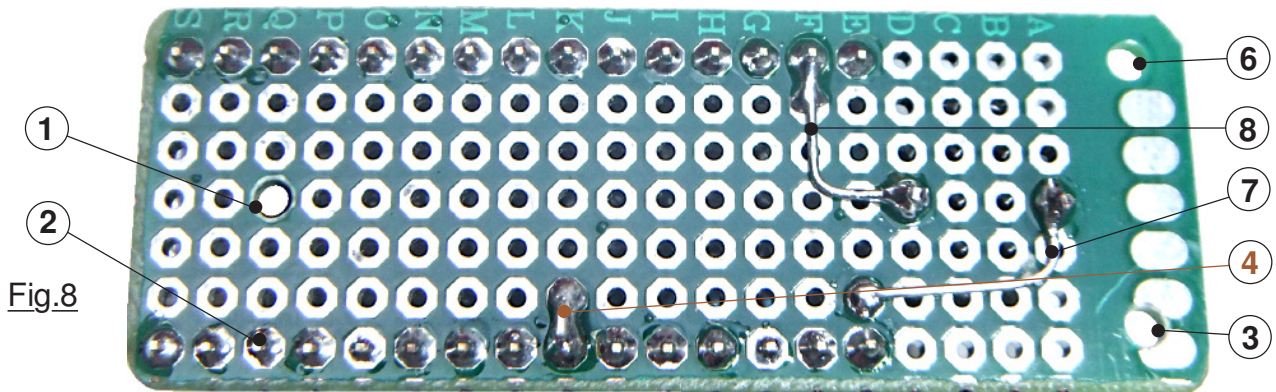


Fig.7

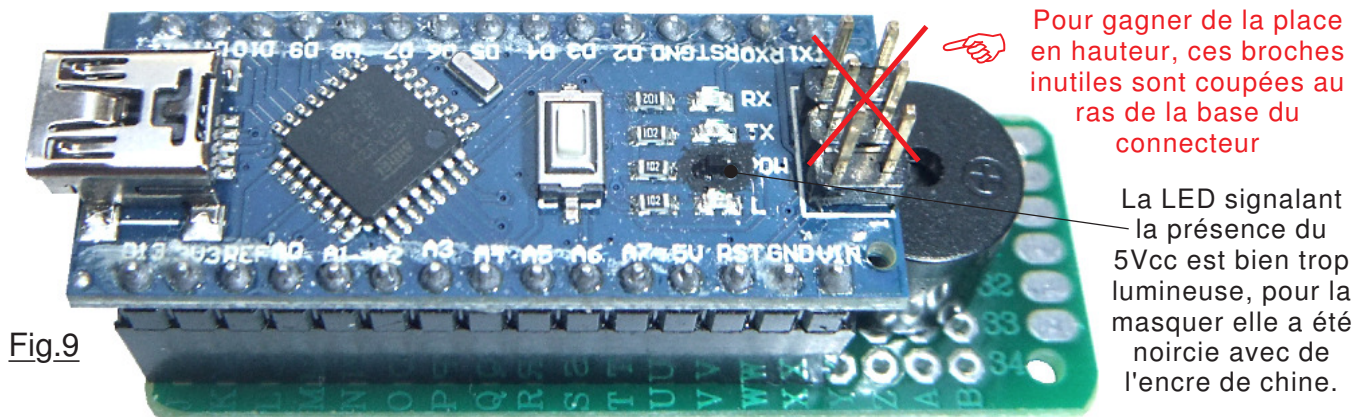
plus totalement dynamique. Sur la Fig.7 la carte Arduino NANO n'est pas installée sur les deux lignes de barrette HE14 **2** qui servent de support. On peut penser qu'il s'agit d'un luxe couteux. Toutefois, vu que j'approvisionne ces barrettes par lots, leur prix de vente est pratiquement dérisoire. Par ailleurs, la carte Arduino qui a été installée sur ce circuit me sert à développer des logiciels depuis des années. Aussi, vu le nombre de téléchargement faramineux qu'elle a enduré, il n'est pas exclus que l'on finisse par dépasser le seuil de fiabilité. De plus, cette carte a subi un incident électronique, la diode d'alimentation de protection sur l'entrée 5Vcc a ... grillé. Aussi, si cette pauvre petite carte fidèle finit par rendre l'âme, la remplacer facilement sera rapide, alors que dessouder les 30 broches serait pratiquement infaisable sans y laisser la santé nerveuse. Par ailleurs, comme la carte Arduino est surélevée, le buzzer "passe" dessous ce qui autorise un circuit le plus court possible. En **5** on retrouve le BUZZER **polarisé** avec sa résistance en ligne **4**. Le circuit imprimé est supporté en trois points dont les trous de passage des vis ϕ M2 sont bien visibles en **1**, en **3** et en **6**.

➤ La vue coté soldures.

Fait assez rare dans mes réalisations pour qu'il soit souligné, ce petit circuit imprimé n'est soudé que d'un seul coté ce qui en facilite les opérations d'assemblage. Il me semble inutile de "rabâcher" ici les divers conseils habituels pour réaliser les soldures, d'autant plus que des chapitres dédiés sont étalés en pages 14, 15 et 19 du didacticiel sur la pyrograveuse. Quand à rédiger une fiche de câblage pour trois connections à établir, autant oublier, la Fig.8 qui reprend certains repères employés sur la Fig.7 est largement suffisante. On soude la résistance **4**, on ajoute le BUZZER puis




les deux lignes HE14 telles que celle située en **2**. Pour réaliser la liaison entre la résistance **4** et le picot de la ligne HE14 **2** je me suis contenté de faire un "paquet de soudure". On continue par la liaison **7** qui dans la pratique est constituée de la queue de **R1** pliées au ras des pastilles de la sérigraphie et coudée à angle droit pour aboutir au picot du BUZZER. Enfin on complète par le fil de liaison **8** constitués par la queue de la résistance **R1** qui a été raccourcie en **4**.



Puis de façon classique on vérifie qu'aucune liaison accidentelle ne réunit deux picots voisins du HE14, et que les liaisons prévues soient effectives. On applique +5Vcc entre la lyre pour la broche de **D4** et **GND**, le BUZZER doit couiner. On peut enfin mettre en place la carte Arduino NANO.

➤ Programmation et mise en service de la carte Arduino NANO.

P phase incontournable qui va donner vie à notre compilateur elle comporte plusieurs étapes qui sont élémentaires, à partir du moment où vous avez réalisé la pyrograveuse. De ce fait l'**IDE** est installé sur l'ordinateur et vous avez déjà établi des liaisons USB et effectué des téléversements sur une NANO. Commencez par mettre l'**IDE** en service et brancher la ligne USB. Validez la prise concernée sur l'ordinateur. Activez le **Moniteur** avec sa commande  puis imposez une vitesse de 115200 baud. Sélectionnez le mode "Retour chariot". Puis téléversez le programme de servitude **Police_de_caracteres_en_EEPROM.ino** et activez une deuxième fois le **Moniteur**. La fenêtre d'écran liste alors l'intégralité des données figées en mémoire EEPROM, code binaire qui constitue une partie de la police de caractères qui sera utilisée par le programme d'exploitation que l'on va maintenant inscrire dans les circuits de l'ATmega328. Dans ce but, téléversez à son tour le programme **Compilateur_pour_TEXTE.ino** et notre carte NANO va pouvoir bavarder avec le P.C. qui se trouve connecté à sa ligne USB. Activez une troisième fois le **Moniteur** qui se présente et affiche comme visible sur la copie d'écran de la Fig.11 un rapide rappel des deux commandes principales. Nous allons sans plus tarder nous faire plaisir et passer en revue les commandes de base et découvrir les options principales qui procurent une grande souplesse d'utilisation de ce compilateur. **Page 5**

4) Un trois fois rien d'historique relatif au développement du programme.

Fidèle à mes "traditions", lorsque je propose un didacticiel quelconque sur la toile, ce dernier comme ses prédécesseurs est accompagné de quelques fiches techniques que l'on peut imprimer recto / verso et pour ceux qui possèdent une plastifieuse, à protéger par du film transparent. Ces fiches sont généralement très commodess, et ce tutoriel en apporte six qui à mon avis *étaient* absolument indispensables. *En développement logiciel, la plus grande difficulté rencontrée lors de cette étude concernait la grandeur des tables de données qui permettent de définir la police de caractères.* Ces octets empiètent directement l'espace réservé aux variables dynamiques à tel point qu'à certain moments une "collision de PILE" a perturbé le fonctionnement du logiciel. *Les textes affichés sur le Moniteur pour établir le dialogue homme / machine impactent également la RAM dynamique, aussi, il faut les limiter au stricte minimum.*

Prograver du texte consiste à déplacer le LASER comme montré sur la Fig.10 avec des mouvements où il grave, *et des décalages où il est coupé.* Pour tracer cette lettre 'A' on va de **1** à **2** en gravant, de **2** vers **3** en traçant. Puis on déplace de **3** vers **4** sans activer le LASER. Enfin on termine la lettre en traçant de **4** vers **5**. Pour définir cette lettre 'A' il faut donc préciser quatre coordonnées avec leurs valeurs de position sur X et sur Y, et préciser si le déplacement se fait "à vide" ou en traçant. Ce sont ces données qui vont imposer la rédaction de tables boulimiques en octets de mémoire. Pensez que pour une lettre comme le 'B' par exemple, c'est onze déplacements qu'il faut décrire, donc 22 octets pour les valeurs sur X et sur Y. La police de caractère intègre les majuscules, les minuscules, les accentués etc. Une banque de donnée "octivore". Aussi, pour utiliser au mieux les ressources de l'ATmega328, son EEPROM a entièrement été gavée par une première partie de la police de caractères. Puis le complément a été intégré au programme *en forçant l'implantation de la table hors mémoire dynamique.* Les textes qui sont également goulus en consommation de mémoire dynamique ont été *initialement* réduits au minimum.

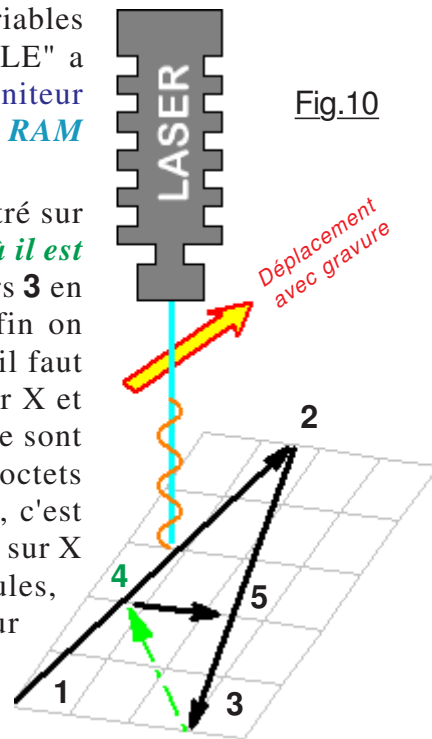


Fig.10

➤ De la place en mémoire dynamique à revendre.

Imposer de placer les données de la police complémentaire en constantes intégrées dans la zone programme a libéré totalement l'espace réservé aux variables dynamiques. Pour ne pas gaspiller cette zone généralement très sollicitée, initialement les textes étaient très laconiques. Puis, ayant trouvé sur Internet un moyen très simple pour les intégrer comme constantes dans la mémoire Flash réservée au programme, ce dernier a rendu possible des bavardages pratiquement sans limites, car dans ce projet le programme d'exploitation en tant que tel reste modeste en taille. Aussi, cette opportunité de liberté dans le texte du dialogue homme / machine a permis d'afficher des "pages" de texte sans pour autant restreindre les options du programme, à tel point que pour "rentabiliser" l'occupation de la zone programme j'ai recherché avec fébrilité des "moyens de consommer des octets" : Une orgie de fonctions annexes pas vraiment indispensable pour l'utilisateur. Par contre, pour le programmeur qui désire modifier le "Sketch" ces petits outils s'avèrent très agréables et font gagner beaucoup de temps. Noter au passage, que cette débauche de textes affichés n'en est pas moins ultra optimisée, ainsi que les diverses procédures du programme. C'est un principe en ce qui me concerne : Optimiser, optimiser et optimiser l'optimisation. Au fil du développement on minimise la place occupée par le code, qui s'avère la bienvenue pour ajouter des options de convivialité.

Dans ce contexte, il a été possible d'intégrer des commandes d'aide à l'utilisateur. En particulier le "HELP" qui liste à l'écran l'intégralité des commandes disponibles comme on va le voir dans le chapitre suivant. De même que la commande "\$W" qui engendre la liste complète des caractères de substitution. *De ce fait, les fiches initialement créées ne sont plus vraiment indispensables.* Celles relatives aux diverses commandes, car les fiches de protocole pour créer et vérifier un texte avant de le soumettre à la machine restent à mon avis indispensables. À vous de voir ...

Notez que le chapitre 5 n'est qu'une découverte des commandes, pour la partie opérationnelle et les protocoles conseillés un chapitre dédié spécifique est disponible plus avant.

5) Quelques commandes du compilateur de texte gco.

Copie d'écran uniquement retravaillée pour minimiser la surface occupée à l'écran, la Fig.11 présente la fenêtre contextuelle du **Moniteur** lorsque l'on active ce dernier ou si l'on déclenche un RESET. On retrouve le champ de saisie en **1** qui se résume à une ligne de texte maximum. Attention, le texte affiché à l'écran du **Moniteur** ne sera compréhensible que si en **8** la vitesse de transmission sur USB correspond à celle définie par l'instruction **Serial.begin(115200)** dans la procédure **void setup()**. Si cette valeur n'est pas supportée par votre ordinateur, ce qui est peu probable du reste, il suffit d'adopter une valeur plus faible en **8** et corriger celle du programme.

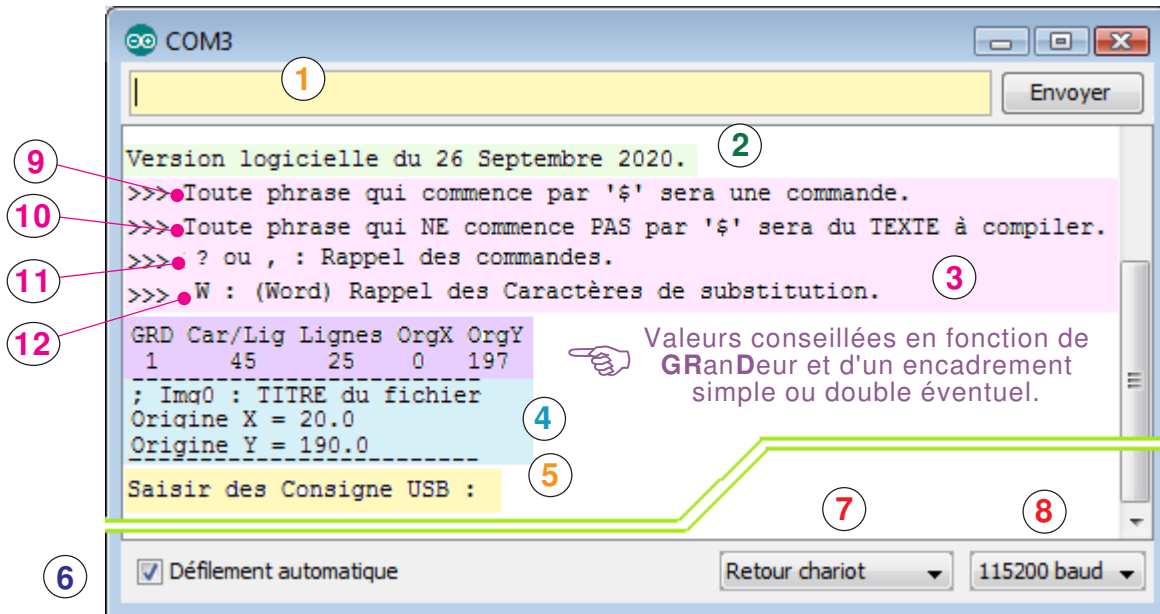


Fig.11

ATTENTION : Le dialogue Homme / machine ne s'établira correctement que si :

- En **6** on coche la case de défilement automatique,
- Et surtout en **7** on sélectionne l'option "**Retour chariot**".

Dans la zone **2** colorée en vert pastel le programme se présente. Ce n'est pas une information vitale, mais comme nous "volons en classe affaire", au diable l'avarice !

Beaucoup plus pertinent en **3** les quatre "bricoles" à savoir et que nous allons tester bien entendu. Chaque fois que l'on valide la ligne de saisie **1** le programme d'exploitation réalisera son travail et affichera un résumé, comme en **4**, des caractéristiques de l'image **gco** qui sera éventuellement générée. Puis, en **5** un court message nous invite à saisir en **1** une **Consigne USB**.

Nous allons immédiatement faire ici la différence entre une **Consigne** qui désignera une **option**, ou une **directive** au programme, et un **TEXTE** à traduire en code gco. Comme précisé dans le rappel en **9** toute ligne en **1** qui commencera par '\$' sera une **Consigne**. Et comme précisé en **10** toute saisie **qui commence par autre chose que '\$'** sera un **TEXTE** à traduire. La toute première commande **11** à expérimenter est "\$h" pour faire afficher le "HELP". Dès que l'on valide cette ligne

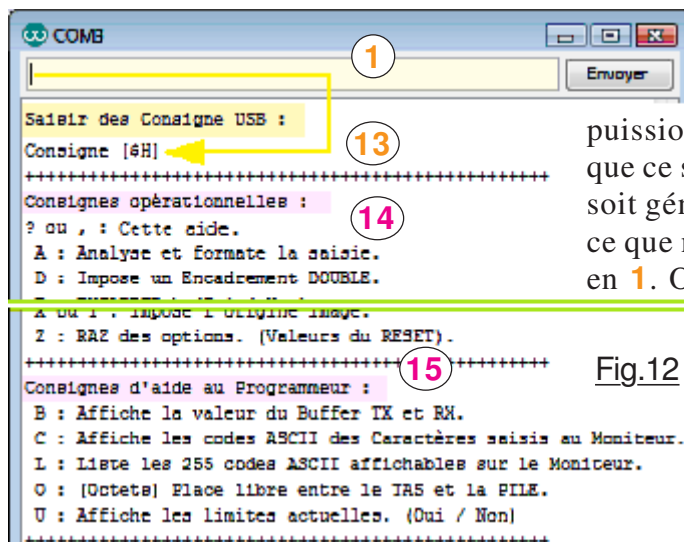


Fig.12

de commande, la ligne de saisie s'efface et le curseur se repositionne à gauche. Puis, ce qui a été frappé en **1** est recopié en **13** pour que nous puissions avoir une trace de ce qui était proposé. Supposons que ce soit un long texte à traduire et qu'un message d'erreur soit généré. Il suffit dans la fenêtre d'affichage de surligner ce que nous avons frappé au clavier, le copier puis le coller en **1**. On corrige ce qui a engendré l'erreur et on valide à nouveau. Suite à "\$h" ou à "\$H" car **minuscules et majuscules sont valides**, le programme liste l'intégralité des consignes possibles avec mise en évidence en **14** et en **15** celles qui relèvent de l'usage du compilateur, et celles dédiées au programmeur éventuel.

> Les caractères de substitution.

Exiger de notre programme la présence d'une police de caractère très riche engendre quelques petites difficultés issues des contraintes d'affichage du Moniteur, et de l'impossibilité en zone de saisie 1 d'obtenir les caractères associés à [ALT], à [Ctrl] etc. Ces caractères particuliers nous sont "interdits" car dans le dialogue établi entre le **Moniteur** et **Compilateur_pour_TEXTE.ino**, ils ne sont pas pris en compte pour diverses raisons techniques. On se limite aux caractères qui fournissent directement un code sur un seul octet. Nous allons passer en revue ces particularités.

Caractères de substitution pour les "non affichables".

Comme n'importe quel système de visualisation, la fenêtre du **Moniteur** dispose d'une police de caractères qui lui est propre. (Et pour ce dernier il n'est pas prévu de pouvoir la changer.) Si vous voulez en avoir la liste complète, il suffit de frapper en 1 la commande "\$I" ou "\$L". La copie d'écran de la Fig.13 présente le listage qui couvre les codes ASCII depuis la valeur 32 jusqu'à 254. (Entre 0 et 32 ce ne sont pas des caractères affichables d'où cette filtration.) Notez au passage que si la commande est saisie en minuscules, les lettres de cette dernière sont converties systématiquement en majuscules, ce qui sur la Fig.13 est mis en évidence en X. Associés aux divers codes de la table, on trouve dans l'encadré violet les symboles tels qu'ils seront affichés. Comme vous le constaterez en observant cette liste, la police de caractère pour notre pyrograveuse est bien différente. Présentée sur la Fig.14 certains caractères semblent un peu de guingois. L'explication figure dans l'encadré situé en haut de la page 9.

Saisir des Consigne USB :		
Consigne [\$L]	←	→ [\$I]
Valeur = 32	Caractère	><
Valeur = 33	Caractère	>!<
Valeur = 34	Caractère	>"<
Valeur = 35	Caractère	>#<
Valeur = 36	Caractère	>\$<
Valeur = 37	Caractère	>%<
Valeur = 38	Caractère	>.<
<hr/>		
Valeur = 251	Caractère	>û<
Valeur = 252	Caractère	>ü<
Valeur = 253	Caractère	>ý<
Valeur = 254	Caractère	>þ<
; Img0 : TITRE du fichier		
Origine X = 20.0		
Origine Y = 187.0		
Grandeur des Caractères = 3		
Saisir des Consigne USB :		

Fig.13

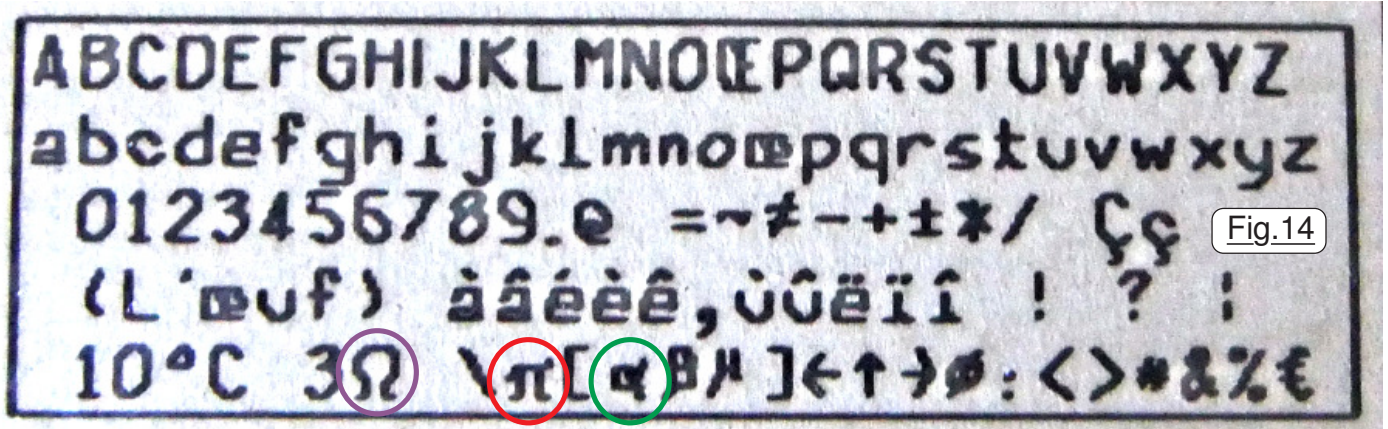


Fig.14

En particulier on désire graver des caractères comme ' Ω ', ' π ', ' α ' qui ne s'affichent pas avec les symboles disponibles dans le **Moniteur**. Ils seront remplacés à l'affichage par "Ohm", "PI" et "Alpha" dans le listage du code objet, et respectivement par 'R', par 'P' et par 'A' dans le listage entouré de '•' présentant le texte qui sera traduit en code graphique. (Voir en Fig.16 un exple de listage entouré.)

Caractères de substitution pour les individus non "présents" au clavier.

Il n'est pas utile de rechercher les touches clavier pour des symboles comme ' π ', ' ϕ ', ' α ', etc. Comme ils ne sont pas disponibles directement, c'est au clavier qu'il faut leur trouver des substituts. Testez l'option 12 de la Fig.11 en proposant la commande "\$w" qui immédiatement sera suivie de l'affichage de la Fig.15 qui propose un listage complet des divers substituts clavier. Sur cette copie d'écran sont mis en évidence en vert pastel les symboles désirés et disponibles pour la pyrograveuse. À droite surlignés en jaune les caractères qu'il faudra saisir pour obtenir les symboles désirés. Par exemple, si vous désirez graver $R = 15\Omega$. Vous devrez saisir $R = 15$ ". Évidemment, vous vous doutez bien que les "ersatz" ne sont pas choisis au hasard. C'est la fiche **Choix des caractères pour mémoriser aisément** qui résume et justifie un peu ces choix. Certains sont motivés par la facilité mnémotechnique pour les retenir. D'autres en fonction de leur position sur le clavier. Il ne faut pas perdre de vue que de nombreuses contraintes restreignent les possibilités laissées à la programmation. En particulier, la lettre majuscule 'C' avec cédille génère deux codes clavier successifs, non acceptable par le logiciel **Compilateur_pour_TEXTE.ino** et impose de ce fait une procédure pour 'Ç' ou

Influence des arrondis en déplacements vectoriels.

Compte tenu des "résidus de calculs" lors des déplacements vectoriels, certains caractères comme "Différent de" ou -> sont codés légèrement dissymétriques pour optimiser le tracé de la petite police de 5mm de haut. Une fois agrandis on observe ces dissymétries, car c'est la plus petite police qui a été privilégiée. Par ailleurs les caractères en police unitaire sont parfois déformés bien que codés symétriques. Cette déformation disparaît à la gravure dès la valeur pour la **Grandeur** en option "\$gN" égale ou dépasse 2.

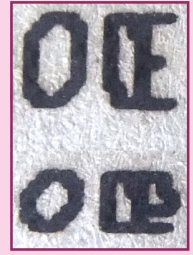


Fig.15

```

Consigne [$w]
+++++
Caractères de substitution :
PI se remplace par "
Ohm se remplace par $
Différent se remplace par -
Alpha se remplace par {
B se remplace par }
± se remplace par ;
ø se remplace par x
« se remplace par [ALT 7]
^ se remplace par [ALT 9]
» se remplace par f
œ se remplace par s
Œ se remplace par ' suivi de 'o'
~ s'obtient avec [ALT Gr ~] mais uniquement
si suivi d'un autre Caractère.
Pour le Caractère Ç ou ç on doit impérativement
saisir 'C' ou 'c' suivi de 'ç'.
Note : Le Caractère '|' sera affiché '|'.
+++++
  
```

'ç' qui sera particulière et surlignée en Fig.15 par du bleu clair. Heureusement pour nous ces différents symboles exigeant des remplacements sont finalement peu utilisés dans du texte "courant". Et puis à tout moment vous pouvez faire appel aux aides utilisateur. Par exemple vous avez proposé un texte que le compilateur vous présente sous la forme de la Fig.16 avec en accusé de réception la délimitation rose dans laquelle figure le texte tel qu'il sera gravé. Observant l'écran il vous revient à l'esprit que le **Œ** est possible mais vous avez oublié le code. Voici la procédure simple à utiliser :

- 1) Surligner le texte et [Ctrl c] suivi en saisie de 'n',
- 2) En saisie "\$w" pour lister les substituts,
- 3) En saisie [Ctrl v] pour récupérer le texte,
- 4) Corriger "OE" par '2' suivi de 'o' et validez.

Le texte sera corrigé et **Œ** affiché normalement.

```

Saisir des Consigne USB :
.....
• bel OEUF de Paquès •
.....
Texte correct ?
  
```

Fig.16

➤ Les commandes de type Options ou de type directives.

Nuance qui mérite d'être clarifiée, une option et une directive génèrent un comportement totalement différent du programme d'exploitation. *Une **OPTION** modifiera le résultat compilé.* Par exemple la commande "\$g3" imposera une police de caractères trois fois plus grande que celle de base "\$g1" qui est la plus petite, c'est celle initialisée par défaut sur un RESET. Du reste, comme l'image qui sera gravée en sera modifiée, la zone 4 de la Fig.11 le précise comme on peut l'observer sur l'encadré bleu foncé de la Fig.13 par exemple. C'est à dire que *toute modification apportée à ce qui sera tracé et qui ne correspond pas "au standard" de base sera ajouté aux informations des caractéristiques en zone 4.* Si maintenant on saisit "\$g1" qui ramène au format de base, l'information "Grandeur des Caractères = 3" n'est plus affichée. Outre l'option de grandeur des caractères, on dispose de deux autres possibilités relatives à un encadrement du texte compilé.

Fig.17

Testez la commande "\$e" par exemple. Immédiatement la zone des caractéristiques image de la Fig.17 ajoute l'information pertinente. C'est avec cette option que se trace automatiquement un encadrement simple tel que celui observable en Fig.14 qui présente la police de caractères disponible. Réitérez "\$e" annule cette option, la commande se comportant comme une bascule de type OUI / NON. Validez encore une fois "\$e" pour rétablir l'encadrement suivi de "\$d". Cette fois, on voit en Fig.18 que l'encadrement sera double octroyant un "effet de style". L'option "\$d" est également de type OUI / NON. L'invalider n'annule pas l'encadrement simple. Par contre "\$d" imposera "\$e" si l'encadrement n'est pas actif. Enfin, annuler l'encadrement simple avec "\$e" supprime également l'option "double" si cette dernière est active. Testez ces quelques comportements assez naturels à mon sens et relativement évidents à l'usage.

```

-----
; Img0 : TITRE du fichier
Origine X = 0.0
Origine Y = 190.0
Encadrement demandé.
-----
  
```

```

-----
; Img0 : TITRE du fichier
Origine X = 0.0
Origine Y = 190.0
Encadrement double demandé.
-----
  
```

Fig.18

► Les DIRECTIVES du logiciel d'exploitation.

Contrairement aux OPTIONS, les directives déclenchent une action immédiate et n'influencent pas les caractéristiques de l'image qui sera pyrogravée. On peut déjà préciser que les aides au programmeur résumées sur la copie d'écran de la Fig.19 sont des directives. Puisqu'elles sont mentionnées ici, évacuons immédiatement leur cas en précisant leurs effets et l'on pourra les oublier. Les deux commandes "vertes" seront très utiles si vous désirez remplacer un symbole actuel par un personnel qui vous sera plus utile. Pour effectuer un tel changement dans le programme les trois fiches nommées *Modifier un caractère dans la table* N/3 précisent la façon de procéder. C'est précisément dans ce cas que "\$C" et "\$L" seront des aides conviviales. Testez ces consignes, vous comprendrez immédiatement leur comportement. La commande "\$U" (*Comme Utilisateur.*) ajoutera les limites images résultant des options actives et du texte soumis au compilateur. Cette information *peut s'avérer intéressante dans certains cas assez particuliers* pour aider l'utilisateur à cadrer son texte, et

```

+++++
Consignes d'aide au Programmeur :
B : Affiche la valeur du Buffer TX et RX.
C : Affiche les codes ASCII des Caractères saisis au Moniteur.
L : Liste les 255 codes ASCII affichables sur le Moniteur.
O : (Octets) Place libre entre le TAS et la PILE.
U : Affiche les limites actuelles. (Oui / Non)
+++++

```

Fig.19

à positionner la cible sur le plateau de la machine. Toutefois, avoir ces données en permanence à l'écran génère une sorte de saturation d'informations contradictoire avec une "simplicité visuelle". Cette directive est donc ajoutée aux consignes, et par défaut la bascule se trouvera en état "NON".

Quand aux deux directives surlignées en ocre pastel, force est de constater qu'elles relèvent d'un petit "délire de programmeur" ! J'ai déjà précisé lors d'autres didacticiels, que je cherchais toujours sur des projets tel que celui-ci, à utiliser au maximum les ressources de l'ATmega328. Comme le "Sketch" *Compilateur_pour_TEXTE.ino* restait inexorablement économe en taille de programme, je cherchais désespérément des moyens pour "gaspiller des octets". Les directives "\$B" et "\$O" résultent de cette déraison. Si vraiment vous êtes très proche du C++ testez ces deux "parasites", dans le cas contraire ... oubliez somptueusement.

Retrouvons en Fig.20 la copie d'écran déjà rencontrée en Fig.12 dans laquelle toutes les directives sont présentes, sauf celles qui ont été expérimentées et citées dans ce qui précède. Passons en revue leurs effets respectifs. À tout moment il sera possible d'enregistrer les caractéristiques actuelles

```

COM3
Saisir des Consigne USB :
Consigne [$S]
+++++
Consignes opérationnelles :
? ou , : Cette aide.
A : Analyse et formate la saisie.
I : Nom de l'IMAGE. (16 Caractères MAX)
M : Comme 'S' Mémoire les options actuelles.
P : Purge la fenêtre du Moniteur.
R : Restitue les paramètres mémorisés.
S : Comme 'M' Sauvegarde les options actuelles.
T : TITRE de l'image. (16) Caractères MAX)
V : Valide les options optimales.
X ou Y : Impose l'origine image.
Z : RAZ des options. (Valeurs de RESET).
+++++

```

Fig.20

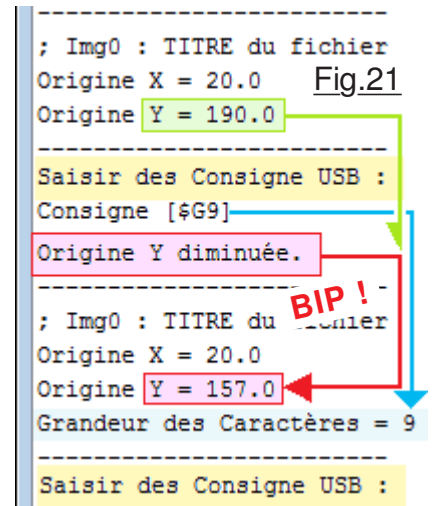
de l'image, c'est à dire tout ce qui est détaillé en zone 4 de la Fig.11 au moyen des deux directives "\$m" ou "\$s" qui sont "interchangeables". La commande réciproque "\$r" sera employée pour reprendre les caractéristiques mémorisées. Si l'on désire en une seule consigne retrouver les valeurs de base du RESET, c'est "\$z" qui convient. Toutes les options sont réinitialisées y compris le TITRE banal initial.

La directive "\$p" n'est qu'une petite bricole qui purge la fenêtre du Moniteur en "effaçant" la zone des comptes-rendus. Comme on s'en doute un peu, la consigne "\$tBlablaba" changera dans la remarque en première ligne du code objet le TITRE. De même que "\$iN" en modifiera le "nom" de l'image. Les deux commandes "\$xN" et "\$yN" servent à imposer la position de l'Origine Image sur le plateau de la machine. Quand on

utilise de grands caractères, positionner cette Origine Image peut s'avérer indigeste, car on aboutit facilement à des débordements vers le haut, le bas, la gauche ou la droite. Ce n'est qu'en validant un TEXTE qu'un message d'alerte sera alors généré, imposant de rectifier les coordonnées et de saisir à nouveau le texte. Aussi, le calcul des l'origines gauche minimale et la position en hauteur minimale pour ne pas déborder à gauche ou à droite sont réévaluées avant chaque affichage des caractéristiques de l'image. Du reste si l'utilisateur change la grandeur des caractères et que l'une des positions

est devenue incorrecte, le logiciel la corrige automatiquement. *Il en informe l'opérateur par un BIP d'alerte et un petit message d'erreur.* Chaque fois que **Compilateur_pour_TEXTE.ino** détectera un incident ou de façon autoritaire parera un problème, il préviendra l'opérateur par un message d'alerte assorti d'un BIP sonore. Expérimentons un peu ces divers comportements :

- Effectuez un RESET puis "\$g9". Le programme se rend compte qu'avec cette grandeur le texte va sortir des limites de la machine par le haut. Il vous alerte par le texte encadré en rouge puis recalcule la plus haute *Origine Image* possible. En Fig.21 il impose alors d'autorité cette nouvelle valeur dans les paramètres.
- Maintenant testez "\$x" pour placer l'image tout à gauche du plateau de la machine. (Une ordonnée sans valeur N sera automatiquement initialisée à zéro.) Puis proposez "\$d". Tentez le texte "A". Mince, le cadre double fait déborder par le haut. On rectifie avec "\$y100" par exemple. On réitère le texte "A" et on valide. Cette fois le cadre double fait déborder par la gauche. Bon, on garde son calme, on pousse le texte à droite avec "\$x50". Et enfin on peut tester "Bonjour." par exemple. GLUPS de GLUPS de GLUPS cette fois c'est vers la droite que ça coince ! Et pourquoi pas vers le bas tant que nous y sommes. Et bien oui, avant de lancer la pyrograveuse par la fenêtre proposez le texte "A\$B\$C\$D". N'oubliez pas *que le caractère '\$' dans un texte fait passer la pyrogravure à la ligne suivante*. On valide et pouf, on descend trop sur Y'Y et on craque, **yenamarre !**

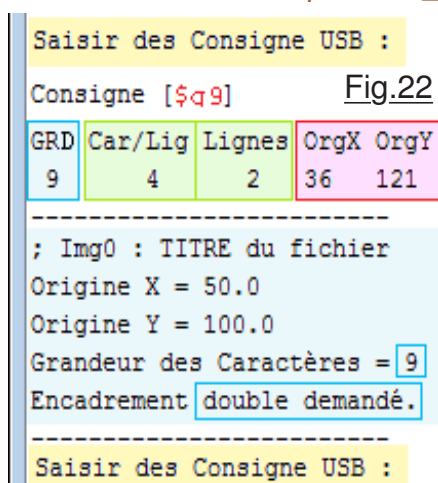


► Le bon conseil au bon moment.

Changer librement de taille de caractères, avec encadrement simple, ou double, engendre une combinatoire de 27 possibilités, chacune modifiant la surface qui sera occupée par le texte sur la cible. De ce fait, il faudra adapter les valeurs de X et de Y pour l'*Origine Image*, limiter le nombre de caractères en largeur et en hauteur en fonction des paramètres sélectionnée dans les options. Et ce n'est pas suffisant, car certains caractères sont plus larges que d'autres comme le 'm' ou le 'w'. Pour la hauteur également la dernière ligne peut aussi venir titiller notre sérénité. Si elle comporte un caractère minuscule avec une queue comme le 'g', le 'p' le 'q' etc ou une cédille, elle sera plus encombrante vers le bas.

- Hé Totoche, ya un problème avec la pyrotruc !
- Ha bon, pourtant elle grave parfaitement cher Dudule.
- C'est que j'ai préparé les textes pour Noël.
- Et alors ?
- Ben j'ai fini par craquer et elle à volé par la fenêtre la pyrotruc !
- Du deuxième étage ? Mais alors elle est fichue espèce de Dudule !
- Ben non, y'avait juste quékun qui passait pile en dessous, elle est intacte. Elle, pas le quékun !

C'est précisément pour éviter de vous prendre la tête que le programme d'exploitation intègre un calculateur qui évaluera à votre place les paramètres pertinents. Pour chaque commande, immédiatement **Compilateur_pour_TEXTE.ino** cogite dans son silicium et propose les valeurs de



la Fig.22 en insistant bien sur le fait qu'il ne fait que conseiller. Du reste dans la zone bleu pastel les données sont inchangées. Dans la colonne GRD il se contente d'indiquer la GRAnDeur de la police utilisée, pour l'information Encadrement ou Double il suffit de regarder juste plus bas dans les caractéristiques de l'image. Comme le programme n'a modifié aucune valeur, il n'y a pas de BIP sonore. Dans l'encadré rose il suggère l'*Origine Image* conseillée pour ne déborder ni en haut, ni à gauche. Et surtout, dans la zone verte on réalise immédiatement qu'à cette taille de caractères, on ne peut placer que quatre symboles en largeur, et deux lignes maximum en hauteur. Donc, si l'on désire absolument "Bonjour." le plus grand possible en surface pyrogravée, il suffit de quelques manipulations faciles à conduire que je propose ici :

- Effectuez un RESET puis "\$g8". On passe à une limite de cinq caractères par ligne. Notre texte en fait huit, il faut encore diminuer un peu la taille.
- Proposez "\$g7". À peine un caractère de plus par ligne, donc on continue à diminuer la taille.
- Tentez un "\$g6" qui montre que c'est encore trop grand.
- Enfin avec "\$g5" on aboutit à exactement huit.
- On impose "\$x20" et "\$y157" puis pour s'assurer que la configuration est optimale, on impose "\$d" suivi du texte "Bonjour." qui sera entouré par un Cadre Double. On valide et sur la Fig.23 le programme présente ce qui sera compilé. Il attend alors une confirmation par 'o' ou par 'n'. Tout autre caractère engendrera un BIP d'alerte. La réponse 'n' annule le texte saisi et le programme attend une nouvelle consigne. Si on valide par 'o' la traduction est immédiate.

Si au préalable nous avons employé les options "\$i3" et "\$tDirectives" on remarquera en A de la Fig.23 que la première ligne de remarque sera personnalisée en B.

À ce stade de la découverte des commandes il ne me paraît pas opportun de commenter le résultat de la compilation, un chapitre spécifique sera consacré à cette analyse. Il me semble plus rationnel de continuer à explorer les directives disponibles. Notez toutefois, que la réponse 'o' engage la traduction et le programme nous en informe par le texte COMPILE suivi entre crochets du TEXTE

```
Saisir des Consigne USB :
.....
• AAç? •
.....
Texte correct ? o
COMPILE [AAç?] -> Cédille non précédée de C ou C.
```

Fig.24

qui sera transformé en codes *gco*. On se doute un peu qu'il va compiler, cette information fleurit la surabondance. Et bien ce n'est pas le cas, c'est *"une assurance contre les accidents"*. Durant la compilation un problème peut être détecté comme le prouve la Fig.24 qui présente alors la réaction du programme. Si le texte est court, autant le saisir à nouveau, mais si la ligne proposée contient plus de

deux cents caractères ... dommage ! Aussi, dans ce cas il suffit de surligner dans la fenêtre des comptes rendus tout ce qui est entre les crochets, de coller ces données dans la fenêtre de saisie. Ensuite, on rectifie le ou les caractères en cause et l'erreur est corrigée. C'est la grande force du **Moniteur** que de pouvoir copier tout ce que l'on veut dans sa grande zone d'affichage.

➤ L'action raisonnée est toujours préférable aux conseils.

Puisque le programme est capable de calculer la position idoine de l'*Origine Image*, pourquoi nous obliger à en saisir les valeurs avec "\$xNN" et "\$yNNN" ? Aussi, dans la mesure où l'on peut faire confiance, autant utiliser directement l'OPTION "\$v". Testons cette possibilité :

```
; Img0 : TITRE du fichier
Origine X = 00.0
Origine Y = 177.0
Grandeur des Caractères = 5
Encadrement double demandé.

Saisir des Consigne USB :
Consigne [$v]
Valide les valeurs optimales :
GRD Car/Lig Lignes OrgX OrgY
5      7      4      20     157

; Img0 : TITRE du fichier
Origine X = 20.0
Origine Y = 157.0
Grandeur des Caractères = 5
Encadrement double demandé.
```

Fig.25

- Effectuez un RESET puis "\$g5", "\$x0" suivi de "\$d".
- Proposez l'OPTION "\$v".

La Fig.25 résume les réactions du programme. Dans la première zone bleu pastel on retrouve la valeur Y'Y de 177.0 qui correspond à l'ordonnée par défaut de l'*Origine Image*. On y constate que l'abscisse sur X'X a bien été remplacée par zéro. Puis, la consigne "\$v" a effectué l'analyse du contexte. Dans le message de retour encadré en vert le programme précise qu'il a bien Validé les valeurs optimales qu'il a remplacé pour l'*Origine Image*. Il nous informe également sur le nombre maximum de caractères que l'on pourra placer dans chaque ligne et le nombre total de lignes envisageables. Comme l'image générée sera modifiée, puisque sa position sur le plateau ne sera plus celles du mode standard, alors un BIP d'alerte sonore est généré pour attirer l'attention de l'opérateur qui utilise le compilateur.

```
; Img3 : Directives
Origine X = 20.0
Origine Y = 157.0
Grandeur des Caractères = 5
Encadrement double demandé.

Saisir des Consigne USB :
.....
• Bonjour. •
.....
Texte correct ? o
COMPILE [Bonjour.]

; Img3 : Directives
; 110 lignes.
G90
; Origine Image.
G0 X10.0 Y127.0
; Origine ligne du haut.
G0 X20.0 Y147.0 S0
;-- Corps du programme --
• Caractère R
```

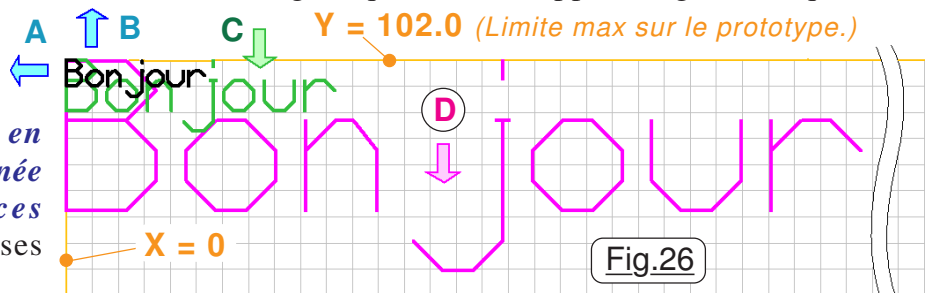
Fig.23

C heminement un peu confus dans les diverses commandes acceptées par le logiciel, bien que nous sommes dans le chapitre qui est sensé traiter des Directives, force est de constater que dans la liste de la Fig.20 sont intercalées quelques OPTIONS. L'exemple typique est celui de "\$v" qui modifie le code *gco* qui sera généré. Remarque analogue pour "\$iN" et "\$t...." qui modifient le contenu de la première ligne compilée. Ce sont donc des OPTIONS et non des Directives. Peu importe ces entorses à la rigueur de la syntaxe, le principal sera de bien comprendre ce que font concrètement les diverses commandes disponibles dont le rappel est obtenu à l'aide de "\$h".

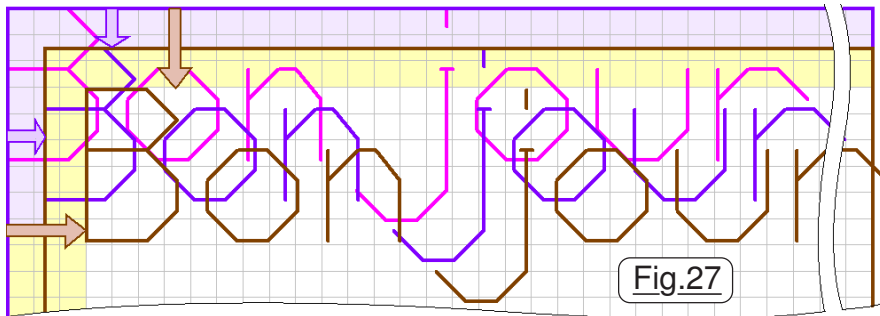
➤ Technique pour optimiser les coordonnées de l'image.

F aire confiance pour optimiser les paramètres avec la consigne "\$v" ne sera avantageux qui si nous savons comment procède le logiciel, et ainsi décider à l'avance si on conserve les valeurs calculées pour la position de l'*Origine Image*, ou s'il sera préférable de la définir manuellement avec "\$xNN" et "\$yNNN". Pour ce qui concerne le nombre de caractères que l'on peut insérer dans une ligne et le nombre maximum de lignes possibles, les valeurs calculées par le programme d'exploitation sont incontournables. Considérons la Fig.26 qui résume l'approche géométrique de la détermination des paramètres.

Le but visé consiste à pouvoir placer un maximum de symboles tant en largeur qu'en "hauteur" en fonction de la grandeur sélectionnée pour la police, et des surfaces couvertes par le LASER les courses étant limitées sur la machine.



Comme nous développons nos phrases de la gauche vers la droite, on comprend immédiatement que pour loger un maximum de symboles il faut commencer tout à gauche en décalant vers A. Remplissant nos pages du haut vers le bas, pour pyrograver un maximum de lignes la première phrase sera placée au plus haut vers B. Passant de la plus petite grandeur tracée en noir sur la Fig.26 il saute au yeux que celle de grandeur 2 en C devra se placer plus bas. Le haut de cette première phrase talonne toujours en Y = 102.0 qui correspond à la position maximale possible du LASER sur la machine. Enfin, pour une police de grandeur 6 par exemple on doit positionner la base encore plus bas en D.



L'encadrement, qu'il soit simple ou double place le début du tracé à gauche du texte. Pour ne pas déborder des limites de la machine, c'est à dire X = 0 dans notre cas, il faut, comme présenté sur la Fig.27 décaler les textes vers la droite. Il est évident que ce décalage sera proportionnel à la

grandeur des caractères. Sur ce dessin, nous reprenons la police de grandeur 6 qui tracée en rose correspond au texte sans encadrement. En violet la commande "\$e" est utilisée. Pour optimiser le nombre de symboles affichables c'est le cadre violet qui maintenant est placé à gauche et tout en haut. Le texte est alors "bousculé" vers la droite et vers le bas d'une valeur correspondant à la distance d'encadrement, décalages représentés par les deux flèches violettes. Si on invoque la consigne "\$d" le texte sera décalé deux fois plus ce que montrent alors les flèches marron. Du coup, il peut arriver qu'avec certaines tailles de symboles, le nombre qu'il devient possible de placer dans le cadre diminue de un. C'est précisément ce que l'on observera dans le tableau figurant sur la fiche nommée *Méthode pour tester les "images texte"* que du reste je vous ai invité à imprimer. Le nombre de ligne maximum possible en hauteur sera également fonction du contexte.

Pour clore ce chapitre, sachez que si la dernière ligne ne comporte pas de caractères avec une queue comme le 'g', le 'j' etc, elle occupera moins de place en hauteur. Au contraire, si on y rencontre des lettres minuscules comme 'p', 'q', 'ç' etc, l'encadrement devra se tracer plus bas. Le logiciel en tient compte pour positionner le tracé. Toutefois, il ne peut savoir à l'avance combien il y aura de caractères en largeur et de lignes de texte. Si vous dépassez les valeurs indiquées dans le cadre vert de la Fig.22, la compilation détectera un débordement assorti d'un message d'alerte ...

➤ Aide au cadrage du texte en fonction des paramètres.

Abondance de bien ne nuit pas déclare le dicton populaire. C'est assez vrai, mais la richesse peut pousser aux abus, à l'exagération. C'est un peu le cas de la dernière consigne que nous allons expérimenter. Il s'agit de la directive "\$a" qui propose une aide au formatage du texte à tracer. Elle ne constitue en rien une révolution et ne doit son existence qu'à la place qui restait pour "tortiller du programme" et à ma manie de vouloir gaver la mémoire flash. L'idée consiste à apporter une aide à l'utilisateur pour qu'il puisse "dégrossir" la mise en page en fonction des circonstances. Quand on sera dans ce mode, toute phrase proposée sera analysée jusqu'à ce que l'on désire en sortir avec la consigne "\$\$". Testons cette Directive sur un exemple pour en évaluer la valeur ajoutée :

- Effectuez un RESET puis "\$g3" suivi de "\$d" pour établir un contexte.
 - Saisir la directive "\$a" qui impose de confirmer avec 'o' car le programme change de comportement.
 - Soumettre le texte "Les chiens aboient et la caravane passe sans se soucier de leur présence.".
- Cette longue phrase est modifiée pour "passer en largeur". Le programme intercale des passages à la ligne. Le texte est converti et devient :

>Les chiens aboient et la caravane passe sans se soucier de leur présence.<

Il est possible d'évaluer les endroits où il sera conseillé d'insérer un passage à la ligne. Toutefois, l'approche présentée ci-avant n'est pas la meilleure, il me semble plus efficace de proposer le texte initial sans les divers espaces. Recommencez :

- Proposer "Leschiensaboientetlacaravanepassesanssesoucierdeleurprésence.".

Le programme retourne :

>Leschiensaboie\$ntetlacaravane\$passesanssesou\$cierdeleurprés\$ence.<

On évalue alors où placer quelques passages à la ligne et l'on propose :

- Saisir "Les chiens\$aboient et\$la caravane\$passe sans\$se soucier\$de leur\$présence.".

Cette fois le texte est acceptable, on peut le proposer au compilateur.

```

Consigne [$A]
Passer en Mode Analyse ? o
Valide les valeurs optimales :
MIN X = 12.0
MAX Y = 190.0
Mode Analyse ACTIF pour GRD = 3 et En
Sortie du mode par '$$'.
Traite >Les chiens$aboient et$la caravane$passesanssesou$cierdeleurprés$ence.<
>Les chiens$aboient et$la caravane$passesanssesou$cierdeleurprés$ence.<
FIN du Mode Analyse.
.....
• Les chiens
• aboient et
• la caravane
• passe sans
• se soucier
• de leur
• présence.
.....
Texte correct ? o
COMPILE [Les chiens$aboient et$la caravane$passesanssesou$cierdeleurprés$ence.<
.....
; Img0 : TITRE du fichier
; 711 lignes.
G90
; Origine Image.
GO X6.0 Y13.0
; Origine ligne du haut.
GO X12.0 Y169.0 SO

```

Fig.28

- Copier cette phrase formatée puis sortir du mode par "\$\$".
- Coller le nouveau texte et validez. Le compilateur affiche le résultat qui nous le verrons plus avant conduit à une gravure correcte et bien encadrée.

NOTE : Si vous proposez une phrase trop longue, le formatage limitera le nombre de lignes au maximum compatible avec le contexte et "tronquera" le texte saisi.

À titre d'exemple revenir dans le mode "\$a" et coller deux fois la phrase pour forcer un "débordement".

Copie d'écran retravaillée, le montage de la Fig.28 résume les manipulations effectuées. Lorsqu'en 1 on confirme le désir de passer en Mode Analyse, en 2 d'autorité les origines sont calculées. Puis en 3 un rappel est affiché pour la procédure de sortie du mode. La zone en bleu clair 4 présente une partie de nos manipulations effectuées durant le Mode Analyse dont on sort en 5.

Puis la phrase réorganisée a été soumise au compilateur qui en 6 présente dans la zone délimitée ce qui sera gravé si on accepte. Notez au passage que rien n'interdit lorsque l'on propose la phrase arrangée, d'y apporter de petites modifications, vu que la fenêtre de saisie du moniteur constitue un éditeur de texte rudimentaire. Par exemple en 7 on a ajouté quelques espaces pour mieux centrer les lignes du bas. En 8 on accepte le résultat dont le listage débute en zone 9. On remarque que ce texte de "rien du tout" exige déjà 711 lignes de code *gco*, c'est à dire que sans le compilateur nous aurions été obligés de calculer plus de 1400 valeurs pour les X et les Y !

6) Protocoles optimisés d'utilisation du compilateur textuel.

Protocole optimisé relève d'un pléonasmе, car par définition un protocole est constitué d'une suite d'actions établies par l'expérience en vue d'aboutir à un maximum d'efficacité dans le domaine concerné. En ce qui nous concerne, le but consiste à préparer un texte en vue de le pyrograver, puis à le compiler pour en faire un fichier **.gco** que pourra traiter la machine. Comme il faut ensuite charger ce fichier sur une carte mémoire SD, l'insérer dans le lecteur de la pyrograveuse puis déclencher le processus, l'idéal consiste à pouvoir vérifier sur l'ordinateur que ce qui sera obtenu corresponde exactement à ce que nous désirons avant d'engager toutes ces manipulations.

➤ Rédiger rapidement un texte.

Indispensable, la fiche **Protocole pour rédiger un texte** a été imprimée et plastifiée. Outre les techniques optimisées pour préparer notre "verbiage", l'autre coté de la fiche nommé **méthode pour tester les "images texte"** sera particulièrement utile pour nous permettre de sélectionner la grandeur du texte en fonction du contexte souhaité, c'est à dire des options d'encadrement.

À titre d'exemple, on suppose que vous vous préparez à recevoir des amis à Noël et que pour la circonstance vous placerez une petite pancarte à la porte d'entrée. Vous prévoyez le texte suivant :

Joyeux Noël à toutes et à tous.
Soyez les bienvenus chez nous.
Laissez dehors vos soucis et vos problèmes.
Confiez-nous vos peines de cœur.

(Comme vous le constaterez rapidement, les caractères de substitution ne sont finalement pas très souvent rencontrés, d'où la dernière phrase pour y avoir recours dans cet exemple.)

Première étape, avec un traitement de texte banal comme le **Bloc-notes** par exemple, rédiger le texte sans finesse. (*Fig.29A.*) On désire que la petite pancarte soit la plus grande possible avec un minimum de grandeur de caractère de 2. Avec "\$g3" on se rend rapidement compte que le nombre

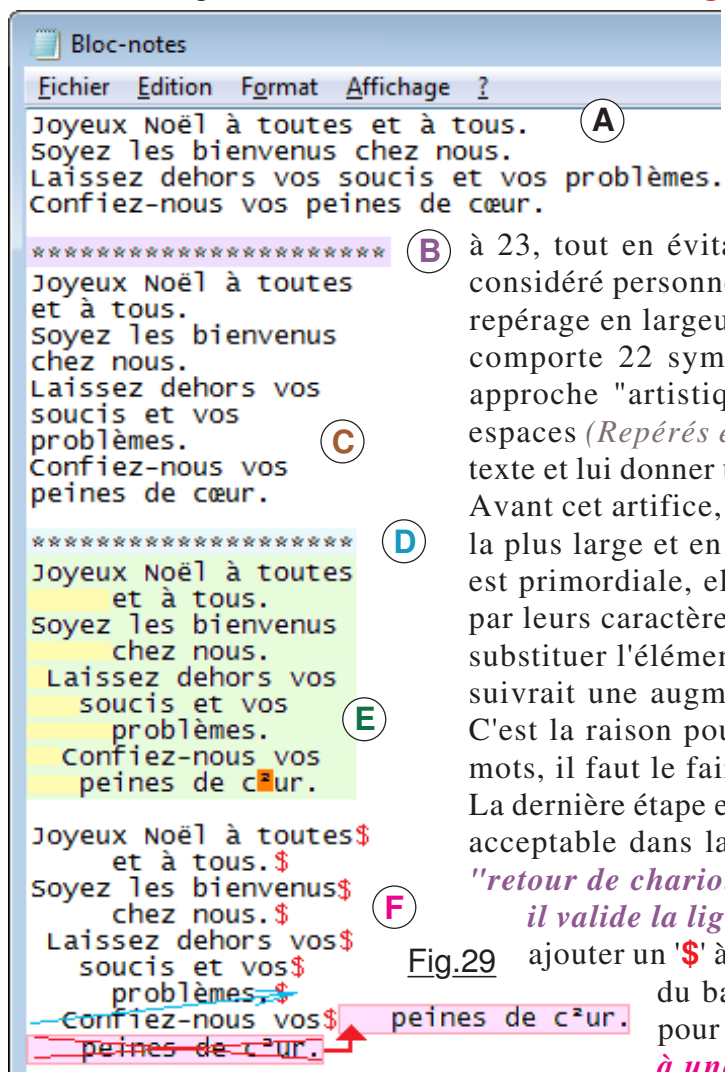


Fig.29

de lignes dépassera la hauteur possible. On décide donc une grandeur double avec "\$g2" qui imposera un maximum de 22 symboles en largeur. On recopie le texte **A** pour constituer en **C** des lignes les plus larges possibles, mais dont le nombre de caractères restera inférieur à 23, tout en évitant de couper des mots car un trait d'union est considéré personnellement comme peu esthétique. Pour faciliter le repérage en largeur on a tracé la ligne d'étoiles **B** qui évidemment comporte 22 symboles. L'étape suivante **E** commence par une approche "artistique", c'est à dire que l'on ajoute à gauche des espaces (*Repérés en jaune sur la copie d'écran.*) pour équilibrer le texte et lui donner un peu l'allure d'un générique de cinématographe. Avant cet artifice, on compte le nombre de caractères dans la ligne la plus large et en **D** on ajuste la ligne d'étoiles. La phase qui suit est primordiale, elle consiste à remplacer les symboles concernés par leurs caractères de substitution. Comme on peut être amenés à substituer l'élément par deux caractères comme pour le **œ**, il s'en suivrait une augmentation "artificielle" de la largeur de la ligne. C'est la raison pour laquelle, si l'on désire centrer les groupes de mots, il faut le faire avant la phase des substitutions.

La dernière étape en **F** sert à transformer le texte en une ligne unique acceptable dans la fenêtre d'édition du **Moniteur** pour laquelle *le "retour de chariot" pour passer à la ligne n'est pas possible car il valide la ligne saisie.* La technique la plus simple consiste à ajouter un '\$' à la fin de chaque ligne située au dessus en partant du bas vers le haut, et de frapper la touche **[Suppr]** pour enlever le "retour de chariot" parasite. *On aboutit à une ligne unique de 164 symboles.* **Page 15**

➤ **Vérifier le résultat compilé avant de créer un fichier.**

Difficile d'imaginer à l'avance ce que donnera un texte en fonction de ses particularités. L'aspect général sera influencé notamment par la largeur des caractères. Contrairement à la police utilisée sur le **Moniteur**, celle du compilateur génère des tracés dont la largeur sera affinée dans le but d'obtenir un résultat visuel le plus esthétique possible. Par exemple en largeur constante, le 'i' majuscule engendrerait un écart trop important. Le 'n' est plus étroit que les autres caractères, alors que 'w' ou 'W' sont plus larges. Ainsi ils sont moins "tassés". Lorsque vous avez regardé la Fig.14 pour la première fois, je doute fort que vous ayez remarqué ces subtilités. On constate globalement un texte dont l'apparence des divers caractères semble homogène. Hors, manifestement la ligne des lettres minuscules est plus large que celle située au dessus, pourtant elles sont identiques au point de vue "grammatical". Aussi, pouvoir vérifier l'impression finale d'une compilation peut amener à modifier légèrement les lignes pour mieux les équilibrer, évitant ainsi de nombreuses manipulations entre l'ordinateur et la pyrograveuse sans compter l'économie réalisée en matériaux à graver.



```

.....
• Joyeux Noël à toutes •
•      et à tous.      •
• Soyez les bienvenus •
•      chez nous.      •
• Laissez dehors vos •
•      soucis et vos •
•      problèmes.     •
• Confiez-nous vos •
•      peines de cœur. •
.....
Texte correct ?

```

Fig.29

La technique va consister à utiliser **Repetier-Host**, peu importe sa version, pour visualiser les déplacements du LASER. Fiche **Méthode pour tester les "images texte"** en main on expérimente :

- Pour conserver sur le bureau le **Bloc-notes** qui contiendra le texte initial, on ouvre une deuxième instance de ce traitement de texte. Puis comme indiqué sur la fiche on sauvegarde son contenu, c'est à dire "rien du tout" sous un nom tel que **TEST.gco** par exemple.
- On prépare le "vérificateur" en ouvrant **Repetier-Host V1.6.2** et on charge une première fois **TEST.gco** pour qu'il enregistre "le chemin" sur le H.D. puis on valide les deux options  et .

- **Attention :** Sur la fiche, il n'est pas précisé un point important. Pour que **Repetier-Host V1.6.2** visualise les déplacements du LASER il faut impérativement cocher l'option **Montrer les mouvements de déplacements**. (Voir Fig.30)

Visualisation

- ☒ **Montrer les Mouvements de Déplacements**
- ☒ **Montrer tout**

Fig.30

- RESET sur le compilateur puis "\$g2" suivi de "\$d" pour imposer nos préférences.
- Commande "\$v" pour forcer un positionnement optimisé.
- Dans le premier **Bloc-notes** on copie avec [Ctrl c] **la longue ligne de 164 symboles**.
- Dans la ligne de saisie du **Moniteur** on colle avec [Ctrl v] cette **longue ligne** et on la valide.

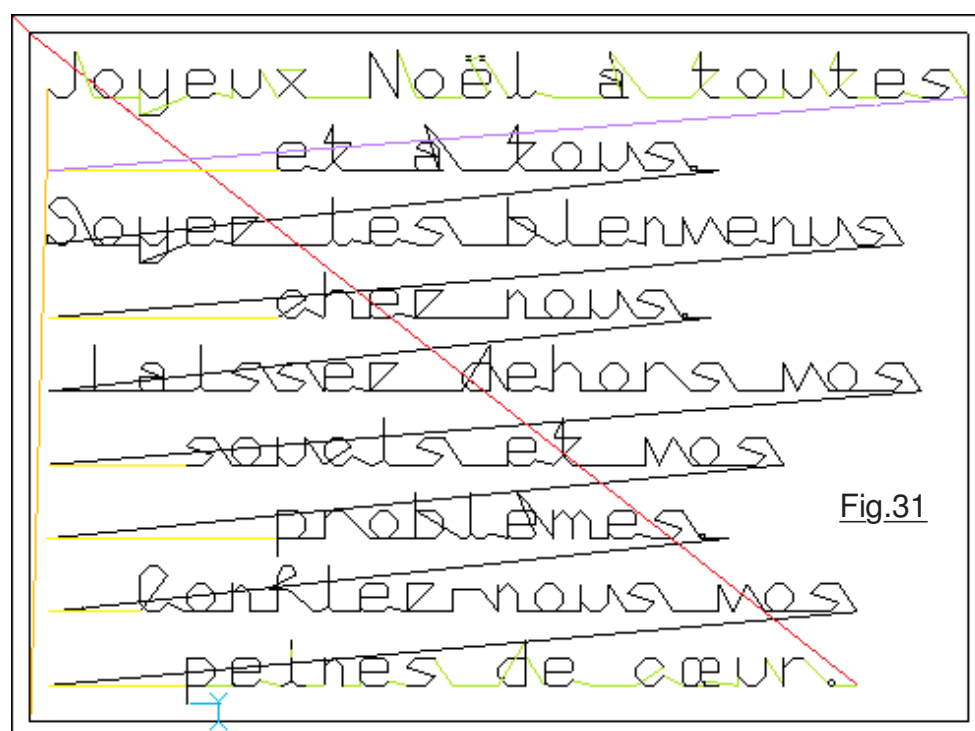


Fig.31

Copie d'écran de ce que montre **Repetier-Host V1.6.2** modifiée pour éviter la grande surface sombre de son arrière plan. Le logiciel présente l'intégralité des déplacements et ne fait pas la différence entre mouvement sans le LASER et gravure. Sur ce dessin certains sauts ont été mis en évidence en couleur. En teinte orange le déplacement initial de l'origine image vers le début de la première ligne. En rouge les mouvements pour tracer les deux cadres. En vert sur la première et la dernière ligne les sauts entre les caractères et pour leur construction. Enfin en jaune les espaces pour centrer les mots dans les lignes et dans l'encadrement.

- Le futur résultat Fig.29 semble correct, donc on valide avec la réponse 'o'.
- Frouuuuu, ça compile rapidement. L'ensemble de ce que liste le programme sera commenté dans un prochain chapitre. *Le code objet.gco est contenu entre les deux lignes "=====*". Surligner ces 1353 lignes de code et les copier avec [Ctrl c].
- Dans le Bloc-notes traitant TEST.gco coller ce code avec [Ctrl v]. Si l'option Barre d'état dans l'onglet Affichage est cochée, en bas à droite on doit avoir Ln 1353, Col 10.
- Avec [Ctrl s] on sauvegarde le contenu.
- Enfin, comme indiqué sur la fiche on active la fenêtre de Repetier-Host V1.6.2 puis on clique sur ▼ en 1, et un peu à gauche sur le nom du fichier en 2 pour voir le résultat qui ressemble à la copie d'écran de la Fig.31 sur laquelle on peut constater que le texte sera effectivement centré dans un cadre double. Globalement les lignes sont bien équilibrées. On peut noter que la dernière ligne (*Repéré en bas en bleu.*) comporte la lettre 'p' avec une queue inférieure. Le cadrage en a tenu compte.

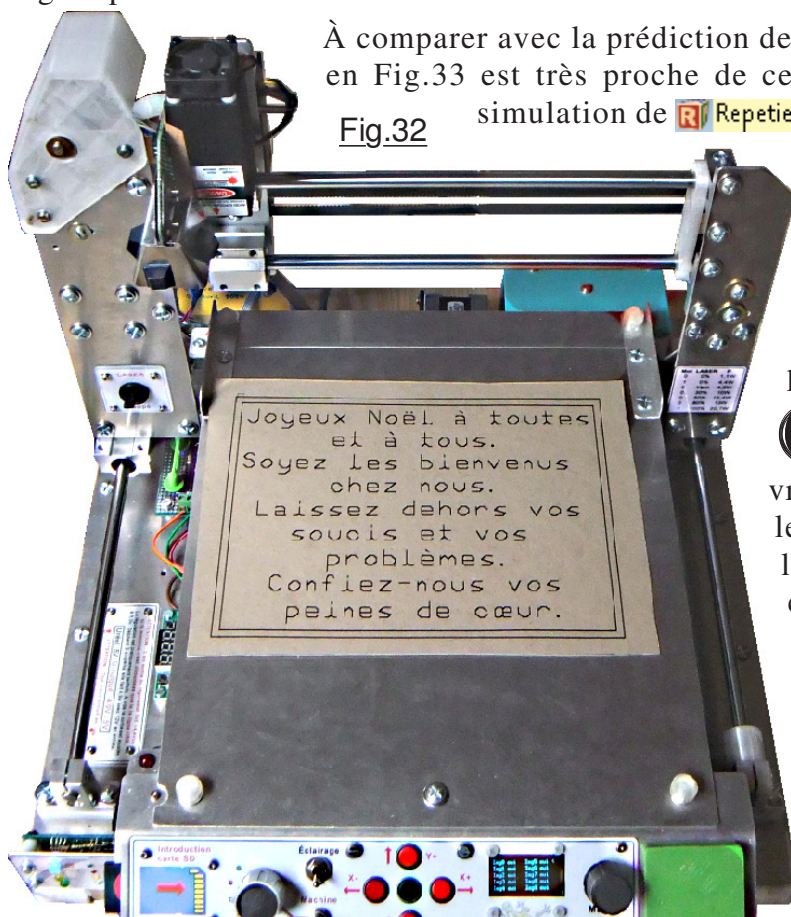
➤ Graver : La finalité de tous ce processus.

Simulation encourageante avec Repetier-Host V1.6.2, les manipulations ne sont pas terminées, car nous allons mener le processus à son terme, c'est à dire pyrograver. Ce n'est qu'à ce stade que sera révélée une pleine réussite, ou que des détails imprévus assombriront notre enthousiasme.

- Sauvegarder TEST.gco sur la carte mémoire SD et le renommer Img5.gco par exemple.
- Insérer la petite mémoire de masse sur le lecteur de la pyrograveuse et sélectionner le bon fichier. Cliquer sur la touche Y-↓ pour vérifier que c'est la bonne image. Le programme de la machine estime à environ dix minutes la durée exigée par le traitement.
- Déclencher le processus avec Y+↑. Comme pour n'importe quel fichier le LASER est déplacé à l'Origine Image, c'est à dire dans l'angle inférieur gauche du cadre pour notre cas. Le LASER est en mode pointage et l'on doit placer la cible sur le plateau de la machine. (Dans la réalité le temps pour graver sera de 19 minutes.)

```
; Img5 : Exemple Noël
Taille 1353 lignes
Origine X = 4.00
Origine Y = 44.00
Gravure : ± 10 min
```

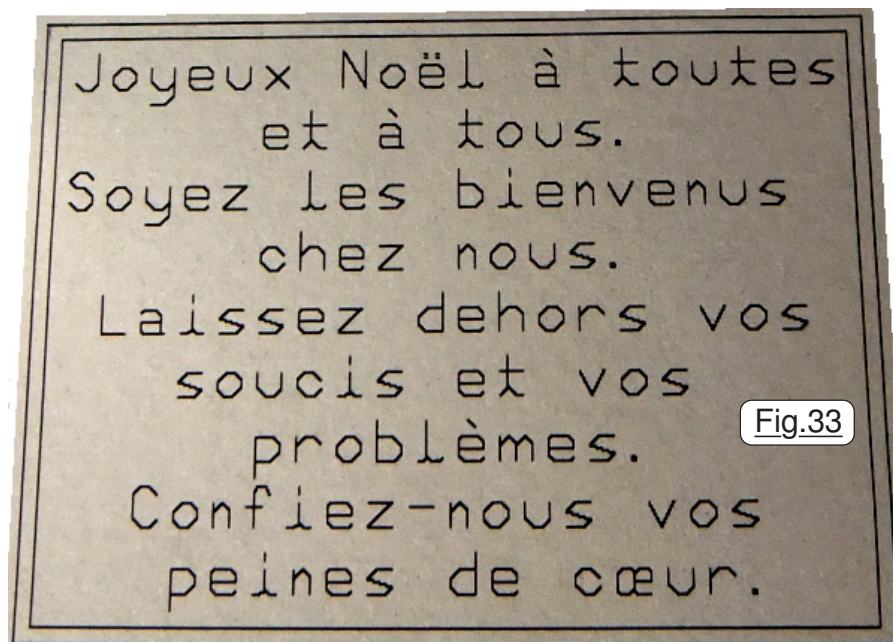
ATTENTION : Bien que la taille des caractères ne fait que deux fois celle de la plus petite police, l'encadrement occupe 212mm de large et 158mm de hauteur. Aussi, comme on peut le voir sur la Fig.32 prévoir une cible de surface suffisante avant de la positionner sur la machine.



À comparer avec la prédiction de la Fig.31 le résultat sur la machine montré en Fig.33 est très proche de ce que l'on pouvait imaginer à l'aide de la simulation de Repetier-Host V1.6.2. Bien que ce logiciel soit conçu à

la base pour piloter des imprimantes 3D et qu'il ne soit absolument pas prévu pour du gravage 2D à l'aide d'un LASER, on observe que pour cette application il soit très fiable, et constitue une aide précieuse pour s'assurer de l'aspect final que présentera le texte pyrogravé.

Certains et certains vont certainement se demander si ce compilateur est vraiment pertinent. Pourquoi ne pas rédiger le texte dans PAINT.exe par exemple, puis le compiler avec LaserGRBL.exe comme explicité dans le didacticiel relatif à la pyrograveuse. C'est tout à fait possible bien entendu. Du reste le plus persuasif consiste à effectuer la manipulation. Vous charger Bienvenue à Noël.jpg dans LaserGRBL.exe et vous le compilez avec les paramètres conseillés dans le didacticiel sur la pyrograveuse. Pensez à imposer une largeur de 21,2cm.



Vous pouvez vérifier en chargeant le résultat [Bienvenu à Noël.gco](#) dans [Repetier-Host V1.6.2](#) que la taille du code objet fait maintenant 7049 lignes au lieu de 114. Il faudra donc 61 fois plus de temps pour le graver, soit environ 19 heures ! Aussi, en ce qui me concerne, pour les quelques euros que l'on investit dans la carte Arduino NANO et les bricoles qui vont autour, il n'y a pas à hésiter. La balance penche largement du côté du compilateur. Seule petite ombre au tableau, le caractère 'n' est un peu étriqué et nuit à mon sens à l'équilibre de l'ensemble, il faut le modifier.

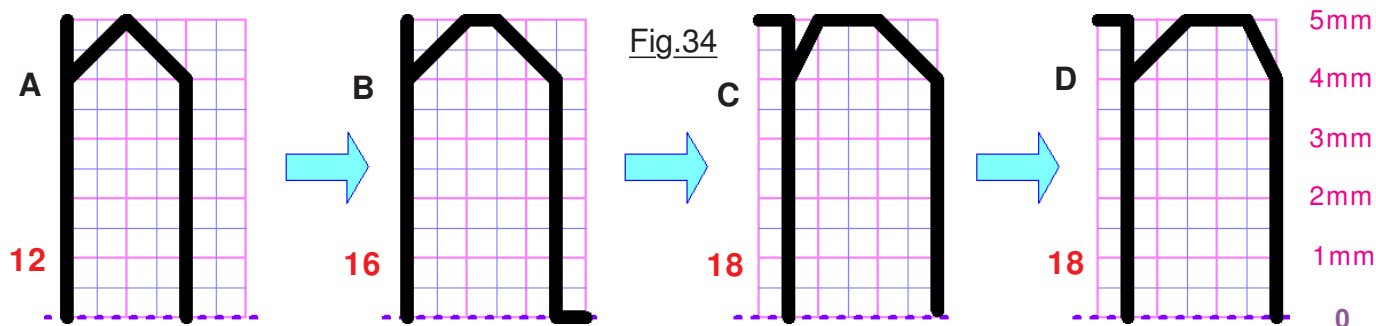
7) Modification de la police de caractères.

Prétexte parfait pour vous montrer comment procéder, vous n'aurez pas à effectuer ces manipulations, car les fichiers définitifs qui accompagnent ce tutoriel ont bénéficié de cette petite amélioration. Une fois encore, j'insiste sur le fait que ce chapitre ne concerne que les programmeurs en C++ qui systématiquement aiment bien personnaliser leurs programmes. Pour les autres, passez directement au chapitre suivant.

➤ **Repenser la forme du symbole.**

Construire une police de caractère comme celle de ce petit programme n'est pas élémentaire du tout, car le programmeur doit gérer de nombreuses contraintes, accepter des compromis. Bref, ce n'est pas un long fleuve tranquille. Par exemple, dans l'état actuel ce n'est pas moins de 1826 déplacements géométriques (*Mouvements et codages d'empreintes.*) que le dessinateur a déterminé, sachant que pour certains symboles l'artiste a envisagé entre trois et cinq variantes avant de se décider à en sélectionner une. Parmi les contraintes qu'il faut prendre en compte on peut citer :

- *La plus petite grandeur envisagée : 5mm de hauteur.*
- *Pour des raisons de proportions 3mm de largeur en standard avec une séparation de 2mm.*
- *La définition de la machine est de 0,5mm donc une matrice de 10 x 6 points sur lesquels débutent ou se terminent les segments rectilignes qui traceront le caractère.*
- *La vectorisation cumule des "décimales" les segments tracés ne sont pas forcément "directs".*
- *Le nombre de déplacements pour un caractère impacte en proportion la taille du code objet.*



Muni d'une page quadrillée, (*Et surtout de patience et de motivation !*) on commence par griffonner une ou plusieurs formes. Puis on fige la candidate sélectionnée dans la table de définition. Enfin on teste le résultat. Précisé dans l'encadré situé en haut de la page 9 on vérifie que la déformation résultant de la vectorisation ne déforme pas trop l'aspect final. Éventuellement on modifie un peu la forme. Bref, ce n'est pas de la programmation en ligne, un nombre significatif de tentatives s'avère incontournable. Ceci dit, pour quelques symboles cette approche progressive reste relativement rapide et "artistiquement intéressante". C'est parti pour le 'n' qui avait

été tracé moins large (*Voir la Fig.34 A.*) que pour le standard probablement pour économiser des octets, *car initialement il était question de loger entièrement la police en EEPROM ...*

Sur la Fig.34 est indiqué en rouge le nombre d'octets consommés pour définir le symbole dans la table des caractères. Plusieurs variantes ont été envisagées, et la seule façon de sélectionner la plus appropriée consiste à les graver toutes à la plus petite grandeur, celle qui déforme le plus, puis à la plus grande si pour diminuer la "distorsion" on a opté pour un tracé non symétrique. (*Voir les fiches nommée **Modifier un caractère dans la table.***)

➤ **Les essais se font avec la table complémentaire.**

Modifier la police de base logée en EEPROM s'avère rapidement indigeste si l'on désire tester diverses variantes car chaque fois on doit corriger `Police_de_caracteres_en_EEPROM.ino`, le téléverser vers la carte NANO, puis à son tour téléverser `Compilateur_pour_TEXTE.ino` pour procéder aux essais. Loger le caractère en cours d'élaboration en mémoire flash sera bien plus rapide, car seul le programme d'exploitation sera à transférer pour examiner le résultat. Le problème, c'est que la police logée en EEPROM est sondée en premier, et si le caractère à tracer s'y trouve, la police complémentaire sera ignorée. Il faut donc enlever l'empreinte actuelle de l'EEPROM :

- Effacer le codage du caractère 'n' dans `Police_de_caracteres_en_EEPROM.ino`, téléverser le programme dans la carte NANO. Activer le **Moniteur** pour vérifier le résultat.
- Les trois variantes imaginées présentent maintenant une largeur standard, il faut "prévenir" le programme d'exploitation `Compilateur_pour_TEXTE.ino` que ce 'n' n'est plus étriqué :
 - * Effacer `case 'n' : {Largeur_caractere = 4; break;}` dans `void Traiter_le_Caractere()`,
 - * Effacer `case 'n' : {Largeur_caractere = 4; break;}` dans `void Calcule_les_limites()`.
- Ajouter la définition du symbole à expérimenter au début de la table complémentaire par exemple, il ne sera pas obligatoire d'en augmenter la taille 850 qui est très largement dimensionnée :

```
|| const uint8_t Complement_de_police[850] PROGMEM = {  
    'n',8,101,100,1,6,0,6,101,104,3,6,5,6,6,4,6,0, // Test variante D.
```


- Tester toutes les variantes sur la pyrograveuse et sélectionner la plus pertinente.

➤ **Remplacer l'empreinte d'un symbole en EEPROM.**

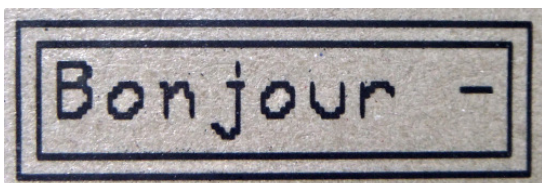
Outre que cet exemple correspond effectivement à une modification tardive du logiciel, c'est l'exemple idéal car il constitue le pire des cas. En effet, l'EEPROM est actuellement saturée. Il n'y reste qu'un seul octet non utilisé. Hors la définition du nouveau 'n' fait passer sa consommation de 12 à 18 octets soit six de plus. Ils ne sont pas disponibles, il faut donc faire de la place. Notons au passage que l'EEPROM ne contenant que les symboles de base, il est peu probable que nous soyons amenés à en modifier beaucoup. C'est précisément dans cette optique que tous les caractères spéciaux ont été logés en mémoire de programme.

- Effacer le codage du caractère '-' dans `Police_de_caracteres_en_EEPROM.ino`, ce qui libère exactement six octets.
- Remplacer l'empreinte du caractère 'n', du coup l'EEPROM est à nouveau saturée.
- Téléversez `Police_de_caracteres_en_EEPROM.ino` et activez le **Moniteur** par sécurité.
- Ajouter la définition du '-' au début de la table complémentaire :

```
|| const uint8_t Complement_de_police[850] PROGMEM = {  
    '-',2,100,105,6,5,
```

- Téléversez `Compilateur_pour_TEXTE.ino` et activez le **Moniteur**.
- Testez la variante avec  **Repetier-Host V1.6.2** comme largement décrit dans les chapitres précédents puis examen final sur la pyrograveuse. C'est parfait ? Alors l'affaire est classée.

Presque deux pages pour décrire la technique à privilégier. Franchement, c'est bien trop, car au final quand on s'y risque on se rend compte que c'est presque élémentaire. La Fig.35 atteste du fruit de ces travaux. Même si c'est subtil, la nouvelle forme montrée ici avec la plus petite grandeur est




visuellement plus belle. Et comme sur les 850 octets réservés on n'en consomme réellement que 802, toutes les initiatives seront possibles. Alors chères programmeuses et chers programmeurs ... à vos armes !

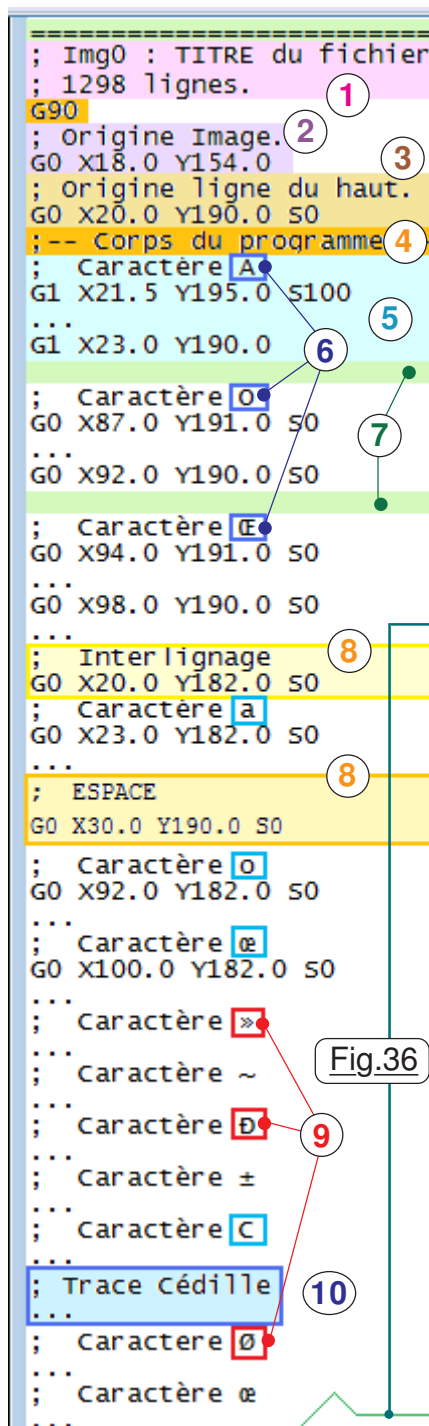
Fig.35

8) Analyse du code objet résultant d'une compilation.

Promesse effectuée dans le paragraphe *Le bon conseil au bon moment* en page 12 du tutoriel, nous allons passer en détails la façon dont code le compilateur. Un grand nombre de remarques sont incluses dans le listage du programme objet pour guider l'utilisateur. Ces informations seront particulièrement pertinentes pour apporter de petites modifications, ou surtout pour diagnostiquer la source d'une éventuelle incongruité toujours envisageable. Ayant déterminé la zone en cause, il sera alors relativement aisé de modifier le texte initial pour parer la difficulté rencontrés. Pour passer en revue tous les détails intéressants, et tous les cas possibles, on va compiler l'alphabet complet de la Fig.14 avec un encadrement double :

- RESET sur le compilateur puis "\$d" dans le **Moniteur** pour imposer l'option.
- Ouvrir le  **Bloc-notes** et charger **ALPHABET.txt** qui contient le texte à pyrograver.
- Frapper [Ctrl a] suivi de [Ctrl c] pour copier la longue phrase.
- Dans la fenêtre de saisie du **Moniteur** [Ctrl v] pour coller le texte et valider pour le compiler.
- Enfin accepter la transposition avec 'o'.

Le listage purgé des lignes de code pour ne pas occuper plusieurs pages de tutoriel est représenté sur



```
=====
; Img0 : TITRE du fichier
; 1298 lignes.
G90
; Origine Image.
G0 X18.0 Y154.0
; Origine ligne du haut.
G0 X20.0 Y190.0 S0
; -- Corps du programme
; Caractère A
G1 X21.5 Y195.0 S100
...
G1 X23.0 Y190.0
; Caractère O
G0 X87.0 Y191.0 S0
...
G0 X92.0 Y190.0 S0
; Caractère E
G0 X94.0 Y191.0 S0
...
G0 X98.0 Y190.0 S0
...
; Inter ligneage
G0 X20.0 Y182.0 S0
; Caractère a
G0 X23.0 Y182.0 S0
...
; ESPACE
G0 X30.0 Y190.0 S0
; Caractère o
G0 X92.0 Y182.0 S0
...
; Caractère æ
G0 X100.0 Y182.0 S0
...
; Caractère »
; Caractère ~
; Caractère D
; Caractère ±
; Caractère C
; Trace Cédille
; Caractère Ø
; Caractère æ
=====
```

Fig.36

la Fig.36 avec ses deux lignes de séparation repérées en vert. Ce sont toutes les lignes *situées entre ces deux repères* qu'il faut copier pour les soumettre à la pyrograveuse. On trouve en **1** les deux premières lignes de commentaire avec juste en dessous le premier **G90**. Comme nous n'avons pas utilisé "\$t" ni "\$iN" ce sont les informations par défaut qui sont en remarque. Puis en **2** le compilateur a calculé la position exacte de l'*Origine Image* où sera placé le LASER pour que l'opérateur puisse positionner correctement la cible sur le plateau de la machine. En **3** le programme a calculé la position du premier déplacement pour tracer les caractères et en **4** il précise où commencent les tracés. Le code du tout premier déplacement avec gravure qui suit la ligne **4** se termine par **S100** indispensable pour préciser que sur les lignes de type **G1** le LASER sera à 100%. *C'est par le menu des options sur la machine que l'utilisateur imposera un dosage de puissance en fonction du matériau pyrogravé.* La zone **5** de chaque symbole commence par

une remarque **6** qui en précise la nature et se termine par une ligne

vide **7** pour bien les séparer visuellement sur le listage. Les symboles non affichables seront précisés par du texte comme en **8** ou par des équivalents visuels comme en **9**. Dans le cas d'un C avec cédille le tracé de cette dernière est repéré comme en **10**. Enfin les quelques lignes relatives à un éventuel encadrement sont repérées

en **11** avec une deuxième zone potentielle **12** si l'option double est validée. Le compilateur se charge on s'en doute de générer comme en **13** les instructions de fin d'une image avec retour du LASER en configuration "machine dégagée".

Notez que les deux lignes de séparation vertes en début et fin de listage NE DOIVENT PAS être copiées dans le fichier image.

10) Adapter le logiciel aux caractéristiques de votre machine.

Programmer n'est pas forcément votre tasse de thé. C'est la raison pour laquelle *je tente toujours de fournir un ensemble complet clef en main*. Toutefois, nous savons qu'entre votre machine et la mienne, il pourra fort bien y avoir de petites différences de performances, c'est à dire que les amplitudes de déplacement le long de X'X et le long de Y'Y ne seront pas exactement identiques. Hors *le logiciel du compilateur a été optimisé pour les déplacements correspondant à ceux qui sont effectifs sur le prototype*. J'imagine que si vous réalisez ce dispositif, c'est qu'au préalable vous avez construit votre pyrograveuse et du coup vous en connaissez parfaitement les limites. Au tout début du programme sont prévues des directives pour pouvoir adapter au mieux et optimiser le logiciel du compilateur en fonction des performances de votre machine. Pour repérer facilement les lignes de programme prévues pour apporter ce genre d'ajustements, elles sont terminées par des remarque de type `//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@` faciles à trouver dans le listage. Cinq lignes de code vous concernent directement :

```
#define Course_MAX_sur_X 228.0 // Valeur à prendre dans le programme P20.  
#define Course_MAX_sur_Y 202.0 // Valeur à prendre dans le programme P20.  
#define Origine_pour_X 20 // @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  
#define Origine_pour_Y 190 // @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  
#define Version_logiciel "Version logicielle du 26 Septembre 2020."
```

On se doute que *les valeurs en rose sont celles relatives à votre machine*, ces valeurs sont déjà présentes dans son programme `P20_Programme_exploitation.ino`.

`Origine_pour_X` et `Origine_pour_Y` sont les coordonnées de l'*Origine Image* par défaut sur RESET. J'ai choisi une position pas trop éloignée de celle de la configuration dégagée pour gagner du temps machine si l'échantillon à graver est de petite taille et qu'il soit possible de le positionner dans cette zone sans qu'il ne dépasse des limites de travail du LASER.

Enfin, la version du logiciel pourra être changée si vous apportez des modifications personnelles. Ce n'est pas une donnée fondamentale, mais parfois elle peut s'avérer utile si on se retrouve avec plusieurs essais et que l'on ne sait plus quel est le dernier qui a été téléversé.

11) La der des der !

Parallèle un peu osé, j'oserais dire que la programmation et la guerre ont un point commun. La modification ultime est la der des der. Malheureusement, force est de constater que pour les déchirements en ce bas monde, un nouveau vient toujours remplacer le dernier. Le programme est terminé, il tourne comme une planète sur son orbite. On peut enfin passer à autre chose. Et puis Pafffff, c'est l'idée géniale qui surgit. Le "dommage que je n'y ai pas pensé avant". En puis mince ... on craque et on remet ça. C'est ce qui c'est passé lorsque je rédigeais ce didacticiel. Au moment d'aborder le chapitre sur les regrets, j'ai basculé du côté obscur de la force et décidé d'ajouter une nouvelle fonction. Pas vraiment une révolution, mais un petit plus bien commode.

➤ *Check-list pour le préambule.*

Trop souvent, au moment où l'on teste un fichier que l'on va soumettre à la pyrograveuse, on se rend compte qu'avec toutes les manipulations effectuée, à un moment où à un autre on a loupé un détail. Il n'y a plus qu'à tout recommencer. Aussi, *un module d'aide qui pas à pas nous imposerait d'initialiser tous les paramètres du futur texte serait bien appréciable*, surtout quand on n'a plus utilisé le compilateur depuis trois mois. L'idée est simple : Les commandes du début ont été un peu chamboulées pour dégager une lettre de commande possible. La commande retenue est "\$H" pour **HELP**. La Fig.39 donne une idée du déroulement "linéaire" des opérations. Comme on enchaîne ensuite plusieurs actions, pour ne pas se fourvoyer le programme exige en **1** une confirmation. Si on valide, alors point par point on va devoir "remplir les cases". Avant d'obtenir ce comportement du programme, il faut ajouter, on s'en doute, le code nécessaire au listage source. Hors la débauche de consommation d'octets pour remplir la mémoire de l'ATmega328 ne laisse plus assez de place. Avec cette manie de vouloir remplir totalement la zone flash, quand on veut modifier le programme, il faut faire de la place. Fastoche ... on enlève l'une des deux fleurs dans *les commandes surprise dont il sera question au prochain chapitre* et basta, de la place on en retrouve à revendre.

Hips, je l'aimais bien cette fleur surprise, c'est bien dommage ...

Lorsque l'on valide cette fonction en **1**, l'enchaînement des séquences est inexorable, sauf si l'on écourte la procédure en déclenchant un RESET. En **2** le programme nous demande le numéro de l'image. Tout autre symbole qu'un chiffre compris entre 0 et 9 provoquera un BIP d'alerte et attendra un nouveau caractère. Le titre que l'on désire donner au fichier *.gco* est saisi puis analysé. S'il est trop long il sera tronqué à 16 caractères puis affiché en **3**. Comme il peut de ce fait ne plus correspondre à ce que l'on désire, le programme va boucler en **3** jusqu'à ce que l'on valide en **4** par **oui**. La saisie enchaîne alors sur les deux options d'encadrement en **5** et en **6** attendant comme réponse un 'o' ou un 'n'. Enfin on termine en **7** par la valeur de la grandeur souhaitée pour les caractères. Tant que la réponse ne sera pas comprise entre 1 et 9 il y aura génération d'un BIP d'alerte et attente en saisie d'un nouveau caractère.

Entièrement renseigné pour les diverses options du texte à pyrograver, en **8** le programme nous informe que d'autorité il valide les coordonnées optimales pour la position de l'*Origine Image* et en **9** liste les diverses données pertinentes. Sans qu'il ne le précise, il sauvegarde ces données utilisateur. Si on valide par erreur un texte nous savons que les données de base sont réinitialisées. De ce fait on perd celles qui étaient présentes. Il suffit d'envoyer la commande "\$r" pour les retrouver. Action :

- RESET sur le compilateur puis "\$h" suivi de 'o'.
- Pour l'exercice proposez les valeurs de la Fig.39 comme paramètres souhaités.
- Quand en **10** on poursuit les saisies proposer "\$z" pour réinitialiser les valeurs par défaut.
- Enfin, la commande "\$r" restitue les valeurs sauvegardées par la fonction "\$r".

➤ Les mystères de la programmation sont impénétrables !

Certainement que vous avez remarqué ces informations assez incongrues et étranges coloriées en gris et repérées **11** sur la copie d'écran. Ces valeurs entre parenthèses comme (111) ou (110) suivent l'affichage du caractère qui a été frappé lorsque l'on doit répondre par 'o' ou par 'n'. Si vous utilisez d'autres caractères, ces nombres changent et vous allez vous rendre compte qu'il s'agit en réalité des codes ASCII des caractères précisés en décimal. *Autant dire que pour l'aspect opérationnel on peut se passer royalement des ces informations "parasites" qui surchargent inutilement l'affichage.* Et bien ... on va les conserver sans si Kékun trouve la solution !

L'origine de ce problème réside dans la procédure `void Attendre_o_ou_n()` qui du reste intègre des commentaires pour préciser l'étonnant problème.

Impossible de faire fonctionner correctement le test `if (!(Caractere == 'o' || Caractere == 'n'))` si l'instruction n'est pas précédée de la ligne de code `Serial.print(byte(Caractere))` alors que fondamentalement cette dernière ne modifie aucune donnée. Dans le test j'ai tenté de remplacer 'o' par `byte('O')`, `char(111)`, toutes les variantes qui me sont venues à l'esprit, rien à faire. Le test ne se comportait pas correctement. Du reste c'est par hasard en faisant afficher les codes ASCII pour visualiser des valeurs nulle, vides etc lors de la recherche de l'origine de ce problème que j'ai trouvé cette solution. Ce n'est pas idéal, ça manque d'élégance ... mais ça fonctionne. On persistera à conserver ce "sparadrap" sauf si une personne trouve une réponse plus logique ...

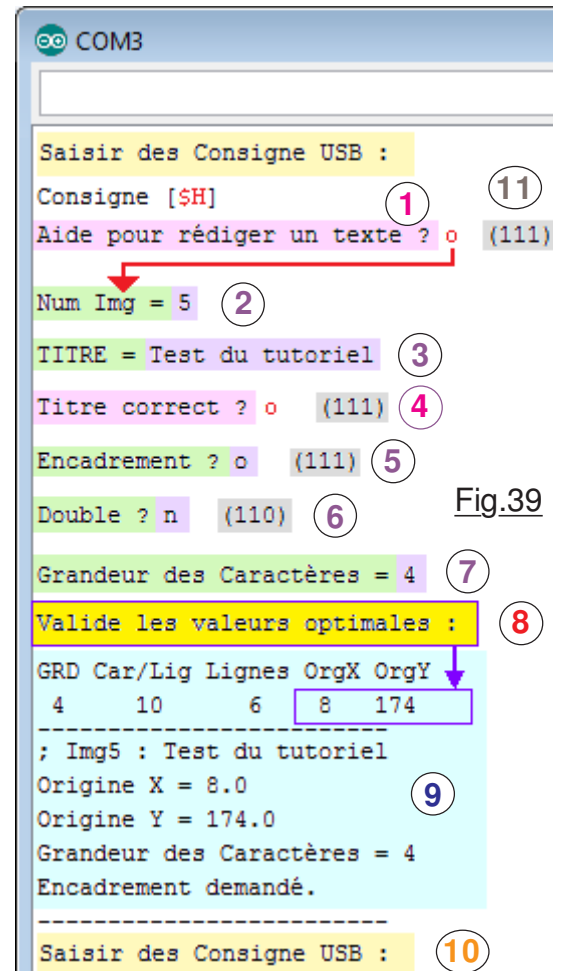


Fig.39



Ben Mòa mòa, je trouve absolument eskandaleux de faire appel au 111 des urgences uniquement parce-que l'on a cliqué sur OUI pour le helpe du compiletruc !

➤ La richesse engendre l'abus et le gaspillage !

O ptimisé à outrance, lorsque le programme à été achevé, et que je ne trouvais plus de nouvelle fonctions pour faciliter l'exploitation, une foule d'aides sous forme de textes affichés à l'écran a été ajoutée dans le listage. Restait encore une place considérable non occupée en mémoire électronique. Aussi, n'ayant plus d'idée pour "consommer intelligemment" de la place, j'ai franchi le coté obscur de la force et ajouté les commandes "\$1" à "\$7" qui ne servent qu'à tracer des petits dessins naïfs. Et tant qu'il y avait de l'espace vital, j'ajoutais un nouveau dessin. Ce ne sont que des petits plaisirs sans plus, la cerise sur le gâteau, le luxe, une orgie délirante !

Grosse bêtise issue d'une frénésie de consommation d'octets, vouloir absolument saturer la mémoire de programme de l'ATmega328 a fait perdre la raison. Idée un peu brutale et non réfléchie, tête baissée j'ai programmé les petits dessins directement par les coordonnées des points à joindre pour construire les petites surprises. Hors chaque point à préciser impose deux **floats** soit 64 octets. Avec cette stratégie seuls deux dessins étaient possibles, et j'avais vraiment envie d'en avoir deux ou trois de plus. C'est à ce moment là que l'exagération de **floats** à révélé un manque sérieux de réflexion, car pour le codage des caractères il était évident que les valeurs des déplacements pouvaient se coder sur des **bytes**. Au prix d'une refonte complète des séquences de code concernées, maintenant on dispose de sept petits dessins "libres". Libre veut dire ici que *leur codage est structuré de façon à ce que vous puissiez facilement les remplacer par les vôtres.*

Le menu de la Fig.20 a un peu évolué et devient celui de la Fig.40 pour prendre en compte ce complément "artistique" logé dans le programme. Dans la zone des diverses **Consignes opérationnelles** a été ajoutée et délimitée la section rose clair **Z** qui affiche l'inventaire de ces petits dessins. Sur la version personnelle j'ai également "\$0" qui trace mon avatar en tout petit, logo qui ne vous concernera certainement pas raison pour laquelle dans le logiciel mis en ligne l'appel à **void Minuscule_Avatar()** est passé en remarque. Le valider dépasse les 30720 octets de disponibles. Sur le logiciel personnel je me suis contenté de passer en remarques les commandes "\$b" et "\$o" et d'enlever les deux lignes d'affichages sur la Fig.40 dans la zone des **Consignes d'aide au Programmeur** pour que ce minuscule logo soit possible.

```
Consigne [ $? ]
+++++
Consignes opérationnelles :
? ou , : Cette aide.
A : Analyse et formate la saisie. Sortie du mode par '$$'.
D : Impose un Encadrement DOUBLE.
E : ENCADRER > (Oui / Non)
G : Impose la GRANDEUR des Caractères.
H : Aide pour rédiger un texte. (HELP)
I : Nom de l'IMAGE.
M : Comme 'S' Mémoire les options actuelles.
P : Purge la fenêtre du Moniteur.
R : Restitue les paramètres mémorisés.
S : Comme 'M' Sauvergarde les options.
T : TITRE de l'image. (16) Caractères MAX)
V : Valide les options optimales.
W : (Word) Rappel des Caractères de substitution.
X ou Y : Origine image.
Z : RAZ des options. (Valeurs du RESET).
+++++
1 : AVATAR
2 & 3 : Fleurs
4 : OISEAU
5 : MINOU
6 & 7 : Notes
+++++
Consignes d'aide au Programmeur :
C : Codes ASCII des Caractères saisis au Moniteur.
L : Liste des 255 codes ASCII affichés sur le Moniteur.
```

Fig.40

➤ Les petits et les grands.

Toute une famille est disponible pour chaque dessin. Dans le codage des tables de coordonnées les valeurs des déplacements relatifs sont doublées par rapport à ce que l'on désire pyrograver. Ainsi, en divisant par deux au moment de piloter le traçage, on dispose des demi-millimètres tout en codant ces mouvements avec des **bytes**. (On divise ainsi par 32 la place occupée en mémoire par chaque petit dessin.) Comme on disposait d'espace programme à profusion, il aurait été dommage de ne pas en profiter pour que le coefficient **Grandeur** soit pris en compte. Du coup chaque dessin est potentiellement disponible en neuf tailles différentes ... à condition toutefois qu'en augmenter les dimensions ne déborde pas de la surface opérationnelle du LASER.

➤ Les mensurations possibles.

Chaque dessin élémentaire possède des dimensions en largeur et en hauteur qui lui sont propres. Les plus petits ne débordent pas des limites machine y compris en **Grandeur** = 9. Ce ne sera vrai que si l'**Origine Image** est correctement placée. Par exemple testez l'expérience suivante : "\$g9", suivi de "\$7" qui va tracer la note de musique en taille maximale. Examinons sur la Fig.41 le déroulement des événements. En 1 la commande "\$g9" impose un facteur **Grandeur** de 9, c'est à dire le maximum possible. Sur RESET l'**Origine Image** sur Y'Y est de 190. Comme à cette taille les caractères déborderaient vers le haut, d'autorité en 2 le programme descend la valeur à 157.0 comme visible en 3. Puisque le programme a pris une initiative, il le signale par les affichages, et pour attirer l'attention de l'opérateur il génère un BIP sonore. À partir d'ici, en 4 il y aura rappel de cette valeur puis en 5 le programme attend une nouvelle commande. En 6 on frappe la commande "\$7" qui demande de compiler le petit dessin ainsi référencé. Immédiatement le logiciel calcule la largeur et la hauteur de cette image qui seront égales à celle de base multipliée ici par 9 la valeur de **Grandeur**. Les dimensions "hors tout" sont alors affichées en 7. (En taille unitaire la petite image fait 6mm x 10mm d'où les valeurs calculées en 7 sur la copie d'écran.) Pour aider au maximum l'opérateur, dans toute la zone rose 8 le programme calcule les positions extrêmes de l'**Origine Image** pour tous les coefficients d'agrandissements possibles. Si la plus grande taille compatible avec les performances de la machine est inférieure à neuf, seules les lignes valides seront affichées. Par exemple sur la Fig.44 la tulipe la plus grande possible est obtenue avec l'amplification de cinq. Seules les cinq premières lignes sont affichées.

Toujours dans le but d'assister au maximum le travail de l'usager, le programme se rend compte qu'avec une ordonnée de 157.0 on va sortir des limites de la machine, en 9 un message d'alerte est déclenché. L'opérateur sait qu'il importe de diminuer sur Y'Y la valeur de la

```

=====
; Img3 : Noire.
; 35 lignes.
G90
; Origine Image.
G0 X20.0 Y50.0 S0
M3 S0
;-- Corps du programme --
G0 X23.0 Y52.0
G1 X22.5 Y51.0
G0 X22.0 Y50.5
G1 X21.0 Y50.5
;--- FIN du programme ---
M5
G0 X0 Y0
=====
GRD Car/Lig Lignes OrgX OrgY
1 45 25 0 197
; Img3 : Bonjour.
Origine X = 20.0
Origine Y = 50.0
=====
Saisir des Consigne USB >

```

Fig.42

```

Consigne [$G9]
Origine Y diminuée.
GRD Car/Lig Lignes OrgX OrgY
9 4 3 0 157
-----
; Img0 : TITRE du fichier
Origine X = 20.0
Origine Y = 157.0
Grandeur des Caractères = 9
Saisir des Consigne USB >

Consigne [$7] (L = 54.0 mm H = 90.0 mm)
Grandeur = 1 Xmax = 222.0 Ymax = 192.0
Grandeur = 2 Xmax = 216.0 Ymax = 182.0
Grandeur = 3 Xmax = 210.0 Ymax = 172.0
Grandeur = 4 Xmax = 204.0 Ymax = 162.0
Grandeur = 5 Xmax = 198.0 Ymax = 152.0
Grandeur = 6 Xmax = 192.0 Ymax = 142.0
Grandeur = 7 Xmax = 186.0 Ymax = 132.0
Grandeur = 8 Xmax = 180.0 Ymax = 122.0
Grandeur = 9 Xmax = 174.0 Ymax = 112.0
-> Débordement par le haut.
GRD Car/Lig Lignes OrgX OrgY
9 4 3 0 157
; Img0 : TITRE du fichier
Origine X = 20.0
Origine Y = 157.0
Grandeur des Caractères = 9
Saisir des Consigne USB >

```

Fig.41

position de l'**Origine Image** avec un maximum à ne pas dépasser de valeur 112.0 pour la **Grandeur** adoptée de 9. Rien n'interdit naturellement de choisir une valeur plus faible comprise entre zéro et le seuil critique de 112.0. Puis en 10 le programme résume les conditions actuelles et en 11 attend une nouvelle commande. **On remarquera au passage que c'est l'opérateur qui reste maître de la position de l'objet cible sur le plateau de la machine, le programme ne fait que fournir des informations pertinentes.**

Noter au passage que le titre de l'image est intégré dans la description des divers points qui la composent. On reste toutefois maître du numéro de l'image qui sera indiqué dans le fichier graphique. Il sera donc avantageux de commencer par la commande "\$iN". Le titre de l'image est contenu dans sa description et sera utilisé dans le fichier.gco. Toutefois, le titre actuel n'est pas perdu. Faisons une expérience élémentaire. RESET sur le microcontrôleur suivi de "\$p" pour purger l'écran. Enchaînons les

commandes "\$i3" accompagnée de "\$tBonjour." sans sauvegarder. Puis on impose "\$y50" pour ne pas risquer de débordement vers le haut. Enfin proposons la commande "\$7" qui compile le fichier. On peut vérifier que le titre de l'image en 1 de la Fig.42 est "Noire" alors que le titre courant affiché en 2 n'a pas été modifié et est resté "Bonjour." ainsi que le numéro de l'image toujours égal à 3.

➤ Principe du calcul de cadrage optimal des images.

Toutes les origines image sont situées en bas et à gauche du dessin. Aussi, les débordements ne peuvent se produire que vers la droite ou vers le haut, c'est la raison pour laquelle seules les limites X_{max} et Y_{max} sont calculées et affichées sur la Fig.41 en zone 8. Examinons sur la Fig.43 la façon dont sont effectués les divers traitements lorsque l'on saisit les commandes encadrées dans la Fig.44 sur laquelle dans la zone violette se trouve la séquence à copier pour réaliser le fichier image. En 1 et 2 sont visibles le titre et le nombre de lignes précisés dans la table de description du dessin.

La grille est composée de carreaux espacés de 10mm, le dessin respecte les proportions. Sur cette vue nous avons tout en bas à gauche l'origine machine en rouge, c'est à dire les *coordonnées nulles sur les deux axes*. Dans ces explications on prend pour exemple le dessin n°3, soit celui de la tulipe. La plus petite *Grandeur* conduit à une image de 17mm de largeur et 39mm de hauteur.

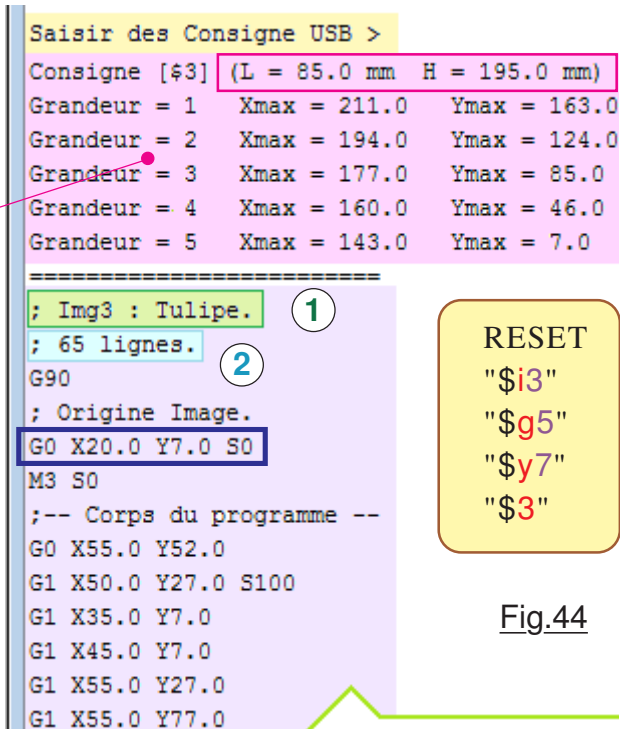
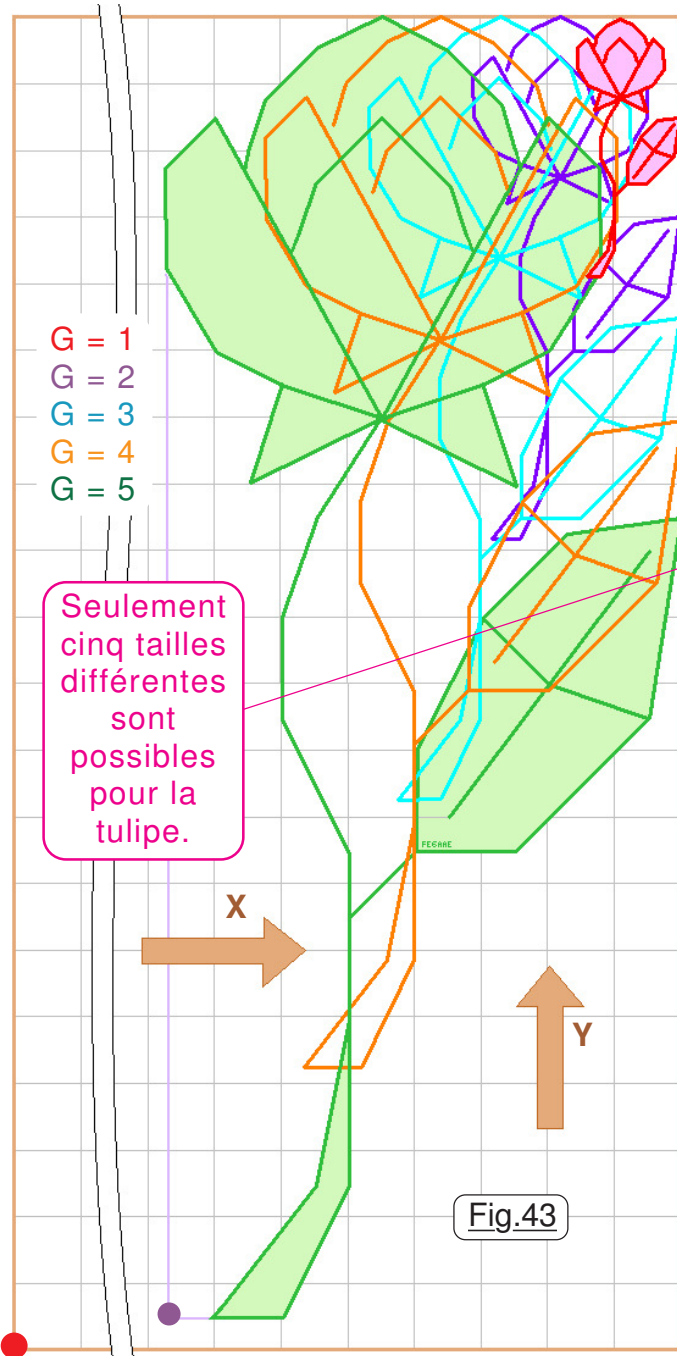


Fig.44

Pour la grandeur maximale possible de 5 la largeur passe à $5 \times 17 = 85\text{mm}$ et la hauteur devient $5 \times 39 = 195\text{mm}$, valeurs affichées dans l'encadré rose. Le programme pousse au maximum à droite flèche **X** "contre" la zone limite du LASER. Il en déduit la valeur maximale de l'*Origine Image* $X_{max} = 228 - 85 = 143\text{mm}$.

Le dessin est ensuite localisé au maximum vers le haut symbolisé par la flèche **Y**. De façon analogue nous avons $Y_{max} = 202 - 195 = 7\text{mm}$. D'où l'*Origine Image* extrême représentée sur la Fig.43 par le disque violet. Sur ce dessin ont été surchargées les quatre autres grandeurs possibles. Tout en haut à droite on observe en rouge et rose la plus petite dimension correspondant au dessin de base. Dans l'encadré bleu de la Fig.44 on trouve les valeurs imposées par l'opérateur.

➤ Les modifications de dernière minute.

Rédigeant le didacticiel ce qui implique des heures à n'en plus finir, les multiples manipulations peuvent faire émerger un "bug" qui était passé incognito durant le développement. C'est assez fréquent dans des programmes tel que celui-ci comportant un très grand nombre d'instructions. La combinatoire des problèmes potentiels et des cas particuliers non envisagés augmente de façon exponentielle. Aussi, lors des exercices que l'on propose dans le tutoriel on peut découvrir un "couf". Dans ce cas, la rédaction du didacticiel est suspendue et le programme est remis en chantier. Par exemple, si vous observez avec une attention particulière la Fig.14, vous allez découvrir que l'encadrement est un peu trop espacé à droite. J'ai remarqué cette verrue tardivement, et expérience à l'appui, traiter l'encadrement et l'option double s'est avéré particulièrement délicat, car un grand nombre de cas particuliers sont à prendre en compte. Bref, certaines modifications ont été apportées au logiciel, du coup il est tout à fait possible que dans certains chapitres la narration ne soit pas totalement conforme à l'actuel logiciel d'exploitation. *(Sans compter les vermines qui n'ont pas encore été découvertes !)* Par exemple l'invite **Saisir une consigne USB :** est devenue **Saisir une consigne USB >** un tantinet plus logique. *Des subtilités sont potentiellement non corrigées sur certaines copies d'écran, il ne faudra pas s'étonner parfois d'observer de petites différences.*

12) Les "loupés".

Fidèle à une tradition que je respecte depuis que je propose des didacticiels sur Internet, le dernier chapitre avant de nous séparer est consacré à un bilan. Ainsi, la porte reste ouverte pour celles et ceux qui trouveraient de la satisfaction à reprendre les études pour "faire mieux". Aucun projet, qu'il soit industriel ou de loisir ne saurait aboutir à la perfection absolue. Entre les désirs initiaux, ce qui était envisagé et ce qui résulte de compromis inévitables, s'insinuera forcément des divergences. Bien que ce petit projet n'échappe pas à ce principe non démontré mais qui frise l'absolu, force est de constater que je n'ai vraiment que peu de "Si c'était à refaire" à formuler. Il m'est possible en cherchant bien d'arriver à noter un "regret" : Trompé par les principes fondamentaux de la géométrie de base, lors des prises d'origine les déplacements se font en vectoriel. La ligne droite n'est finalement pas le plus court chemin pour aller d'un point à un autre. Utiliser deux déplacements cartésiens au lieu de la diagonale serait plus avantageux. En effet, le chemin à couvrir par le LASER est identique puisque numérisé par des escaliers. Dans ce cas, faire deux déplacements au lieu de "beaucoup" serait plus rapide et diminuerait les vibrations sur la machine.

➤ Bilan, conclusion post ambule.

Remarquablement bavard, ce programme sort de l'ordinaire et il ne faudrait pas considérer qu'il constitue un exemple à suivre. En effet, l'ennemi numéro un de la programmation réside dans la taille du programme. Si l'application impose un grand nombre de traitements avec beaucoup de calculs transcendants, des kyrielles de procédures, arrive le moment tant redouté où la compilation "ne passe plus". On doit alors tout remettre en cause, optimiser les optimisations pour grignoter quelques octets. Dans un tel contexte, *il importera de minimiser les textes affichés qui sont gros consommateurs d'emplacements mémoire.* Dans ce petit projet, par discipline le développement a constamment économisé le code objet, chaque instruction a été peaufinée si bien qu'à la fin, une place considérable restait disponible au point que pour "gaver" le microcontrôleur j'ai cherché désespérément à ajouter une foule d'affichages pour améliorer la convivialité d'exploitation. Gardez à l'esprit que c'est un luxe rare qui ne se présentera pas souvent sur des projets "meumeu".

Je souhaite intensément que certaines et certains oseront s'engager dans la réalisation d'un clone, et je ne doute pas de leur réussite. Surtout, je vous souhaite à toutes et à tous de trouver dans ces lignes le plaisir de la découverte. Si d'aventure vous engagez vos heures de liberté dans une telle réalisation et que vous rencontriiez une difficulté, vous pouvez me contacter sur : michel.droui@laposte.net et dans les limites de mon temps de libre, c'est avec grand plaisir que je tenterai de vous dépanner.

Je vous souhaite à toutes et à tous une très agréable lecture.

Chaleureusement : Nulentout.