Οι βιβλιοθήκες asciiart και ncurses

Ανδρέου Γιώργος
(gbandreo@tem.uoc.gr)

20 Ιανουαρίου 2004

Έκδοση 1.0

Περίληψη

Με το κείμενο αυτό παρουσιάζω τις δυνατότητες δύο βιβλιοθηκών, asciiart και ncurses, για τη γλώσσα προγραμματισμού C, σε περιβάλλον Linux[1]. Οι βιβλιοθήκες αυτές προσφέρουν στον προγραμματιστή τη δυνατότητα να κατασκευάσει με εύκολο τρόπο ένα πιο χρηστικό και φιλικό περιβάλλον για τον χρήστη των εφαρμογών για την κονσόλα του Linux.

Η βιβλιοθήκη asciiart προορίζεται για απλές εφαρμογές και γι' αυτό το λόγο μπορεί να πραγματοποιήσει κάποιες στοιχειώδεις λειτουργίες, όπως είναι η χρήση χρωμάτων και η άμεση γνώση των χαρακτήρων, που πληκτρολογεί ο χρήστης, από το πρόγραμμα (unbuffered mode).

Η βιβλιοθήκη ncurses, όμως, δίνει στον προγραμματιστή πολλές άλλες επιπλέον δυνατότητες, με δεκάδες συναρτήσεις που περιέχει αυτή μαζί με τις αδελφές της, βιβλιοθήκες panel, form και menu[2]. Έτσι εκτός από τις ευκολίες για το χειρισμό των ιδιοτήτων (attributes) του τερματικού μας (χρώμα, έντονα γράμματα, κ.τ.λ.) δίνει επίσης τη δυνατότητα κατασκευής όμορφου και εύχρηστου περιβάλλοντος εργασίας για τον χρήστη (UI/User Interface) με παράθυρα και μενού, ενώ επιπλέον κάτω από προϋποθέσεις επιτρέπεται και η χρήση του ποντικιού.

[1] Linux είναι κατοχυρωμένο εμπορικό σήμα του Linus Torvalds.

[2] Θα παρουσιαστούν σε μελλοντική βελτιωμένη έκδοση του κειμένου αυτού.

Περιεχόμενα

Αφιερώνω το πόνημα τούτο στη μνήμη των αγαπημένων μου Δασκάλων, Αδελφού Πολύκαρπου και Δημήτρη Λάππα. Υπήρξαν άξιοι συνεχιστές του παιδαγωγικού έργου του Ιωάννη - Βαπτιστή Δελασάλ και υπηρέτησαν την εκπαιδευτική τους αποστολή με αξιοζήλευτη αρετή και σεβασμό στο μαθητή.

1 Εισαγωγή

1.1 Πρόλογος

Ένα μέρος αυτού του κειμένου παρουσιάστηκε σε μια δίωρη διάλεξη τον Μάιο του 2003 κατά τη διάρκεια του μαθήματος Αρχεία και Βάσεις Δεδομένων, στους φοιτητές του Τμήματος Εφαρμοσμένων Μαθηματικών του Πανεπιστημίου Κρήτης. Από τότε είχα δεσμευτεί να γράψω ένα μικρό οδηγό για αυτές τις δύο βιβλιοθήκες ώστε να έχουν ένα σημείο αναφοράς οι συνάδελφοι φοιτητές.

Ωστόσο, και ενώ ήταν ήδη έτοιμο το μεγαλύτερο μέρος του, στη πορεία άλλαξα άποψη για το σκοπό που θα έπρεπε να ικανοποιεί η συγγραφή του. Καθώς το αντικείμενο αυτού του κειμένου έχει γενικότερο ενδιαφέρον στην κοινότητα του Linux, αποφάσισα να δώσω μια άλλη μορφή στο κείμενο, απευθυνόμενος πλέον σε μια μεγαλύτερη πληθυσμιακή ομάδα, στους Έλληνες Λινουξάδες προγραμματιστές. ;-)

Γράφοντας αυτό τον οδηγό, προσπάθησα να είμαι αρκετά αναλυτικός στα δύσκολα σημεία ώστε ακόμα και ένας αρχάριος προγραμματιστής της γλώσσας C να μπορέσει να χρησιμοποιήσει τις βιβλιοθήκες με άνεση. Άλλωστε το ίδιο το API της βιβλιοθήκης ncurses είναι φιλικό προς τον προγραμματιστή και αμφιβάλλω εάν κάτι από όσα περιγράφονται εδώ θα σας προβληματίσει. Φυσικά εάν παρατηρήσετε ασάφειες ή λάθη θα χαρώ πολύ να ακούσω τα σχόλια σας· απευθυνθείτε στο `gbandreo@tem.uoc.gr`.

1.2 Τεχνικές πληροφορίες για το κείμενο

Το κείμενο αυτό αποτελεί πνευματική ιδιοκτησία του Γιώργου Ανδρέου και διανέμεται δωρεάν. Στοιχειοθετήθηκε με το Ω έκδοση 1.23.2.2 σε λειτουργικό σύστημα Linux. Χρησιμοποιήθηκαν διάφορα πακέτα του LaTeX και έγινε ορθογραφικός έλεγχος με το aspell. Είναι η πρώτη έκδοση που δημοσιεύεται (20 Ιάν 2004). Από την επόμενη έκδοση θα υπάρχει διαθέσιμο το κείμενο στην αγγλική και τη γαλλική γλώσσα.

1.3 Τι μένει να γίνει

Πολλά πράγματα! :-) Συγκεκριμένα, υπάρχει μια κίτρινη καρφίτσα στο φελλοπίνακα που συγκρατεί ένα μουντζουρωμένο χαρτάκι με τις φρασούλες:

α) Δημιουργία ευρετηρίου.

β) Επεκτάσεις της βιβλιοθήκης asciiart και παραδείγματα.

γ) Πώς γίνεται compile η βιβλιοθήκη ncurses και πώς εφαρμόζουμε τα patches.

δ) Περισσότερα παραδείγματα για τη βιβλιοθήκη ncurses. Παραδείγματα για τη χρήση του ποντικιού και των συναρτήσεων της 3.12.

ε) 256 color Xterm & TERM, `man curs_termattrs, curs_terminfo, curs_termcap`.

στ) Η δομή SCREEN και οι σχετικές ρουτίνες *newterm(), set_term()*.

ζ) Wide characters, Unicode και η υποστήριξη της ncurses.

η) Οι βιβλιοθήκες panel, form & menu και παραδείγματα.

θ) Οι αγγλικές και γαλλικές μεταφράσεις!

1.4 Τι με έκανε να αναζητήσω αυτές τις βιβλιοθήκες

Από τότε που ξεκίνησα να προγραμματίζω σε γλώσσα C, συχνά ερχόμουν αντιμέτωπος με προβλήματα όπως: «πώς να καθαρίσω την οθόνη» ή «πώς να εισάγω ένα κωδικό χωρίς να φαίνονται τα γράμματα στην οθόνη» ή «τι καλά που θα ήταν να έχω έγχρωμα μηνύματα» και άλλα... Για καθένα από τα επιμέρους ζητήματα έβρισκα μια λύση «στο πόδι» και με λίγες εντολές έκανα τη δουλειά μου. Όμως για μεγάλα προγράμματα ο κώδικας μεγάλωνε υπερβολικά από δυσνόητες εντολές και που τελικά αντί να λύνουν ένα πρόβλημα, το περιέπλεκαν. Η ανάγκη για ένα συγκεντρωτικό τρόπο αντιμετώπισης των παραπάνω απαιτήσεων, υπό τη μορφή μιας βιβλιοθήκης, ήταν επιτακτική.

Έτσι άρχισα να ψάχνω για διαθέσιμες βιβλιοθήκες στο Διαδίκτυο που να μου επιτρέπουν να κατασκευάζω ένα περιβάλλον εργασίας για τις εφαρμογές, φιλικό προς το χρήστη, και με ευκολία στο προγραμματισμό. Πολύ γρήγορα γνώρισα τη βιβλιοθήκη ncurses. Έχει παράθυρα, μενού, χρώματα, μεγάλο πλήθος συναρτήσεων για είσοδο/έξοδο χαρακτήρων με χρήση buffer ή και χωρίς. Ο «παράδεισος» του προγραμματιστή σε συστήματα Linux!

Ωστόσο, μαζί με μεγάλες εφαρμογές, προγραμμάτιζα και μικρότερες, και είναι άχαρο να χρησιμοποιεί κανείς μια ογκώδη βιβλιοθήκη για να τυπώνει απλώς με πράσινο ή κόκκινο μηνύματα επιτυχίας ή αποτυχίας στη κλήση συναρτήσεων! Έτσι η περιπέτεια της αναζήτησης με έφερε στις ιστοσελίδες ενός καθηγητή πληροφορικής στη Γαλλία. Εκεί βρήκα μια μικρή βιβλιοθήκη, την οποία έχει ονομάσει asciiart, η οποία επιτρέπει τη γνώση χαρακτήρων κατά άμεσο τρόπο, δηλαδή σαν να μην υπήρχε ενδιάμεσος χώρος αποθήκευσης (buffer), και την εμφάνιση έγχρωμου κειμένου στην οθόνη!

Όμως, πριν δούμε αναλυτικά αυτές τις δύο βιβλιοθήκες, θέλω να δώσω μερικές εισαγωγικές πληροφορίες, να παρουσιάσω ορισμένα προβλήματα που συναντούμε συχνά, και να προτείνω κάποιες λύσεις για αυτά...

1.5 Κρατήστε τις οθόνες καθαρές!

Όπως γνωρίζουμε, το πρότυπο ANSI/ISO C δεν απαιτεί την ύπαρξη οθόνης, για αυτό και δεν έχουμε συναρτήσεις για μορφοποίηση της οθόνης στη κονσόλα μας που να λειτουργούν με τον ίδιο τρόπο στα διαφορετικά λειτουργικά συστήματα. Αντίθετα, αυτό αφήνεται στην αρμοδιότητα του λειτουργικού συστήματος και στην προγραμματιστική τακτική (implementation-defined behavior) [3]. Έτσι για παράδειγμα για να καθαρίσουμε την οθόνη σε ένα σύστημα UNIX μπορούμε να χρησιμοποιήσουμε την εντολή

```
system("clear");
```

ενώ στο MS-DOS[4] η εντολή αλλάζει σε

```
system("cls");
```

Αυτή η λύση χρησιμοποιεί κλήση μιας εντολής του λειτουργικού συστήματος και κατά τη γνώμη μου δεν είναι κομψή.

Τι κάνει όμως η εντολή `clear` στην κονσόλα μας; Γιατί «δουλεύει»; Η `clear` πραγματοποιεί τον καθαρισμό της οθόνης και την επαναφορά του δρομέα στη πάνω αριστερή γωνία (πρώτη γραμμή και πρώτη στήλη). Φυσικά, όταν είμαστε σε ένα τερματικό, ο δρομέας θα περιμένει λίγους χαρακτήρες δεξιότερα, αφού έχει παραχωρήσει λίγες θέσεις στη προτροπή του συστήματος (system/shell prompt). Αυτό λοιπόν που κάνει, κρυφά από εμάς, είναι να στέλνει μια *ακολουθία χαρακτήρων διαφυγής* (escape sequence) στο τερματικό μας. Με ένα κόλπο μπορούμε να δούμε ποια είναι αυτή για το τερματικό που δουλεύουμε· π.χ. σε ένα Xterm εάν δώσουμε την εντολή

(george@earth) ~$ **tput clear > clear-seq.txt**

[3] Στο [2], παράγραφος 5.1.2.3, λέει επί λέξη: «What constitues an interactive device is implementation-defined.».

[4] «Microsoft Disk Operating System» της Microsoft Corporation βασισμένο στο QDOS, του Tim Paterson της Seattle Computer Products.

τότε το αρχείο `clear-seq.txt` περιέχει τους χαρακτήρες `^[[H ^[[2J`. Όπου βλέπουμε το ζευγάρι `^[` είναι η οπτική παράσταση του [Ctrl-V + Esc][5], ενώ όλοι οι υπόλοιποι χαρακτήρες είναι εκτυπώσιμοι (`[H, [2J`). Εδώ ίσως έχει μια αξία να αναφέρουμε ότι η εντολή `tput` είναι πολύ χρήσιμη ιδιαίτερα για όσους ενασχολούνται με το προγραμματισμό του κελύφους, π.χ. Bash ή tcsh σε Linux (βλέπε `man {tput,bash,tcsh}` καθώς και το [4]).

Σε ένα σύστημα UNIX όλες οι ακολουθίες χαρακτήρων διαφυγής για τα διάφορα τερματικά είναι καταχωρημένες σε μια βάση δεδομένων στο αρχείο `terminfo` (βλέπε `man terminfo`). Αυτή τη βάση δεδομένων διαβάζουν οι διάφορες εφαρμογές που εκτελούμε και με τις ακολουθίες των χαρακτήρων διαφυγής που βρίσκουν εκεί πραγματοποιούν τις ζητούμενες λειτουργίες: Μετακίνηση του δρομέα σε συγκεκριμένες συντεταγμένες στην οθόνη, καθαρισμός τμήματος της οθόνης, κύλιση της οθόνης, αλλαγή χρωμάτων, αντιστροφή χρωμάτων κ.τ.λ.

Αυτή τη βάση δεδομένων συμβουλεύεται και η βιβλιοθήκη ncurses που για την περίπτωση του καθαρισμού της οθόνης αρκεί η κλήση της συνάρτησης *clear()* για να σβηστεί ότι υπάρχει στην οθόνη. Πληροφοριακά να αναφέρω ότι κάτι αντίστοιχο ισχύει και στο MS-DOS με τη βιβλιοθήκη conio και τη συνάρτηση *clrscr()*.

1.6 *Ξεβάφει* στο καθάρισμα;

Όσον αφορά τη δυνατότητα εμφάνισης έγχρωμου κειμένου στην οθόνη, πάλι αυτό εξαρτάται από τις δυνατότητες του τερματικού μας (δες και την εισαγωγή στο [1]). Έτσι δεδομένου ότι το τερματικό μας υποστηρίζει χρώμα, εάν δώσουμε στο κέλυφος την εντολή

(george@earth) ~$ **echo "^[[0;31;40mtext in red^[[0;37;40m"**

τότε θα μας εμφανίσει... text in red με κόκκινο χρώμα και μετά θα επαναφέρει το κανονικό λευκό χρώμα.[6]

Παρόλα αυτά, εμείς θέλουμε να προγραμματίσουμε σε γλώσσα C. Πώς μπορούμε να πετύχουμε το ίδιο αποτέλεσμα με εντολές C; Ένας τρόπος, ενδεχομένως να τον μαντέψατε, είναι ο εξής:

```
#include <stdio.h>

int main(void)
{
  char escape=27;
  printf("%c[0;31;40mtext in red%c[0;37;40m\n", escape, escape);
  return 0;
}
```

Θα μπορούσαμε λοιπόν γνωρίζοντας όλες αυτές τις ακολουθίες χαρακτήρων διαφυγής να φτιάξουμε συναρτήσεις στις οποίες να δίναμε σαν παράμετρο το χρώμα που θέλουμε. Μια λίστα με ακολουθίες σαν την `^[[0;31;40m` θα βρείτε στο [4].

Όπως φαίνεται στο παράρτημα 4.1 έτσι δουλεύει και η βιβλιοθήκη asciiart δίνοντας μας προς χρήση 8 προκαθορισμένα χρώματα. Η ncurses με παρόμοιο τρόπο δίνει 16 χρώματα για την εμφάνιση των χαρακτήρων, αλλά επιπλέον μας επιτρέπει να αλλάξουμε και το χρώμα του παρασκηνίου (ή αλλιώς φόντου, άμα προτιμάτε)!

Πριν περάσουμε στην ουσία αυτού του κειμένου — που είναι βεβαίως οι βιβλιοθήκες αυτές καθαυτές — θα ήθελα να σταθώ σε ένα ακόμη πρόβλημα που θέλει τη λύση του…

[5] Σας θυμίζω ότι το συνδυασμό πλήκτρων [Ctrl-V + key] το χρησιμοποιούμε και στο κειμενογράφο Vim· εδώ τον εισάγουμε στη γραμμή εντολών.

[6] Σε περίπτωση που κάτι πάει στραβά, δώστε στο τερματικό σας την εντολή `reset` και όλα θα επανέλθουν στην αρχική τους κατάσταση.

1.7 Το πρόβλημα με τη scanf()

Αυτή η συνάρτηση είμαι βέβαιος ότι έχει προβληματίσει όλους όσους έχουνε ασχοληθεί με τη γλώσσα C. Πολλές φορές ίσως να έχει προκαλέσει και τον εκνευρισμό των αρχάριων προγραμματιστών όταν αυτοί έρχονται για πρώτη φορά αντιμέτωποι με μια παρενέργειά της που θα εξετάσουμε σε αυτή τη παράγραφο.

Το πρόγραμμα που ακολουθεί ζητά από το χρήστη να μαντέψει έναν ακέραιο στο διάστημα 1 έως και 10 μέχρι να τον βρει. Εάν ο χρήστης θέλει, μπορεί να ξαναπαίξει.

Πρόγραμμα 1: asnc-source-1.0/intro/program1.c

```
1   /* program1.c :
      A simple guessing game: Guess an integer between 1-10. */

    #include <stdio.h>
5   #include <stdlib.h>
    #include <time.h>

    int main(void) {
      int num=0, guess=0, c=0;
10
      srand(time(NULL)); /* randomization seed */

      do { puts("Guess a number between 1-10.");
          num = rand()%10 +1;  /* 1, 2, ..., 10 */
15
          do { printf("Your guess is: ");
              scanf("%d", &guess);
              if( guess!=num) puts("Sorry, try again!");

20        } while( num!=guess );

          puts("Yes, that's it!");
          puts("Play again? (Q: quits)");
          c=getchar();
25
      } while( c!=EOF && c!='Q' && c!='q' );

    return EXIT_SUCCESS; }
```

Το πρόγραμμα αυτό δεν εκτελείται όπως θα περιμέναμε· η γραμμή 17 ευθύνεται για αυτό. Ορίστε ένα παράδειγμα του εκτελέσιμου:

```
(george@earth) ~/asnc-source-1.0/intro$ ./program1
Guess a number between 1-10.
Your guess is: 4
Sorry, try again!
Your guess is: 5
Yes, that's it!
Play again? (Q: quits)
Guess a number between 1-10.
Your guess is:
...
```

Παρατηρείστε ότι το πρόγραμμα δεν κάνει αυτό που του ζητάμε. Η εντολή *getchar()* στη γραμμή 24 θέλουμε να πάρει ένα χαρακτήρα από το ρεύμα εισόδου (stdin) και εάν είναι q να τερματιστεί η εκτέλεση. Αυτή όμως φαίνεται να αγνοείται και η ροή του προγράμματος συνεχίζει το βρόχο περνώντας με επιτυχία (TRUE) τις συνθήκες του while στη γραμμή 26.

Μάλιστα, η παραπάνω συμπεριφορά ισχύει μόνο στη περίπτωση που δώσουμε αριθμό όπως μας ζητάει το πρόγραμμα. Για δοκιμάστε να δώσετε σαν είσοδο ένα άλλο χαρακτήρα και δείτε τι θα συμβεί…

Τι γίνεται λοιπόν; Πράγματι αγνοείται η *getchar()*; Όχι βέβαια! Αυτή δουλεύει σωστά. Υπεύθυνη για αυτή την ανεπιθύμητη συμπεριφορά είναι η *scanf()*, η οποία ναι μεν διαβάζει τα ορίσματα ακολουθώντας πιστά τη μορφή που της λέμε, αλλά αγνοεί όλους τους «λευκούς» χαρακτήρες που συναντά (κενό διάστημα – \x20, χαρακτήρας νέας γραμμής – \n, χαρακτήρας στηλογνώμονα – \t) και παρεμβάλλονται μεταξύ αυτών! Οπότε κάπως έτσι αντιμετωπίζει και το τελικό \n που πληκτρολογούμε για να δηλώσουμε το τέλος της εισαγωγής των δεδομένων· το αγνοεί, αφού μετά και το τελευταίο όρισμα που αναθέτει σε μια μεταβλητή υπάρχει λευκός χαρακτήρας, ο χαρακτήρας νέας γραμμής. Αυτός παραμένει στο ρεύμα εισόδου και τον διαβάζει η *getchar()* στη συνέχεια.

Η λύση στο πρόβλημα είναι προφανής. Προσθέτουμε μια κλήση της *getchar()* ακριβώς μετά τη *scanf()* στη γραμμή 17. Αυτό μας επιτρέπει να ξεφορτωθούμε το \n που έχει μείνει στο buffer του ρεύματος εισόδου και να κάνουμε τελικά το πρόγραμμα να δουλέψει όπως επιθυμούμε:

```
(george@earth) ~/asnc-source-1.0/intro$ ./program1
Guess a number between 1-10.
Your guess is: 3
Sorry, try again!
Your guess is: 6
Yes, that's it!
Play again? (Q: quits)
r
Guess a number between 1-10.
Your guess is: 2
Yes, that's it!
Play again? (Q: quits)
q
(george@earth) ~/asnc-source-1.0/intro$
```

Το συγκεκριμένο πρόγραμμα είναι αρκετά απλό και ίσως δεν έχει σημασία, όμως στη γενική περίπτωση θα θέλαμε να έχουμε καλύτερο έλεγχο στο ρεύμα εισόδου stdin, που είναι τα δεδομένα που εισάγει ο χρήστης. Υπάρχει λοιπόν μια άλλη λύση που προτείνω: Να αντικαταστήσουμε το ζευγάρι των συναρτήσεων *scanf() – getchar()* με το ζευγάρι *fgets() – sscanf()*. Οπότε και έχουμε τις εντολές:

```
scanf("%d", &guess); getchar();
```

να αλλάζουν σε:

```
char buffer[1024]; /* input line from console */

...

fgets(buffer, sizeof buffer, stdin);
sscanf(buffer, "%d", &guess);
```

Με αυτό τον τρόπο κερδίζουμε σε δύο σημεία. Καταρχήν μπορούμε να χρησιμοποιήσουμε την *getchar()* της γραμμής 24 με ασφάλεια, αφού είμαστε σίγουροι πως θα δράσει όπως επιθυμούμε και έπειτα μας μένει ο πίνακας buffer διαθέσιμος για οποιαδήποτε επεξεργασία θέλουμε· π.χ. έλεγχος εγκυρότητας των στοιχείων που έδωσε ο χρήστης: Εάν ο χρήστης μας δώσει σαν είσοδο ένα γράμμα θα το γνωρίζουμε εγκαίρως και δε θα αφήσουμε το πρόγραμμα να μπει σε ατέρμων βρόχο.

Για αυτό το θέμα θα ήθελα να παραθέσω ένα μικρό απόσπασμα από το [7]:

> «The function `scanf` works like `printf`, except that `scanf` reads numbers instead of writing them. `scanf` provides a simple and easy way of reading numbers *that almost never works*. The function `scanf` is notorious for its poor end-of-line handling, which makes scanf useless for all but an expert.
>
> However, we've found a simple way to get around the deficiencies of `scanf` — we don't use it. Instead, we use `fgets` to read a line of input and `sscanf` to convert the text into numbers.»

2 Βιβλιοθήκη asciiart

2.1 Εισαγωγή

Η βιβλιοθήκη asciiart είναι μια μικρή βιβλιοθήκη που γράφτηκε για τις ανάγκες του μαθήματος προγραμματισμού της γλώσσας C στο Linux από τον καθηγητή πληροφορικής Eric Berthomier και το συνεργάτη του Laurent Signac (βλέπε [5]). Το όνομά της μπορεί να είναι πομπώδες για τις δυνατότητες της, (όπως λέει ο Berthomier στην εισαγωγή στο παράρτημα Β του [5]), αλλά δε μένει παρά σε εμάς τους ίδιους να τη βελτιώσουμε!

Η σημερινή μορφή της επιτρέπει τα εξής:

α) τη χρήση 8 χρωμάτων (μαύρο, κόκκινο, πράσινο, κίτρινο, μπλε, φούξια, γαλάζιο, λευκό),

β) τη γνώση των χαρακτήρων από το ρεύμα εισόδου stdin, με ή χωρίς ηχώ,

γ) την εμφάνιση κειμένου με την ιδιότητα του αναβοσβήματος (blinking text), (πειραματικά για την ώρα, δε δουλεύει παντού· εξαρτάται από το εάν υποστηρίζεται από το τερματικό μας).

2.2 Η χρήση της βιβλιοθήκης

Η βιβλιοθήκη αποτελείται από τα αρχεία `asciiart.c, asciiart.h`. Για να τη χρησιμοποιήσουμε, πρέπει να βάλουμε στο κώδικά μας το `#include "asciiart.h"`, και αφού βεβαιωθούμε ότι και τα δύο αρχεία βρίσκονται στο τρέχοντα κατάλογο δίνουμε την εντολή:

```
(george@earth) ~/asnc-source-1.0/asciiart$ gcc -c asciiart.c
```

Η οποία και θα παράγει τον αντικειμενικό κώδικα στο αρχείο `asciiart.o`. Έστω ότι το πρόγραμμά μας είναι το ακόλουθο:

Πρόγραμμα 2: asnc-source-1.0/asciiart/program2.c

```
1  /* program2.c :
     Demonstrates asciiart's library capabilities. */

   #include <stdio.h>
5  #include "asciiart.h"

   int main(void)
   {
     test_asciiart();
10   return 0;
   }
```

Η συνάρτηση *test_asciiart()* περιέχεται στη βιβλιοθήκη asciiart και αυτό που κάνει είναι να παρουσιάζει τις δυνατότητές της. Για να το μεταγλωττίσουμε δίνουμε:

```
(george@earth) ~/asnc-source-1.0/asciiart$ gcc -c program2.c
(george@earth) ~/asnc-source-1.0/asciiart$ gcc -o program2 program2.o asciiart.o
```

Τώρα μπορούμε να τρέξουμε το εκτελέσιμο:

```
(george@earth) ~/asnc-source-1.0/asciiart$ ./program2
```

2.3 Οι συναρτήσεις

Εκτός από τη συνάρτηση *test_asciiart()* που δε μας είναι χρήσιμη για τα προγράμματά μας στο `asciiart.h` περιλαμβάνονται τα πρωτότυπα των συναρτήσεων:

`char getch(void);`

Άμεση γνώση ενός χαρακτήρα χωρίς ηχώ.

`char getche(void);`

Άμεση γνώση ενός χαρακτήρα με ηχώ.

`void textcolor(int i);`

Αλλαγή του χρώματος των χαρακτήρων στο χρώμα που αντιστοιχεί στην τιμή *i*. Πρέπει $0 \leq i \leq 7$.
Αναλυτικά είναι:

0	μαύρο	4	μπλε
1	κόκκινο	5	φούξια (ματζέντα)
2	πράσινο	6	γαλάζιο (κυανό)
3	κίτρινο	7	λευκό

`void textblink(int val);`

Ενεργοποίηση αναβοσβήματος χαρακτήρων. Είναι πειραματική ακόμη όπως λέει στο asciiart.h (βλέπε 4.1), και δυστυχώς δε μπορούμε να τη χρησιμοποιήσουμε σε όλα τα τερματικά. Πρέπει *val*=1 ή *val*=0:

1	Ενεργοποίηση λειτουργίας αναβοσβήματος
0	Απενεργοποίηση της λειτουργίας

`void textreset(void);`

Επαναφορά της κονσόλας στην αρχική κατάσταση.

Στο τέλος αυτού του εγγράφου θα βρείτε μερικά παραρτήματα με πληροφορίες που θα σας βοηθήσουν στις προγραμματιστικές σας δραστηριότητες. Εκεί επίσης έχω συμπεριλάβει τα αρχεία `asciiart.h` και `asciiart.c`. Διαβάστε τα στη παράγραφο 4.1.

2.4 Παράδειγμα 1: Βελτίωση του προγράμματος program1.c (σελ. 4)

Τώρα που γνωρίζουμε τις συναρτήσεις της βιβλιοθήκης asciiart ας «ομορφήνουμε» το πρόγραμμα που είδαμε στη παράγραφο 1.7, σελίδα 4.

Προσέθεσα χρωματιστά μηνύματα και αντικατέστησα την *getchar()* με την *getch()*, απενεργοποιώντας με αυτό το τρόπο την ηχώ. Επίσης στο τέλος κάνω κλήση της *textreset()* αν και δε χρειάζεται. Απλώς μου δίνει την ευκαιρία να επισημάνω ότι έτσι είμαστε σίγουροι πως επιστρέφοντας στο κέλυφος του Linux δε θα έχουμε ενεργοποιημένη κάποια ανεπιθύμητη ιδιότητα που έχουμε ξεχάσει να απενεργοποιήσουμε μέσα στο πρόγραμμα.

Επαναλαμβάνω τις οδηγίες για τη μεταγλώττιση που είδαμε σε προηγούμενη παράγραφο:

```
(george@earth) ~/asnc-source-1.0/asciiart$ gcc -c program3.c
(george@earth) ~/asnc-source-1.0/asciiart$ gcc -o program3 program3.o asciiart.o
```

Η εντολή `gcc -c asciiart.c` είναι περιττή αφού το `asciiart.o` υπάρχει ήδη στον τρέχοντα κατάλογο.

Πρόγραμμα 3: asnc-source-1.0/asciiart/program3.c

```
/* program3.c :
  A simple guessing game: Guess an integer between 1-10.
  (program1.c with asciiart.h) */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "asciiart.h"

int main(void) {
   int num=0, guess=0, c=0;

   srand(time(NULL)); /* randomization seed */

   do { puts("Guess a number between 1-10.");
        num = rand()%10 +1;  /* 1, 2, ..., 10 */

        do { printf("Your guess is: ");
             scanf("%d", &guess); getch(); /* still buggy, use fgets!
*/
             if( guess!=num) { textcolor(1); /* red */
                       puts("Sorry, try again!");
                       textcolor(7); /* white */ }

        } while( num!=guess );

        textcolor(2); /* green */
        puts("Yes, that's it!");
        textcolor(7); /* white */
        puts("Play again? (Q: quits)");
        c=getch(); /* no echo */

   } while( c!=EOF && c!='Q' && c!='q' );

   textreset();

return EXIT_SUCCESS; }
```

2.5 Παράδειγμα 2: Επέκταση της βιβλιοθήκης asciiart: έντονα γράμματα

Σε αυτό το παράδειγμα θα επιχειρήσω να προσθέσω μια ακόμη δυνατότητα στη βιβλιοθήκη asciiart: να μπορούμε να τυπώνουμε μηνύματα με έντονα γράμματα στο τερματικό. Ήδη είδαμε πως πετυχαίνουμε εμφάνιση χρώματος και πως να τυπώνουμε κείμενο που αναβοσβήνει (βλέπε στη παράγραφο 4.1, στο αρχείο `asciiart.c`, γραμμές 40-45 και 47-57).

Συμβουλευόμενος το [4] για την ιδιότητα έντονα γράμματα κατασκεύασα την κεφαλίδα που ακολουθεί:

Επέκταση - έντονα γράμματα: asnc-source-1.0/asciiart/asciiart-ext.h

```
/*    Color Definitions     */

#define BLACK   0
#define RED     1
#define GREEN   2
#define YELLOW  3
#define BLUE    4
#define MAGENTA 5
#define CYAN    6
#define WHITE   7

/*    escape key         */

char escape=27;

/*    Activate + Deactivate  */

#define SET_ON       1     /* activate */
#define SET_OFF      0     /* deactivate */

/*    Bold escape sequence   */

char *b_on="[1m";     /* set bold attribute */
char *b_off="[0m";    /* unset bold attribute */

/*    Bold function       */

void textbold(int val)
{
  if (val==SET_ON)
    printf("%c%s", escape, b_on);
  else
    printf("%c%s", escape, b_off);
}

```

Όπως βλέπετε η συνάρτηση *textbold()* λειτουργεί με τον ίδιο τρόπο με τη *textblink()* στο `asciiart.c`. Για να ενεργοποιήσουμε την ιδιότητα για εμφάνιση έντονων χαρακτήρων δίνουμε την εντολή *textbold(SET_ON)* και για να την απενεργοποιήσουμε την εντολή *textbold(SET_OFF)*. Ας τη δοκιμάσουμε με ένα απλό πρόγραμμα:

Πρόγραμμα 4: asnc-source-1.0/asciiart/program4.c

```
#include <stdio.h>
#include "asciiart.h"
#include "asciiart-ext.h"

int main() {

  textbold(SET_ON);
```

```
    puts("This text is bold.");
    textbold(SET_OFF);
    puts("But this text is normal.");
    textbold(SET_ON);
    textblink(SET_ON);
    textcolor(YELLOW);
    puts("Warning!");
    textreset();
    return 0;

}
```

Κατεβάστε τον κώδικα από το `http://www.tem.uoc.gr/~gbandreo` και κατά το γνωστό τρόπο κάντε τη μεταγλώττιση. Όταν τρέξετε το πρόγραμμα θα παρατηρήσετε κάτι εντυπωσιακό: ο συνδυασμός έντονα γράμματα και χρώμα μας δίνει πιο φωτεινά χρώματα! (Π.χ. δείτε τις γραμμές 11-13 στο πρόγραμμα 4). Έτσι τα 8 χρώματα της βιβλιοθήκης γίνονται τώρα 16! Και είναι πράγματι 16 και όχι 15 γιατί ακόμα και το μαύρο μπορεί να αλλάξει. Το έντονο μαύρο αντιστοιχεί στο σκούρο γκρι. Συμβουλευόμενοι το [4], θα πρέπει να δώσουμε την παρακάτω εντολή για να πάρουμε το χρώμα αυτό:

(george@earth) ~$ **echo "^[[1m^[[30;40mDark Grey"; tput sgr0**

2.6 Κάντε τις δικές σας προσθήκες στη βιβλιοθήκη

Τελειώνοντας εδώ με την παρουσίαση αυτής της πολύ χρήσιμης βιβλιοθήκης, θα ήθελα να σας παροτρύνω να φτιάξετε και δικές σας συναρτήσεις. Σαν ιδέα σας δίνω να βρείτε πώς μπορούμε να έχουμε χρώμα στο παρασκήνιο (background color) και πώς μπορεί να γίνει η μετακίνηση του δρομέα σε κάποια θέση της κονσόλας (βλέπε για πληροφορίες για τα σχετικά escape sequences στο [4]). Εκτός από αυτά τα απλούστερα, προτείνω και κάτι πιο σύνθετο: φτιάξτε μια συνάρτηση αυτόματης κατασκευής μενού πολλαπλών επιλογών όπου η αρίθμηση θα έχει διαφορετικό χρώμα από το υπόλοιπο κείμενο.

3 Βιβλιοθήκη ncurses

3.1 Εισαγωγή

Όπως είδαμε σε προηγούμενες παραγράφους, για οτιδήποτε θελήσουμε να αλλάξουμε στο τερματικό μας θα πρέπει να γνωρίζουμε την ακολουθία διαφυγής που το πετυχαίνει αυτό. Οι διάφορες εφαρμογές λοιπόν αναζητούν στο αρχείο `terminfo` αυτή τη πληροφορία. Όμως, όπως ήδη φάνηκε με τη βιβλιοθήκη asciiart, ο προγραμματισμός για το χειρισμό των ιδιοτήτων του τερματικού μας, δυσκολεύει καθώς προσθέτουμε νέες συναρτήσεις. Η βιβλιοθήκη curses (λογοπαίγνιο του *cursor optimization*) δίνει τη λύση σε αυτό το χάος.

Αυτό που κάνει είναι να λειτουργεί σαν φίλτρο ανάμεσα σε εμάς και το τερματικό μας ώστε εμείς να μην ασχολούμαστε καθόλου με ακολουθίες διαφυγής. Αποτελεί ένα ευέλικτο API (Application User Interface) παρέχοντάς μας μεγάλο πλήθος συναρτήσεων που μεταξύ άλλων μετακινούν το δρομέα, αλλάζουνε χρώματα, δημιουργούν παράθυρα και χειρίζονται το ποντίκι.

Η ncurses (new curses) αποτελεί κλώνο της παλιάς curses από το System V Release 4 και εξέλιξη της curses από το 4.4BSD. Είναι πλήρως συμβατή με παλαιότερες εκδόσεις και διανέμεται ελεύθερα με το πηγαίο κώδικα. Θα τη βρείτε στον επίσημο ιστοτόπο `http://dickey.his.com/ncurses`.

Στις επόμενες σελίδες θα επιχειρήσω να παρουσιάσω τις πιο σημαντικές συναρτήσεις της βιβλιοθήκης. Περισσότερες πληροφορίες θα βρείτε στις Αναφορές, στο τέλος αυτού του κειμένου. Επίσης είναι χρήσιμο να διαβάσετε το on-line manual δίνοντας `man ncurses` στη γραμμή εντολών.

3.2 Χρήση της ncurses

Η βιβλιοθήκη βρίσκεται στο σύστημά μας στο `/usr/include/curses.h` και `/usr/include/-ncurses.h`, οπού το ένα είναι συμβολικός δεσμός του άλλου, οπότε εμείς μπορούμε να χρησιμοποιούμε και τις δύο εκδοχές στα προγράμματά μας:

`#include <curses.h>` και μεταγλώττιση προσθέτοντας `-lcurses`,

είτε

`#include <ncurses.h>` και μεταγλώττιση με `-lncurses`,

είναι ακριβώς το ίδιο και δεν έχει σημασία.[7]

Σε αυτό το σημείο να αναφέρω πως όπως λένε τα [1] και [9] δε χρειάζεται να συμπεριλάβουμε την εντολή `#include <stdio.h>` γιατί αυτό γίνεται από την `curses.h`. Ωστόσο επειδή έχουμε συνηθίσει να το γράφουμε πρώτο πρώτο στα προγράμματά μας δεν είναι κάτι που θα πρέπει να μας προβληματίσει. Απλά…είναι περιττό.

3.3 Το Α και το Ω

Για τη βιβλιοθήκη η οθόνη του τερματικού μας είναι ένας πίνακας μ γραμμών και ν στηλών. Κάθε μια θέση αυτού του πίνακα περιέχει πληροφορία όχι μόνο για το χαρακτήρα που βρίσκεται εκεί αλλά και για τις ιδιότητές του. Αυτόν τον πίνακα τον δηλώνει σαν ένα δείκτη σε μια δομή που ονομάζει WINDOW.

Επίσης όλα τα προγράμματα που κάνουνε χρήση της βιβλιοθήκης πρέπει να έχουνε:[8]

α) Πριν από οποιαδήποτε άλλη συνάρτηση της βιβλιοθήκης curses, την *initscr()* η οποία αρχικοποιεί το τερματικό μας για χρήση. Δεσμεύει μνήμη για μια δομή curscr (current screen) που

[7] Στο εξής, όταν γράφω ncurses θα αναφέρομαι αποκλειστικά στην ncurses του Linux, ενώ όταν γράφω curses θα υπονοώ πως τα γραφόμενα εφαρμόζονται και σε άλλα συστήματα.

[8] Για όλες τις συναρτήσεις που παρουσιάζονται από εδώ μέχρι και το τέλους του κεφαλαίου θα πάρετε αναλυτικές πληροφορίες για τη χρήση τους δίνοντας `man` *όνομα συνάρτησης* στο τερματικό σας.

αντιπροσωπεύει αυτό που βλέπουμε στην οθόνη, και μια δομή stdscr (standard screen) που αντιπροσωπεύει αυτό που θα θέλαμε να έχουμε. Δηλαδή, είναι μια εικονική οθόνη στη μνήμη RAM στην οποία κάνουμε τις αλλαγές και τις μορφοποιήσεις που θέλουμε μέχρι μια εντολή *refresh()* να ανανεώσει αυτές τις αλλαγές πάνω στη curscr. Με την κλήση της *initscr()* γίνεται καθαρισμός της οθόνης του τερματικού και γέμισμά της με κενά.

β) Την *refresh()*, η οποία είναι πολύ σημαντική γιατί ανανεώνει την πληροφορία της stdscr στη curscr. Έτσι εάν δοκιμάσουμε να τυπώσουμε ένα μήνυμα, δε θα το δούμε στην οθόνη παρά μόνο όταν φτάσει η ροή του προγράμματος σε μια τέτοια εντολή. Αυτό σημαίνει ότι μπορούμε να κάνουμε πολλαπλές αλλαγές στη stdscr και μετά με μια μόνο κλήση *refresh()* να δούμε το αποτέλεσμα στο τερματικό μας. Η συνάρτηση αυτή ανανεώνει μόνο όση πληροφορία διαφέρει μεταξύ stdscr και curscr.

γ) Την *endwin()*, συνήθως στο τέλος του προγράμματος ή και αρκετά νωρίτερα, αρκεί να μην υπάρχει καμία άλλη συνάρτηση της curses που να την ακολουθεί.[9] Αυτό που κάνει είναι να τερματίζει τον έλεγχο που έχει η curses στο τερματικό μας, να ελευθερώνει όση μνήμη κατέλαβε και να επιστρέφει το τερματικό στη κανονική του μορφή.

Κάθε πρόγραμμα λοιπόν πρέπει να έχει αυτή τη μορφή:

Πρόγραμμα 5: asnc-source-1.0/ncurses/program5.c

```
#include <curses.h>

int main(void)
{
  /* initialize ncurses mode */
  initscr();
  /* print a message on the stdscr */
  printw("What a wonderful library!");
  /* make the message appear (update curscr) */
  refresh();
  /* wait for the user to press a key */
  getch();
  /* terminate ncurses mode */
  endwin();
  return 0;
}
```

Σημείωση: *Η συνάρτηση printw() τυπώνει ένα μορφοποιημένο μήνυμα στην οθόνη (βλέπε 3.6) ενώ η getch() διαβάζει ένα χαρακτήρα (βλέπε 3.7).*

Ακόμη θα πρέπει να γνωρίζουμε και τα εξής:

Η βιβλιοθήκη curses χρησιμοποιεί δύο μεταβλητές για να περιγράψει τις διαστάσεις του τερματικού:

τύπος	όνομα	περιγραφή
int	LINES	αριθμός γραμμών του τερματικού
int	COLS	αριθμός στηλών του τερματικού

Οπότε το τερματικό μας με βάση αυτές τις μεταβλητές θα έχει συντεταγμένες από (0,0)· πρώτη στήλη και γραμμή, μέχρι (LINES-1, COLS-1)· τελευταία στήλη και γραμμή. Η συνάρτηση που μας

[9] Μια εξαιρεσούλα υπάρχει στην αμέσως επόμενη παράγραφο.

επιτρέπει να πηγαίνουμε σε όποια θέση θέλουμε είναι η *move(row,col)*, δηλαδή αυτό που κάνει, είναι να τοποθετεί το δρομέα στη θέση *(row,col)* της οθόνης.

Η βιβλιοθήκη εισάγει ένα νέο τύπο και δηλώνει κάποιες σταθερές που μας βοηθάνε πολύ στο προγραμματισμό, και τις οποίες συναντούμε και στις man pages:

bool	$\Rightarrow$	δυαδικός τύπος (στη πραγματικότητα είναι unsigned char)
TRUE	$\Rightarrow$	αντιστοιχεί στο 1 (π.χ. `bool has_kids=TRUE;`)
FALSE	$\Rightarrow$	αντιστοιχεί στο 0
ERR	$\Rightarrow$	επιστρέφεται από τις συναρτήσεις όταν αποτυγχάνουν (-1)
OK	$\Rightarrow$	επιστρέφεται όταν όλα είναι εντάξει (αντιστοιχεί στο 0)

3.4 Ανάμικτος προγραμματισμός (mixing standard functions with curses)

Η ncurses από τη στιγμή που θα αναλάβει τη διαχείριση του τερματικού μας, περιθωριοποιεί τις συναρτήσεις εισόδου/εξόδου της πρότυπης βιβλιοθήκης. Έτσι η κλήση:

```
printf("Can you see me?");
```

δε θα εμφανίσει τίποτα στην οθόνη μας! Εάν ωστόσο το επιθυμούμε μπορούμε να βγούμε προσωρινά από το περιβάλλον της ncurses και να επιστρέψουμε σε αυτό αργότερα. Ορίστε το προηγούμενο παράδειγμα στη γενικευμένη του μορφή:

Πρόγραμμα 6: asnc-source-1.0/ncurses/program6.c

```
#include <curses.h>
#include <stdlib.h>

int main(void) {
  printf("Normal mode is so boring!");
  getchar();                                    /* pause */

  initscr();
  printw("Curses mode is nice!\n");
  refresh();
  getch();                                     /* pause */

  def_prog_mode();
  endwin();

  /* type exit to quit the shell */
  system("/bin/sh -c 'echo \"But necessary sometimes\"';/bin/sh");

  reset_prog_mode();
  refresh();

  printw("Nice is a small word. It's WONDERFUL!\n");
  refresh();
  getch(); /* pause */

  endwin();

  printf("Man, this is so cool! I want to know more about curses!\n");
  return EXIT_SUCCESS;
}
```

Όπως φαίνεται και στο πρόγραμμα για να εγκαταλείψουμε προσωρινά το περιβάλλον της βιβλιοθήκης curses, θα πρέπει να γράψουμε με αυτή τη σειρά τις συναρτήσεις:[10]

def_prog_mode()	Με αυτή τη συνάρτηση αποθηκεύονται οι τρέχουσες ρυθμίσεις του τερματικού όπως έχουνε διαμορφωθεί μέσα στο πρόγραμμα με σκοπό να τις ανακαλέσουμε αργότερα.
endwin()	Εγκαταλείπει τα παράθυρα της βιβλιοθήκης ncurses και επαναφέρει τη συμπεριφορά του τερματικού στη κανονική του κατάσταση.
reset_prog_mode()	Επιστρέφει στην προηγούμενη κατάσταση ανακαλώντας τις ρυθμίσεις που είχαμε σώσει με την *def_prog_mode()*.
refresh()	Επαναφέρει την οθόνη στην κατάσταση που την είχαμε αφήσει, αντιγράφοντας τον πίνακα των μ επί ν στοιχείων από την προσωρινή μνήμη στην οθόνη.

3.5 Συναρτήσεις αρχικοποίησης (initialization functions)

Αμέσως μετά την *initscr()* πολλές φορές στα προγράμματά μας καλούμε ορισμένες συναρτήσεις για να αρχικοποιήσουμε κάποιες παραμέτρους με ιδιότητες που επιθυμούμε. Αργότερα μέσα στο πρόγραμμά μας μπορούμε να καλέσουμε τις αντίστοιχες συναρτήσεις που αναιρούν αυτές τις αρχικές συνθήκες και γενικά μπορούμε να τις εναλλάσσουμε όπως θέλουμε. Ας δούμε τις πιο βασικές συναρτήσεις που ρυθμίζουν την αλληλεπίδρασή μας με το σύστημα (interactivity):

cbreak()	Με αυτή την εντολή απενεργοποιείται ο stdin buffer. Έτσι κάθε χαρακτήρας που πληκτρολογούμε γίνεται αμέσως γνωστός στο πρόγραμμα. Εάν δώσουμε ένα χαρακτήρα ελέγχου όπως Ctrl-Z (suspend) ή Ctrl-C (interrupt) τότε αυτός θα λειτουργήσει ως συνήθως δημιουργώντας το σήμα (signal) suspend ή interrupt αντίστοιχα.
nocbreak()	Ο buffer γίνεται ενεργός, οπότε οι χαρακτήρες που πληκτρολογούμε γίνονται γνωστοί στο πρόγραμμα αφού πατήσουμε Enter. (Επιστροφή του τερματικού στην κανονική του κατάσταση.)
raw()	Απενεργοποίηση του stdin buffer. Η διαφορά του με το *cbreak()* είναι ότι τώρα δε δημιουργείται το σήμα π.χ. interrupt, δίνοντάς μας προγραμματιστικά τη δυνατότητα να χειριστούμε όπως θέλουμε αυτό το συνδυασμό χαρακτήρων (π.χ. κλείσιμο των ανοικτών αρχείων, ελευθέρωση της μνήμης που έχουμε δεσμεύσει δυναμικά μέσα στο πρόγραμμα, εκτύπωση μηνύματος λάθους και επιστροφή στο κέλυφος).
noraw()	Βγάζει το τερματικό από την κατάσταση raw. Τώρα οι χαρακτήρες ελέγχου Ctrl-C (interrupt), Ctrl-/ (quit), Ctrl-Z (suspend) και Ctrl-S, Ctrl-Q (flow control) όταν πατηθούν, στέλνουν το ανάλογο σήμα στο τερματικό.
echo()	Ενεργοποιεί την ηχώ. Οι χαρακτήρες εμφανίζονται καθώς πληκτρολογούνται από τον χρήστη.
noecho()	Απενεργοποιεί την ηχώ. Τώρα οι χαρακτήρες δεν εμφανίζονται. Συνήθως το χρησιμοποιούμε με την *getch()* και μπορούμε να χειριστούμε όπως θέλουμε την ηχώ, π.χ. να εμφανίζουμε τους χαρακτήρες σε μια συγκεκριμένη θέση.

[10] Όπου δε γίνεται λόγος για ορίσματα των συναρτήσεων, σημαίνει ότι είναι τύπου *void* και επιστρέφουν *int*.

curs_set() Ρυθμίζει την κατάσταση του δρομέα σε 0: αόρατος, 1: κανονικός, 2: έντονα ορατός. Εφόσον το τερματικό υποστηρίζει την ορατότητα που ζητήσαμε επιστρέφεται η προηγούμενη τιμή ορατότητας διαφορετικά η τιμή λάθους ERR.

πρωτότυπο: `int curs_set(int visibility);`

keypad() Πολύ χρήσιμη συνάρτηση. Ενεργοποιεί την ανάγνωση των function keys όπως τα F1-F12 και τα βελάκια.[11] Δέχεται δύο ορίσματα. Το πρώτο είναι το παράθυρο στο οποίο θέλουμε να ενεργοποιήσουμε αυτή την ιδιότητα και το δεύτερο είναι η δυαδική τιμή TRUE ή FALSE. Για παράδειγμα για την stdscr καλούμε `keypad(stdscr, TRUE);` Η αρχική (default) τιμή είναι `keypad(stdscr, FALSE);`

πρωτότυπο: `int keypad(WINDOW *win, bool bf);`

Ας δούμε τώρα ένα παράδειγμα που παρουσιάζει όσα είδαμε μέχρι εδώ:

Πρόγραμμα 7: asnc-source-1.0/ncurses/program7.c

```
/*   Tip: Press Ctrl-C and see what happens!   */
#include <curses.h>

int main(void)
{
  int c=0;

  initscr();
  raw();
  noecho();
  keypad(stdscr,TRUE);
  curs_set(0);

  printw("Press F5 to open a shell, or Q to quit the program.");
  refresh();

  do { switch(c=getch())
     {
       case 'q':  /* fall through */
       case 'Q':  /* exit */
         endwin();
         break;
       case KEY_F(5): /* take a trip to the bourne shell */
         def_prog_mode();
         endwin();
         system("/bin/sh");
         reset_prog_mode();
         refresh();
         break;
       case 0x03: /* ctrl-c */
         endwin();
         puts("Interrupt Signal Caught. Program Quits!");
         c='q';
         break;
```

[11] Μια λίστα με όλα τα function keys θα βρείτε στο man curs_getch.

Πρόγραμμα 7 (συνέχεια): asnc-source-1.0/ncurses/program7.c

```
   default: /* some other unprintable character */
          beep();
          break;
     }
   } while (c != 'Q' && c != 'q');

   return 0;
}
```

3.6 Συναρτήσεις εκτύπωσης (output functions)

Μπορούμε να τις χωρίσουμε σε τρεις ομάδες:

α) Εκτύπωση ενός μόνο χαρακτήρα με κάποιες ιδιότητες: *addch(), waddch()* κ.ά.

β) Εκτύπωση ενός αλφαριθμητικού (string): *addstr(), waddnstr()* κ.ά.

γ) Εκτύπωση μορφοποιημένου αλφαριθμητικού: *printw(), wprintw()* κ.ά.

Για το ποια συνάρτηση θα επιλέξουμε να χρησιμοποιήσουμε κάθε φορά, αυτό είναι καθαρά δικιά μας επιλογή, αφού το ίδιο αποτέλεσμα μπορεί να επιτευχθεί με διαφορετικούς τρόπους. Ας δούμε τώρα αναλυτικά αυτές τις συναρτήσεις:

3.6.1 Ομάδα Α: addch()

Οι συναρτήσεις της ομάδας αυτής τυπώνουν ένα χαρακτήρα στην τρέχουσα θέση και προχωρούν τη θέση του δρομέα κατά μια θέση (ανάλογες με την *putchar()* της stdio). Εάν η τρέχουσα θέση είναι η τελευταία στήλη στη τρέχουσα γραμμή τότε ο δρομέας θα μεταφερθεί στην πρώτη στήλη της επόμενης γραμμής. Άμα βρισκόμαστε στη τελευταία γραμμή τότε ο χαρακτήρας θα εμφανιστεί στην πρώτη στήλη της τελευταίας γραμμής. Δηλαδή εξ ορισμού δε γίνεται scroll! Ωστόσο αυτή τη συμπεριφορά μπορούμε να την αλλάξουμε με την εντολή:

```
scrollok(stdscr,TRUE);
```

για το παράθυρο stdscr (βλέπε και 3.12). Η λίστα με τις συναρτήσεις έχει ως εξής:

addch() Τυπώνει ένα χαρακτήρα στο παράθυρο stdscr, π.χ. addch('c');

πρωτότυπο: `int addch(const chtype ch);`

Η παράμετρος είναι τύπου chtype και όχι char όπως θα περιμέναμε να δούμε. Αυτό οφείλεται στο γεγονός ότι χρειαζόμαστε μια μεγαλύτερη μεταβλητή από 1 byte ώστε να κρατήσει και τις ιδιότητες του χαρακτήρα, π.χ. έντονα γράμματα, χρώμα, κ.τ.λ. Ο νέος τύπος για τον χαρακτήρα ορίζεται στο `ncurses.h` σαν:

```
typedef unsigned long chtype;
```

waddch() Τυπώνει ένα χαρακτήρα στο παράθυρο που ορίζουμε, π.χ. waddch(mywin, 'g');

πρωτότυπο: `int waddch(WINDOW *win, const chtype ch);`

mvaddch() Μετακινεί το δρομέα στη θέση που του ορίζουμε, στο stdscr και εκεί τυπώνει τον χαρακτήρα, π.χ. mvaddch(LINES/2, COLS/2, '*');
Είναι το ίδιο με τις εντολές: move(LINES/2, COLS/2); addch('*');

πρωτότυπο: `int mvaddch(int y, int x, const chtype ch);`

mvwaddch() Μετακινεί το δρομέα στη θέση και στο παράθυρο που του ορίζουμε και τυπώνει τον χαρακτήρα, π.χ. mvwaddch(topwin, 6, 22, 'T');

πρωτότυπο:

`int mvwaddch(WINDOW *win, int y, int x, const chtype ch);`

echochar() Έχει το ίδιο αποτέλεσμα σαν να καλέσαμε *addch()* και *refresh()*, π.χ. echochar('?');

πρωτότυπο: `int echochar(const chtype ch);`

wechochar() Είναι ισοδύναμη με το ζευγάρι *waddch()* και *wrefresh()*[12], π.χ. wechochar(status,'K');

πρωτότυπο: `int wechochar(WINDOW *win, const chtype ch);`

Σίγουρα αυτή τη στιγμή θα σκέπτεστε ότι το να τυπώνουμε ένα χαρακτήρα τη φορά δεν έχει κανένα ενδιαφέρον. Όμως αυτό που κάνει αυτές τις συναρτήσεις σημαντικές είναι η δυνατότητα να τις χρησιμοποιούμε για να εμφανίσουμε ένα χαρακτήρα με διαφορετικές ιδιότητες από το υπόλοιπο κείμενο. Αυτό γίνεται με τον τελεστή για bit, | (OR). Για παράδειγμα η εντολή:

```
addch( 'Y' | A_BOLD );
```

Θα εμφανίσει το γράμμα 'Y' στη τρέχουσα θέση με την ιδιότητα έντονα γράμματα. Περισσότερα για τις ιδιότητες θα δούμε στη παράγραφο 3.8, σελίδα 23.

Επίσης ας δούμε το επόμενο πινακάκι με κάποιους από τους ειδικούς χαρακτήρες και τη συμπεριφορά τους σε μια συνάρτηση της ομάδας αυτής.

`addch('\n');`	όλοι οι χαρακτήρες από τη τρέχουσα θέση μέχρι το τέλος της γραμμής γίνονται κενά και ο δρομέας πηγαίνει στην αρχή της επόμενης γραμμής
`addch('\t');`	οι επόμενοι οκτώ χαρακτήρες γίνονται κενά
`addch('\b');`	μια θέση πριν την τρέχουσα θέση γίνεται κενή και ο δρομέας παραμένει στη τρέχουσα θέση
`addch('\r');`	ο δρομέας πηγαίνει στην αρχή της επόμενης γραμμής

Σε αυτό το σημείο θέλω να κάνω μερικές παρατηρήσεις:

1) Για όλες τις συναρτήσεις που παίρνουνε ορίσματα συντεταγμένες θέσης στην οθόνη ακολουθείται η σύμβαση πρώτα να γράφουμε τις γραμμές και μετά τις στήλες. Στο manual συμβολίζεται με *y* η γραμμή και *x* η στήλη και αυτό μπορεί να προκαλέσει σύγχυση (αφού έχουμε συνηθίσει να διαβάζουμε (x,y)). Επομένως χρειάζεται μεγάλη προσοχή!

2) Μια ακόμη σύμβαση στην ονοματολογία των συναρτήσεων που θα συναντήσουμε και στη συνέχεια είναι και η εξής:

[12] `int wrefresh(WINDOW *win);` Κάνει ό,τι και η *refresh()* για το παράθυρο *win*.

α) Κάθε συνάρτηση που αναφέρεται σε ένα παράθυρο γράφεται όπως η συνάρτηση που αναφέρεται στη stdscr, με το πρόθεμα w (δηλ. window), π.χ.

`addch('4' | A_BOLD);` → `waddch(mywin,'4' | A_BOLD);`

`refresh();` → `wrefresh(mywin);`

β) Κάθε συνάρτηση που αλλάζει τη τρέχουσα θέση του δρομέα πριν προβεί σε οποιαδήποτε άλλη αλλαγή γράφεται ως την αντίστοιχη συνάρτηση που δεν αλλάζει τη θέση του δρομέα με το πρόθεμα mv (δηλ. move), π.χ.

`addch('x');` → `mvaddch(LINES-1, COLS-1, 'x');`

3) Στις συναρτήσεις που μετακινούν το δρομέα, οι συντεταγμένες *y,x* είναι *σχετικές* στο παράθυρο στο οποίο αναφέρονται! Έτσι εάν ένα παράθυρο έχει διαστάσεις 5 γραμμές επί 20 στήλες και προσπαθήσουμε να τυπώσουμε στη θέση (2,24) η συνάρτηση θα επιστρέψει τη τιμή λάθους ERR! Αναλυτικά για τα παράθυρα γίνεται λόγος στην παράγραφο 3.10, σελίδα 27.

3.6.2 Ομάδα B: addstr()

Με τις συναρτήσεις αυτής της ομάδας μπορούμε να τυπώνουμε ένα αλφαριθμητικό. Είναι ανάλογες με την εντολή *puts()* της stdio. Ή ανάλογες με τόσες διαδοχικές κλήσεις της *addch()* όσοι είναι και οι χαρακτήρες του αλφαριθμητικού. Ας δούμε ποιες είναι:

addstr()	*waddstr()*	*mvaddstr()*	*mvwaddstr()*
addnstr()	*waddnstr()*	*mvaddnstr()*	*mvwaddnstr()*

Συμβουλευόμενοι τις παρατηρήσεις της προηγούμενης παραγράφου μπορούμε τώρα εύκολα να μαντέψουμε τι κάνει η κάθε συνάρτηση. Ας αναφερθούμε στη γενικότερη από αυτές:

```
int mvwaddnstr(WINDOW *win, int y, int x, const char *str, int n);
```

Η συνάρτηση τυπώνει το πολύ μέχρι *n* χαρακτήρες από το αλφαριθμητικό *str*, (το οποίο τερματίζεται με \0), στο παράθυρο *win*, ξεκινώντας από τη γραμμή *y* και τη στήλη *x* του παραθύρου αυτού. Εάν το *n* είναι αρνητικός τότε τυπώνεται όλο το αλφαριθμητικό μέχρι να συναντήσει το τέλος της γραμμής ή το τερματικό χαρακτήρα \0, οποιοδήποτε από τα δύο συμβεί πρώτο.

Παράδειγμα:

```
mvwaddnstr(top, 0, 3, "Fetching...", -1);
```

Πριν περάσουμε να δούμε και την τελευταία ομάδα των συναρτήσεων εκτύπωσης, ας δούμε ένα ακόμη παράδειγμα:

```
addstr("nCurses");
addnstr("nCurses", -1);
waddstr(stdscr, "nCurses");
waddnstr(stdscr, "nCurses", -1);
mvaddstr(0, 0, "nCurses");
mvaddnstr(0, 0, "nCurses", -1);
mvwaddstr(stdscr, 0, 0, "nCurses");
mvwaddnstr(stdscr, 0, 0, "nCurses", -1);
```

Παρατηρείστε ότι οι παραπάνω εντολές κάνουν ακριβώς το ίδιο πράγμα! (Θεωρούμε ότι η τρέχουσα θέση είναι η (0,0)). Εδώ μπορούμε να πούμε ότι φαίνεται και η φιλοσοφία της βιβλιοθήκης curses: Τα πάντα αναφέρονται σε ένα παράθυρο και σε συντεταγμένες του παραθύρου αυτού.

3.6.3 Ομάδα Γ: printw()

Η τρίτη ομάδα συναρτήσεων εκτύπωσης, αν και αναφέρεται τελευταία, είναι και η πιο χρήσιμη, αφού με αυτές τις συναρτήσεις μπορούμε να τυπώσουμε μορφοποιημένο κείμενο. Με άλλα λόγια είναι η γενικότερη των προηγούμενων ομάδων και μάλιστα εάν χρησιμοποιούσαμε θεωρία συνόλων θα μπορούσαμε να γράψουμε ομάδα Α $\subseteq$ ομάδα Β $\subseteq$ ομάδα Γ. Έτσι, οι επόμενες συναρτήσεις λειτουργούν όπως οι αντίστοιχες συναρτήσεις των βιβλιοθηκών stdio και stdarg, *printf()* και *vprintf()*:

printw() Τυπώνει ένα μορφοποιημένο κείμενο στο παράθυρο stdscr ξεκινώντας από την τρέχουσα θέση. Στις περιπτώσεις όπου γίνεται υπερχείλιση της γραμμής ή της σελίδας ακολουθείται ό,τι έχουμε ήδη αναφέρει, π.χ.
`printw("I love my pets. I got %d cats.\n", niaou);`

wprintw() Τυπώνει το μορφοποιημένο κείμενο στο παράθυρο που ορίζουμε, στη τρέχουσα θέση για το παράθυρο αυτό, π.χ.
`wprintw(StatusWIN, "%s", message);`

mvprintw() Πρώτα γίνεται μετακίνηση του δρομέα σε νέα θέση και μετά τυπώνεται το κείμενο, στο παράθυρο stdscr, π.χ.
`mvprintw(3,8,"Ready for launch.");`

mvwprintw() Τυπώνει το κείμενο στο παράθυρο και στη θέση που ορίζουμε, π.χ.
`mvwprintw(popupwin, row, col, "Failure at %d", berror);`

vwprintw() Δέχεται μια λίστα μεταβλητών ορισμάτων τα οποία τυπώνει σε ένα παράθυρο ξεκινώντας από τη τρέχουσα θέση.

πρωτότυπο:

```
int vwprintw(WINDOW *win, char *fmt, va_list varglist);
```

vw_printw() Κάνει ακριβώς το ίδιο με την vwprintw(). Απλώς, στη χρήση των δύο αυτών συναρτήσεων τίθενται ορισμένα ζητήματα μεταφερσιμότητας, όπως λέει και στο `man curs_printw`. Για το Linux, μπορούμε να χρησιμοποιούμε και τις δύο, προτιμητέα ωστόσο είναι η vw_printw().

πρωτότυπο:

```
int vw_printw(WINDOW *win, char *fmt, va_list varglist);
```

3.7 Συναρτήσεις εισαγωγής (input functions)

Στην προηγούμενη παράγραφο είδαμε τις δυνατότητες της βιβλιοθήκης curses για εκτύπωση στην οθόνη. Αυτό όμως δεν είναι αρκετό για να γράφουμε αμφίδρομα προγράμματα. Θέλουμε επίσης να διαβάζουμε δεδομένα από το χρήστη.

Η βιβλιοθήκη curses μας παρέχει ορισμένες συναρτήσεις εισαγωγής οι οποίες, κατά αναλογία με τις συναρτήσεις εκτύπωσης, χωρίζονται σε τρεις ομάδες:

α) Ανάγνωση ενός χαρακτήρα: *getch(), wgetch()* κ.ά.

β) Ανάγνωση ενός αλφαριθμητικού: *getstr(), wgetstr()* κ.ά.

γ) Ανάγνωση μορφοποιημένου αλφαριθμητικού: *scanw(), wscanw()* κ.ά.

3.7.1 Ομάδα A: getch()

Περιλαμβάνει τις συναρτήσεις:

getch() Διάβασμα ενός χαρακτήρα από το παράθυρο stdscr.

πρωτότυπο: `int getch(void);`

wgetch() Διάβασμα ενός χαρακτήρα από το παράθυρο που ορίζουμε.

πρωτότυπο: `int wgetch(WINDOW *win);`

mvgetch() Διάβασμα ενός χαρακτήρα από τη θέση που ορίζουμε στο παράθυρο stdscr.

πρωτότυπο: `int mvgetch(int y, int x);`

mvwgetch() Το ίδιο αλλά τώρα ορίζουμε και το παράθυρο.

πρωτότυπο: `int mvwgetch(WINDOW *win, int y, int x);`

ungetch() Επιστρέφει ένα χαρακτήρα στον buffer, έτσι ώστε αυτός να διαβαστεί από την επόμενη εντολή getch(). Για όλα τα παράθυρα υπάρχει ένας μόνο input buffer.

πρωτότυπο: `int ungetch(int ch);`

Αυτές οι συναρτήσεις συνδυάζονται με τις συναρτήσεις αρχικοποίησης που είδαμε στη παράγραφο 3.5. Έτσι για να διαβάσουμε ένα χαρακτήρα χωρίς ηχώ και χωρίς να χρειάζεται να πατήσουμε το Enter γράφουμε:

(παράδειγμα 1)

```
 [...]

cbreak();
noecho();

ch=wgetch(mywin);
switch(ch) {
           case '1':
                  function_1();
                  break;
           case '2':
                  function_2();
                  break;
 [...]
          default:
                  beep();
                  break;}
 [...]
```

(παράδειγμα 2)

```
 [...]
cbreak();
noecho();

keypad(stdscr, TRUE);
ch=getch();
switch(ch) {
           case KEY_DOWN:
                  next_item();
                  break;
           case KEY_F(1):
                  info();
                  break;
 [...]
           default:
                  beep();
                  break;}
 [...]
```

Στο παράδειγμα 1, διαβάζουμε ένα χαρακτήρα από το παράθυρο mywin. Στη συνέχεια γίνεται έλεγχος εάν διαβάσαμε το 1, 2, 3 κ.τ.λ. και καλούμε μια συνάρτηση. Εάν το πλήκτρο που διαβάσαμε δεν είναι σε κάποιο case, τότε η *beep()* θα μας προειδοποιήσει για αυτό με ένα ήχο από το speaker

του υπολογιστή. Στο παράδειγμα 2 έχουμε ενεργοποιήσει και τα function keys. Αυτά ξεκινάνε με KEY_ και μια λίστα μπορείτε να βρείτε στο `man curs_getch`.

Ανάμεσά τους συχνότερα χρησιμοποιούμε τα βελάκια και τα πλήκτρα F1 έως F12:

KEY_UP	$\longrightarrow$	πάνω βελάκι
KEY_DOWN	$\longrightarrow$	κάτω βελάκι
KEY_LEFT	$\longrightarrow$	αριστερό βελάκι
KEY_RIGHT	$\longrightarrow$	δεξί βελάκι
KEY_F(n)	$\longrightarrow$	πλήκτρα F*n* όπου για το *n* ισχύει $0 \leq n \leq 63$

Η ncurses μας παρέχει μια συνάρτηση για να ελέγχουμε εάν το τερματικό μας υποστηρίζει κάποιο KEY_ :

has_key() Επιστρέφει TRUE ή FALSE αντίστοιχα, εάν το τερματικό μας αναγνωρίζει ή όχι ένα KEY_ , π.χ. `has_key(KEY_ENTER);`

πρωτότυπο: `int has_key(int ch);`

3.7.2 Ομάδα Β: getstr()

Κατά αναλογία με τις συναρτήσεις εκτύπωσης της Β ομάδας η ncurses μας δίνει τη δυνατότητα να διαβάσουμε strings. Ας δούμε τα πρωτότυπά τους:

```
int getstr(char *str);
int getnstr(char *str, int n);
int wgetstr(WINDOW *win, char *str);
int wgetnstr(WINDOW *win, char *str, int n);
int mvgetstr(int y, int x, char *str);
int mvwgetstr(WINDOW *win, int y, int x, char *str);
int mvgetnstr(int y, int x, char *str, int n);
int mvwgetnstr(WINDOW *, int y, int x, char *str, int n);
```

Το αλφαριθμητικό που διαβάζουμε και τοποθετείται στο *str* δε περιέχει το \n. Συνήθως τις συναρτήσεις αυτές τις χρησιμοποιούμε με ενεργοποιημένο το *echo()* και το *nocbreak()*. Εάν επιπροσθέτως έχουμε ενεργοποιημένο το *keypad()* για αυτό το παράθυρο τότε ο χαρακτήρας backspace και το αριστερό βελάκι έχουν το ίδιο αποτέλεσμα, δηλαδή διαγράφουν τον προηγούμενο χαρακτήρα που πληκτρολογήσαμε. Μια περίπτωση που ενδεχομένως θέλουμε να έχουμε ενεργό το *noecho()* είναι όταν εισάγουμε ένα κωδικό. Σε αυτή τη περίπτωση το backspace πάλι διαγράφει τον προηγούμενο χαρακτήρα που πληκτρολογήσαμε.

Παρατήρηση: Οι συναρτήσεις που έχουνε το *n* σαν όρισμα διαβάζουν το πολύ *n* χαρακτήρες. Οποιαδήποτε προσπάθεια να εισάγουμε περισσότερους χαρακτήρες (εκτός του χαρακτήρα νέας γραμμής) θα προκαλέσει μια ηχητική προειδοποίηση (beep). Το ίδιο ισχύει για τα function keys, προκαλούν beep και αγνοούνται.

3.7.3 Ομάδα Γ: scanw()

Είναι η πιο ενδιαφέρουσα ομάδα από τις συναρτήσεις εισαγωγής. Με αυτές διαβάζουμε μορφοποιημένα αλφαριθμητικά όπως κάναμε με την *scanf()*. Οι επόμενες συναρτήσεις λειτουργούν σαν να είχαμε καλέσει *wgetstr()* για να διαβάσουμε ένα αλφαριθμητικό και μετά την *sscanf()* για να αντιστοιχίσουμε τις τιμές στις μεταβλητές μας:[13]

[13] Λειτουργεί δηλαδή όπως το ζευγάρι *fgets() – sscanf()* για το οποίο έγινε λόγος στην εισαγωγή. Αυτό σημαίνει ότι η κακή συμπεριφορά που είχαμε με το τερματικό χαρακτήρα \n στην *scanf()* εδώ δεν υφίσταται και μπορούμε άφοβα να

scanw() Διάβασμα τιμών από τη τρέχουσα θέση του παράθυρου stdscr, π.χ.
`scanw("%3.2f %c", value, scale);`

πρωτότυπο:

```
int scanw(char *fmt, ...);
```

wscanw() Διάβασμα από ένα παράθυρο, στη τρέχουσα θέση, π.χ.
`wscanw(popup, "%f", miles);`

πρωτότυπο:

```
int wscanw(WINDOW *win, char *fmt, ...);
```

mvscanw() Διαβάζει από τη θέση (y,x) της stdscr, π.χ.
`mvscanw(2, col, "%.6f", moles);`

πρωτότυπο:

```
int mvscanw(int y, int x, char *fmt, ...);
```

mvwscanw() Διαβάζει από τη θέση (y,x) ενός παραθύρου, π.χ.
`mvwscanw(msgwin, row/2, col/2 + 8, "%d", children);`

πρωτότυπο:

```
int mvwscanw(WINDOW *win, int y, int x, char *fmt, ...);
```

vw_scanw() Χρησιμοποιείται όταν έχουμε μεταβλητό πλήθος ορισμάτων· ανάλογη με την *vscanf()*.

πρωτότυπο:

```
int vw_scanw(WINDOW *win, char *fmt, va_list varglist);
```

vwscanw() Το ίδιο με την vw_scanw(). Περισσότερα βλέπε στο `man curs_scanw`.

πρωτότυπο:

```
int vwscanw(WINDOW *win, char *fmt, va_list varglist);
```

3.8 Συναρτήσεις χειρισμού ιδιοτήτων (attributes control routines)

Η δυνατότητα αλλαγής των ιδιοτήτων των χαρακτήρων είναι πολύ σημαντική στα interactive προγράμματα. Για παράδειγμα μπορούμε να δώσουμε έμφαση σε κάποιο κείμενο τυπώνοντας με έντονα γράμματα ή με κάποιο χρώμα. Ο πίνακας που ακολουθεί δείχνει ποιες είναι οι ιδιότητες που μπορούμε να χρησιμοποιήσουμε με τη βιβλιοθήκη curses:

A_NORMAL	Normal display (no highlight)
A_STANDOUT	Best highlighting mode of the terminal
A_UNDERLINE	Underlining
A_REVERSE	Reverse video
A_BLINK	Blinking
A_DIM	Half bright
A_BOLD	Extra bright or bold

καλέσουμε μια *wgetch()* ανά πάσα στιγμή!

A_PROTECT	Protected mode
A_INVIS	Invisible or blank mode
A_ALTCHARSET	Alternate character set
A_CHARTEXT	Bit-mask to extract a character
COLOR_PAIR(n)	Color-pair number n

Αυτές οι ιδιότητες είναι μια *bit-μάσκα* και μπορούμε να τις συνθέσουμε ανά δύο ή ανά τρεις (ή περισσότερες) με τον τελεστή OR (|) για bit:

```
addch( '+' | A_BOLD | COLOR_PAIR(2) );
```

Επιπλέον, ορισμένες ιδιότητες μπορεί να έχουνε το ίδιο αποτέλεσμα για το τερματικό μας ή να μην υποστηρίζονται καθόλου![14] Για παράδειγμα το A_STANDOUT στο τερματικό που χρησιμοποιούμε ενδέχεται να δείχνει το ίδιο φωτεινό ένα γράμμα όσο το A_BOLD. Ή ακόμα μπορεί το A_UNDERLINE να μην τυπώνει υπογραμμισμένο κείμενο! Έτσι, όταν γράφουμε ένα πρόγραμμα και ενδιαφερόμαστε για τη μεταφερσιμότητά του μεταξύ UNIX συστημάτων, πρέπει να λάβουμε υπόψη μας αυτές τις μικρές λεπτομέρειες που επηρεάζουν το οπτικό αποτέλεσμα. Στο `man curs_attr` μπορείτε να διαβάσετε λεπτομέρειες.

Η ιδιότητα COLOR_PAIR(*n*) αναφέρεται σε ένα χρωματικό ζευγάρι *n*, το οποίο ορίζουμε εμείς στην αρχή του προγράμματος. Μέχρι 64 συνδυασμοί επιτρέπονται για το χρώμα του προσκηνίου και του παρασκηνίου. Το ζευγάρι 0 (μηδέν) αντιστοιχεί στο συνδυασμό λευκό-μαύρο και είναι προκαθορισμένο. Όμως περισσότερα για το χρώμα θα μάθουμε στην επόμενη παράγραφο.

Ας δούμε όμως με ποιες συναρτήσεις γίνεται ο χειρισμός αυτών των ιδιοτήτων:

attron()	Ενεργοποιεί μια ιδιότητα ή ένα σύνολο ιδιοτήτων, στο παράθυρο stdscr, ώστε η επόμενη εντολή εκτύπωσης να γίνει με αυτές τις ιδιότητες, π.χ. `attron(A_REVERSE);`
wattron()	Το ίδιο αλλά επιπλέον ορίζουμε ένα παράθυρο, π.χ. `wattron(left, A_NORMAL | A_UNDERLINE);`
attroff()	Απενεργοποιεί μια ιδιότητα ή ένα σύνολο ιδιοτήτων στο παράθυρο stdscr, π.χ. `attroff(A_INVIS);`
wattroff()	Το ίδιο αλλά για το παράθυρο που ορίζουμε, π.χ. `wattroff(right, A_BOLD | COLOR_PAIR(2));`
attrset()	Εφαρμόζει μια ιδιότητα ή ένα σύνολο ιδιοτήτων σε ολόκληρο το παράθυρο stdscr αλλάζοντας με αυτό το τρόπο όλες τις προϋπάρχουσες ιδιότητες, π.χ. `attrset(A_STANDOUT | COLOR_PAIR(1));`
wattrset()	Η ιδιότητα ή οι ιδιότητες αλλάζουν για το παράθυρο που ορίζουμε, π.χ. `wattrset(bottom, A_REVERSE);`

Στα παραπάνω πρέπει να προσέξουμε τη διαφορά της *attron()* με την *attrset()*. Η πρώτη ενεργοποιεί μια ιδιότητα ώστε να χρησιμοποιηθεί στην stdscr από εκείνη τη στιγμή και μετά, ενώ η δεύτερη επιδρά σε ολόκληρη την stdscr, αντικαθιστώντας τις παλιές ιδιότητες με τις καινούριες που δίνουμε σαν παραμέτρους. Έτσι, κάθε στιγμή θα πρέπει να έχουμε μια ιδέα στο μυαλό μας για το πως είναι η οθόνη, διαφορετικά κινδυνεύουμε με μια εντολή να αλλάξουμε ριζικά την εμφάνισή της. Ένας τρόπος για να αποφεύγουμε τέτοιες αβλεψίες είναι να χωρίζουμε την οθόνη σε μικρότερα παράθυρα, για τα οποία θα μιλήσουμε σε επόμενη παράγραφο. Άρα λοιπόν, δεν πρέπει να ξεχνάμε ποτέ τη διπλή φύση που έχει το κείμενο στην οθόνη μας: Είναι ο χαρακτήρας ακολουθούμενος από την ιδιότητά του.

Παράλληλα με τις παραπάνω συναρτήσεις υπάρχουνε και κάποιες άλλες που βοηθάνε στο να

[14] Βλέπε και στην εισαγωγή σχετικά με το `terminfo`.

κάνουμε τον κώδικά μας πιο ευανάγνωστο, δείτε τις αντιστοιχίες και θα καταλάβετε τι εννοώ:

Συνάρτηση	*Κλήση & Αντιστοιχία*
standout()	`standout();` ⇔ `attron(A_STANDOUT);`
wstandout()	`wstandout(ltwin);` ⇔ `wattron(ltwin, A_STANDOUT);`
standend()	`standend();` ⇔ `attrset(A_NORMAL);` ∼ `attrset(0);`
wstandend()	`wstandend(rcwin);` ⇔ `wattrset(rcwin, A_NORMAL);`

Για τις περιπτώσεις όπου έχουμε ξεχάσει ποιες ιδιότητες έχει ένα παράθυρο μπορούμε να τις ανακτήσουμε με τις συναρτήσεις *attr_get()* για την stdscr και *wattr_get()* για ένα άλλο παράθυρο:

attr_get() *πρωτότυπο*:
```
int attr_get(attr_t *attrs, short *pair, void *opts);
```
wattr_get() *πρωτότυπο*:
```
int wattr_get(WINDOW *win, attr_t *attrs, short *pair, void *opts);
```

Δηλαδή θα πρέπει να έχουμε δηλώσει κάπου στο πρόγραμμά μας:

`WINDOW *leftwin;`	Ορίζουμε ένα παράθυρο leftwin.
`attr_t MyAttrs;`	Κρατάει όλες τις ιδιότητες που χαρακτηρίζουνε ένα παράθυρο εκτός από το χρώμα.
`short MyColorPair;`	Ένας μικρός ακέραιος (μεταξύ του 0 και του 63) που κρατάει ένα χρωματικό ζευγάρι.

Και μετά καλούμε:

```
wattr_get(leftwin, MyAttrs, MyColorPair, NULL);
```

Οπότε και στις μεταβλητές MyAttrs, MyColorPair μπαίνουν οι ανάλογες τιμές. Το τελευταίο όρισμα των συναρτήσεων δεν έχει προς το παρόν καμία χρήση. Αυτή η παράμετρος πρόκειται να χρησιμοποιηθεί σε μελλοντικές εκδόσεις της βιβλιοθήκης ncurses, έτσι για την ώρα απλώς πρέπει να γράφουμε *NULL*.

Τελειώνοντας με τις συναρτήσεις χειρισμού ιδιοτήτων θα δούμε την πολύ χρήσιμη οικογένεια συναρτήσεων *chgat()*. Με αυτές τις συναρτήσεις μπορούμε να αλλάξουμε τις ιδιότητες μιας ομάδας χαρακτήρων, χωρίς να αλλάξει η τρέχουσα θέση του δρομέα. Ας δούμε τα πρωτότυπά τους:

chgat()
```
int chgat(int n, attr_t attr, short color, const void *opts);
```
wchgat()
```
int wchgat(WINDOW *win, int n, attr_t attr, short color, const void
*opts);
```
mvchgat()
```
int mvchgat(int y, int x, int n, attr_t attr, short color, const void
*opts);
```
mvwchgat()
```
int mvwchgat(WINDOW *win, int y, int x, int n, attr_t attr, short color,
const void *opts);
```

Θα σχολιάσω την τελευταία από αυτές που είναι και η γενικότερη:

Τα τρία πρώτα ορίσματα είναι τα γνωστά μας: Το παράθυρο, η γραμμή και η στήλη από όπου θα ξεκινήσει η τροποποίηση των ιδιοτήτων. Το τέταρτο όρισμα, n, είναι ο αριθμός των χαρακτήρων των οποίων θα αλλάξει η ιδιότητα. Το δεξί άκρο της γραμμής του παραθύρου αποτελεί φράγμα. Έτσι για τιμές του n που ξεπερνούν το όριο της γραμμής ή για την τιμή -1 (αλλά και για όποια άλλη αρνητική ακέραια) το αποτέλεσμα είναι να αλλάξουν οι ιδιότητες όλων των χαρακτήρων μέχρι το τέλος της γραμμής. Το πέμπτο όρισμα είναι οι ιδιότητες που θέλουμε να εφαρμόσουμε πλην του χρωματικού ζευγαριού που αντιστοιχεί στο έκτο όρισμα. Το τελευταίο όρισμα είναι πάντα *NULL*.

Παρατήρηση: Επειδή με τις παραπάνω συναρτήσεις αλλάζουμε μόνο τις ιδιότητες και όχι το περιεχόμενο, δηλαδή τους χαρακτήρες που απεικονίζονται στην οθόνη μας, καλώντας τη συνάρτηση *refresh()* δε θα πετύχουμε την εφαρμογή τους! Αντ'αυτού θα πρέπει να καλέσουμε τη συνάρτηση *touchwin()* ή κάποια αντίστοιχη από αυτές στο `man curs_touch`.[15] Επομένως αυτό είναι ένα ακόμη σημείο-παγίδα που πρέπει να προσέξουμε στα προγράμματά μας.

Τέλος, ειδικά για το χρώμα, υπάρχει η συνάρτηση *color_set()*[16] που αλλάζει το χρωματικό ζευγάρι για τη stdscr και η *wcolor_set()* για το δοθέν παράθυρο:

color_set() *πρωτότυπο*:
```
int color_set(short color_pair_number, void* opts);
```
wcolor_set() *πρωτότυπο*:
```
int wcolor_set(WINDOW *win, short color_pair_number, void* opts);
```

Το τελευταίο όρισμα είναι πάλι πάντα *NULL*. Για παράδειγμα η επόμενη εντολή αλλάζει στο χρωματικό ζευγάρι που έχουμε αντιστοιχίσει την τιμή 1 τα χρώματα του προσκηνίου/παρασκηνίου: `wcolor_set(left, 1, NULL);`

3.9 Συναρτήσεις χειρισμού χρωμάτων (color manipulation routines)

Όπως έχουμε ήδη πει, με τη βιβλιοθήκη curses μπορούμε να έχουμε χρώμα στην κονσόλα μας, εφόσον το επιτρέπει το τερματικό μας. Για να δούμε εάν το τερματικό μας υποστηρίζει χρώμα χρησιμοποιούμε τη συνάρτηση:

has_colors() *πρωτότυπο*: `bool has_colors(void);`

Εάν η *has_colors()* επιστρέψει TRUE τότε μπορούμε να αρχικοποιήσουμε το χρώμα με τη συνάρτηση αρχικοποίησης:

start_color() *πρωτότυπο*: `int start_color(void);`

Αμέσως επόμενο βήμα είναι να δημιουργήσουμε τα χρωματικά ζευγάρια που θέλουμε να χρησιμοποιήσουμε στο πρόγραμμά μας. Αυτό γίνεται με τη συνάρτηση:

init_pair() *πρωτότυπο*: `int init_pair(short pair, short f, short b);`

Όπου *pair* είναι ένας ακέραιος αριθμός μεταξύ 1 και 63, *f* είναι το χρώμα που θέλουμε για το προσκήνιο (foreground) και *b* το χρώμα για το παρασκήνιο (background). Τα χρώματα που ορίζονται στην επικεφαλίδα `ncurses.h` είναι τα εξής:

#define	COLOR_BLACK	0	*(μαύρο)*
#define	COLOR_RED	1	*(κόκκινο)*
#define	COLOR_GREEN	2	*(πράσινο)*
#define	COLOR_YELLOW	3	*(κίτρινο)*
#define	COLOR_BLUE	4	*(μπλε)*
#define	COLOR_MAGENTA	5	*(ματζέντα, φούξια)*
#define	COLOR_CYAN	6	*(γαλάζιο, κυανό)*
#define	COLOR_WHITE	7	*(λευκό)*

[15] Η προσεκτική ανάγνωση του `man curs_touch` είναι απαραίτητη! Δείτε και τη παράγραφο 3.10.4.

[16] Στην έκδοση 5.3 της βιβλιοθήκης ncurses υπάρχει ένα bug που σχετίζεται με αυτή τη συνάρτηση. Δείτε το σχόλιο που ακολουθεί το παράδειγμα στη σελίδα 44.

Έτσι για παράδειγμα μπορούμε να πάρουμε τους συνδυασμούς:

```
init_pair(1, COLOR_RED, COLOR_BLACK);
init_pair(2, COLOR_GREEN, COLOR_BLACK);
init_pair(3, COLOR_YELLOW, COLOR_BLUE);
init_pair(4, COLOR_WHITE, COLOR_GREEN);
...
```

Το χρωματικό ζευγάρι 0, όπως έχουμε αναφέρει σε προηγούμενη παράγραφο, είναι προκαθορισμένο με (f,b)=(COLOR_WHITE,COLOR_BLACK) και μπορούμε να χρησιμοποιήσουμε τη συνάρτηση *pair_content()* για να το διαπιστώσουμε:

pair_content() *πρωτότυπο*: `int pair_content(short pair, short *f, short *b);`

Δηλαδή έχοντας δηλώσει:

```
short foreground, background;
```

Καλούμε:

```
pair_content(0, &foreground, &background);
```

Και οι μεταβλητές παίρνουνε τις τιμές:

```
foreground=COLOR_WHITE, background=COLOR_BLACK
```

3.10 Συναρτήσεις χειρισμού παραθύρων (curses windows)

3.10.1 Εισαγωγή

Τα παράθυρα είναι μια από τις βασικές δυνατότητες της βιβλιοθήκης curses. Το σημαντικότερο πλεονέκτημα είναι ότι μας βοηθούν να διαμορφώσουμε ένα φιλικό περιβάλλον για το χρήστη, χωρίζοντας την οθόνη σε τμήματα. Αυτή η διαίρεση της οθόνης, παράλληλα με την καλή οπτική αίσθηση, δίνει προγραμματιστικά το πλεονέκτημα του καλύτερου ελέγχου στις αλλαγές που θέλουμε να κάνουμε στην εμφάνισή της. Ήδη είδαμε μια πληθώρα συναρτήσεων που παίρνουν σαν παράμετρο ένα παράθυρο (μια δομή WINDOW *). Μπορείτε να φανταστείτε τι βάσανο θα ήταν να αρκούμασταν στην πρότυπη οθόνη, stdscr, για να χειριστούμε όλες τις αλλαγές που γίνονται σε αυτήν; Αντ'αυτού λοιπόν διαιρούμε σε μικρότερα κομμάτια την οθόνη μας και τα χειριζόμαστε καθένα ξεχωριστά, αντιστοιχίζοντάς τους από μια χρήση, π.χ. ένα παράθυρο για το μενού, ένα παράθυρο για την εκτύπωση, ένα παράθυρο για τα αναδυόμενα (pop-up) μηνύματα, κ.τ.λ.

3.10.2 Παράθυρα και υποπαράθυρα

Με την ncurses μπορούμε να δημιουργούμε παράθυρα που καταλαμβάνουν είτε όλη την οθόνη είτε μέρος αυτής, να τα διαγράφουμε, να τα μετακινούμε και να τα αντιγράφουμε. Εκτός από τα παράθυρα δίνεται η δυνατότητα να δημιουργήσουμε υποπαράθυρα μέσα σε ένα παράθυρο.

Ας δούμε λοιπόν με ποιες συναρτήσεις μπορούμε να χειριστούμε τα παράθυρα:

newwin() Καλούμε αυτή τη συνάρτηση για να οριοθετήσουμε ένα νέο παράθυρο. Δεσμεύει τη μνήμη που απαιτείται για μια δομή WINDOW και αρχικοποιεί τη δομή αυτή στις διαστάσεις που θέλουμε να έχει το παράθυρο, καθώς επίσης και τη θέση του στην οθόνη:

πρωτότυπο:

```
WINDOW *newwin(int nlines, int ncols, int begin_y, int
begin_x);
```

Όπου *nlines, ncols*, ο αριθμός των γραμμών και των στηλών αντίστοιχα και όπου *begin_y, begin_x* η θέση του πρώτου χαρακτήρα του παραθύρου (πάνω αριστερή γωνία).
Για παράδειγμα έστω:

```
WINDOW *title;
title=newwin(3,30,3,COLS/2-15);
```

Το παράθυρο title αποτελείται από τρεις γραμμές, τριάντα στήλες και ξεκινάει από τη θέση (y, x)=(3, COLS/2-15), δηλαδή εάν COLS=80, τότε (y, x)=(3, 25) και όπως καταλαβαίνετε είναι κεντραρισμένο στην οθόνη μας. Στη περίπτωση που θέσουμε για *nlines* ή *ncols* την τιμή μηδέν, τότε οι παράμετροι αρχικοποιούνται στις τιμές LINES-*begin_y* και COLS-*begin_x* αντίστοιχα και είναι πολύ βολικό άμα το καλοσκευτείτε…Κατά συνέπεια η κλήση `newwin(0,0,0,0)` φτιάχνει ένα παράθυρο που καταλαμβάνει όλη την οθόνη.

Σημείωση: Μετά τη κλήση της συνάρτησης δεν παρατηρούμε καμία αλλαγή στην οθόνη. Για να «ζωγραφίσουμε» στο νέο μας παράθυρο πρέπει να δώσουμε τις επιθυμητές εντολές ορισμού των ιδιοτήτων π.χ. *attrset()* και μετά να τυπώσουμε κάτι με *wprintw()* και να το εμφανίσουμε με την *wrefresh()*, όπως είδαμε στις προηγούμενες παραγράφους.

delwin() Ελευθερώνει τη μνήμη που έχει δεσμευτεί για το παράθυρο που δέχεται σαν όρισμα. Εφόσον το παράθυρο περιέχει υποπαράθυρα θα πρέπει πρώτα να διαγράψουμε αυτά, προτού επιχειρήσουμε τη διαγραφή του πατρικού παραθύρου, διαφορετικά θα έχουμε απροσδιόριστα αποτελέσματα.

πρωτότυπο:

```
int delwin(WINDOW *win);
```

Προσοχή: Τα δεδομένα του παραθύρου μπορεί να διαγράφονται οριστικά από τη μνήμη, ωστόσο αυτά παραμένουν στην οθόνη. Για να τα σβήσουμε και από εκεί θα πρέπει μόνοι μας να γεμίσουμε με κενά όλη την περιοχή που καταλάμβανε το παράθυρο. Μάλιστα, ένα τρικ είναι το εξής: Προτού καλέσουμε την *delwin()* καλούμε την *werase()* η οποία κάνει ακριβώς αυτό που θέλουμε, δηλαδή γεμίζει όλες τις θέσεις του παραθύρου με κενά. Συμβουλευτείτε και το `man curs_clear`.

mvwin() Μετακινεί το παράθυρο έτσι ώστε η πάνω αριστερή γωνία να βρίσκεται στη θέση (y,x). Τα φυσικά όρια της οθόνης αποτελούν φράγματα και δε μπορούμε να μετακινήσουμε ένα παράθυρο πέρα από αυτά. Επίσης η μετακίνηση υποπαραθύρων επιτρέπεται, ωστόσο το `man curs_window` μας συμβουλεύει να αποφεύγουμε κάτι τέτοιο.

πρωτότυπο:

```
int mvwin(WINDOW *win, int y, int x);
```

dupwin() Αυτή η συνάρτηση δημιουργεί ένα ακριβές αντίγραφο του παραθύρου που δέχεται σαν όρισμα.

πρωτότυπο:

```
WINDOW *dupwin(WINDOW *win);
```

Σε αυτό το σημείο είμαι υποχρεωμένος να τονίσω κάτι το προφανές: Αφού τα παράθυρα αποτελούν θεμελιώδη δομή για τη βιβλιοθήκη curses προτού αρχίσουμε να χρησιμοποιούμε ένα καινούριο παράθυρο θα πρέπει να είμαστε σίγουροι ότι αυτό υπάρχει. Ένα λάθος σε μία μόνο παράμετρο της *newwin()* που προσπαθεί να δημιουργήσει ένα παράθυρο έξω από τα όρια της οθόνης, μπορεί να οδηγήσει σε κατάρρευση του προγράμματος αμέσως μόλις η ροή του ελέγχου συναντήσει μια αναφορά στο ανύπαρκτο παράθυρο που νομίσαμε ότι φτιάξαμε Για την αντιμετώπιση του κινδύνου αρκεί μια εντολή ελέγχου της μορφής *if (NULL==mywin) { ...}* .

Μέσα σε κάθε παράθυρο μπορούμε να φτιάξουμε υποπαράθυρα που όμως, αναγκαστικά, θα πρέπει να περιέχονται στα όρια του πατρικού παραθύρου. Ένα υποπαράθυρο επιτρέπεται να έχει και το ίδιο υποπαράθυρα, αρκεί να μην ξεπερνούν τις διαστάσεις του υποπαραθύρου μέσα στο οποίο βρίσκονται. Έτσι πρέπει να πούμε ότι *σχεδόν επιβάλλεται* να ελέγχουμε εάν το παράθυρό μας δημιουργήθηκε επιτυχώς πριν αρχίσουμε να το χρησιμοποιούμε.

subwin() Με τη συνάρτηση αυτή δημιουργούμε υποπαράθυρα με συντεταγμένες αναφοράς για την πάνω αριστερή γωνία του παραθύρου τις διαστάσεις της οθόνης:

πρωτότυπο:

```
WINDOW *subwin(WINDOW *orig, int nlines, int ncols, int
begin_y, int begin_x);
```

Για παράδειγμα έστω η κλήση:
`small=subwin(big, 5, 20, LINES-5, COLS-20);`
Φτιάχνει το υποπαράθυρο small που περιέχεται στο big με διαστάσεις 5 γραμμές επί 20 στήλες και βρίσκεται στο κάτω δεξιό άκρο της οθόνης. Σύμφωνα με τα όσα είπαμε για να είναι δυνατή η δημιουργία του υποπαραθύρου η περιοχή αυτή θα πρέπει να αποτελεί τμήμα του παραθύρου big.

derwin() Κάνει ακριβώς το ίδιο με τη συνάρτηση *subwin()* μόνο που τώρα η θέση του παραθύρου προσδιορίζεται ως προς τη θέση του πατρικού παραθύρου (την πάνω αριστερή γωνία του).

πρωτότυπο:

```
WINDOW *derwin(WINDOW *orig, int nlines, int ncols, int
begin_y, int begin_x);
```

Έτσι, έστω το παράδειγμα:

```
WINDOW *textcell;
textcell=subwin(textform, 1, 40, 3, 12);
```

Με αυτή την κλήση δημιουργούμε το υποπαράθυρο textcell που ξεκινάει από τη θέση (3,12) του παραθύρου textform (μετρώντας από τη πάνω αριστερή του γωνία) και έχει διαστάσεις μια γραμμή και σαράντα στήλες. Για μια τελευταία φορά τονίζω πως το υποπαράθυρο δε θα δημιουργηθεί εάν ξεπερνάει έστω και για ένα χαρακτήρα τα όρια του πατρικού του παραθύρου.

mvderwin() Μετακινεί το υποπαράθυρο (με οποιαδήποτε από τις δύο προηγούμενες συναρτήσεις και εάν δημιουργήθηκε) μέσα στα όρια του πατρικού παραθύρου. Οι συντεταγμένες του ως προς την οθόνη παρόλα αυτά δεν αλλάζουν. Την χρησιμοποιούμε για να επιτρέπουμε την εμφάνιση των χαρακτήρων που καλύπτονται από το υποπαράθυρο και ανήκουν στο πατρικό παράθυρο.

πρωτότυπο:

```
int mvderwin(WINDOW *win, int par_y, int par_x);
```

Με τα τόσα πολλά παράθυρα που δημιουργούμε μια εύλογη απορία είναι το τι γίνεται με τη μνήμη του υπολογιστή. Μπορεί στους σύγχρονους υπολογιστές να είναι υπεραρκετή όμως η βιβλιοθήκη χρησιμοποιούνταν και εξακολουθεί να χρησιμοποιείται και να είναι αποδοτική και σε συστήματα με περιορισμένη μνήμη. Όπως έχουμε πει κάθε παράθυρο αποτελείται από μια δομή WINDOW που κρατάει τις πληροφορίες για αυτό. Ωστόσο δεν επεκταθήκαμε σε λεπτομέρειες Τι ακριβώς συμβαίνει λοιπόν;

Σε κάθε παράθυρο της βιβλιοθήκης curses αντιστοιχούν δύο δομές δεδομένων. Η μία διαμοιράζεται ανάμεσα σε όλα τα παράθυρα και υποπαράθυρα που μπορούν να συνυπάρξουν στην οθόνη και ουσιαστικά αντιπροσωπεύει αυτό που βλέπουμε στην οθόνη. Η άλλη είναι αποκλειστική σε κάθε ένα παράθυρο ή υποπαράθυρο και περιλαμβάνει μόνο τις αλλαγές που συμβαίνουν πάνω στο παράθυρο ή υποπαράθυρο αυτό. Επομένως με αυτό το τρόπο επιτυγχάνεται περιορισμός της απαιτούμενης μνήμης.

Οπότε μια συνάρτηση *wrefresh(apple)* ενεργεί κάνοντας χρήση της δομής που είναι αποκλειστική για το παράθυρο *apple*. Έτσι οι αλλαγές δε μπορούν να γίνουν γνωστές στα υπόλοιπα παράθυρα της ιεραρχίας δηλαδή αυτά από τα οποία προέρχεται ή στα υποπαράθυρά του κάτι που έχει ως συνέπεια αυτό που θα δούμε στην οθόνη να μην είναι απαραίτητα και το επιθυμητό. Για να διορθώσουμε αυτή την κατάσταση χρησιμοποιούμε τις ρουτίνες *wsyncup(), wsyncdown(), syncok()* και *wcursyncup()* οι οποίες αναλαμβάνουν να ενημερώσουν όλα τα παράθυρα της ιεραρχίας για τις αλλαγές που υφίσταται ένα μέλος της. Μια σύντομη περιγραφή τους θα βρείτε στο `man curs_window`.

3.10.3 Όρια των παραθύρων

Μέχρι τώρα αντιμετωπίζαμε ένα παράθυρο σαν μια οριοθετημένη περιοχή στην οθόνη που όμως δεν κάναμε τίποτα για να οπτικοποιήσουμε τις διαστάσεις του με κάποιο τρόπο. Σε αυτή τη παράγραφο θα δούμε πώς η βιβλιοθήκη curses μας δίνει τη δυνατότητα να τυπώσουμε οποιοδήποτε χαρακτήρα κατά μήκος της περιμέτρου των παραθύρων που δημιουργούμε. Και μάλιστα όχι απλώς ένα χαρακτήρα, κάτι περισσότερο: Ένα χαρακτήρα με ιδιότητες!

Για το περίγραμμα του παραθύρου η βιβλιοθήκη χρησιμοποιεί κάποιους εξ'ορισμού χαρακτήρες που έχει αναθέσει σε κάποιες συμβολικές σταθερές και ξεκινούν από τα γράμματα ACS. Οι χαρακτήρες αυτοί λέγονται *'box and line drawing characters'*[17] και ανήκουν στον OEM Extended ASCII Code [10].

[17] Για όσους γνωρίζουν HTML είναι τα entities &box[h,v,u,d][,r,l,u,d]; και &box[H,V,U,D][,R,L,U,D]; καθώς και οι συνδυασμοί πεζών με κεφαλαίων hU, hV, Hu, Hv, κ.τ.λ. .

Εάν δώσουμε στο τερματικό μας την εντολή:

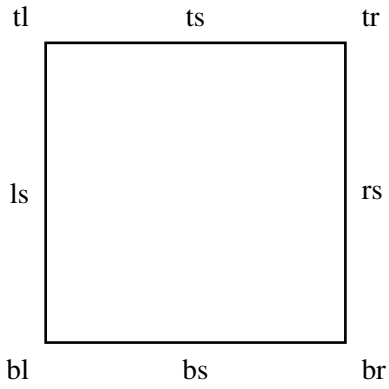
(george@earth) ~$ **grep ACS /usr/include/ncurses.h**

θα δούμε αυτές τις συμβολικές σταθερές. Όμως για να μπορέσουμε να εμφανίσουμε αυτούς τους χαρακτήρες στο τερματικό μας θα πρέπει να χρησιμοποιήσουμε μια γραμματοσειρά που να περιλαμβάνει αυτό το σύνολο συμβόλων. Επιπλέον εμείς θα προτιμούσαμε να έχουμε μια ελληνική γραμματοσειρά. Μετά από αναζήτηση στις γραμματοσειρές του πακέτου kbd[18] για τη κονσόλα του Linux βρήκα τη γραμματοσειρά iso07u-16.psfu που ικανοποιεί αυτή τη διττή απαίτησή μας. Δίνουμε λοιπόν την εντολή:

(george@earth) ~$ **setfont iso07u-16.psfu**

Και στη συνέχεια εκτελούμε τα προγράμματά μας. Εάν τώρα στο σύστημά σας δεν υπάρχει το ελληνικό locale αναζητήστε μια γραμματοσειρά που να δείχνει σωστά τους OEM Extended ASCII χαρακτήρες δίνοντας αντίστοιχες εντολές ή κάντε τις απαραίτητες ρυθμίσεις πειράζοντας τα locales του συστήματός σας. (Οι γραμματοσειρές σε μια τυπική εγκατάσταση πρέπει να βρίσκονται στον κατάλογο `/usr/lib/kbd/consolefonts` ή κάτι αντίστοιχο). Εάν το πακέτο kbd στο σύστημά σας είναι παλιό, μπορείτε να εγκαταστήσετε την τελευταία έκδοση στο λογαριασμό σας, κάτι που ισχύει και για τη βιβλιοθήκη ncurses! Δικαιώματα root απαιτούνται μόνο όταν πρόκειται να εγκατασταθούν σε κατάλογο του συστήματος απ'όπου όλοι οι χρήστες έχουν πρόσβαση (`/usr`, `/usr/local`, `/opt`).

Ακολουθεί ένα πινακάκι και ένα σχήμα που θα βοηθήσει να καταλάβουμε την χρήση των συναρτήσεων στη συνέχεια:



παράμετρος	θέση	συμβολική σταθερά
tl	πάνω αριστερά	ACS_ULCORNER
tr	πάνω δεξιά	ACS_URCORNER
bl	κάτω αριστερά	ACS_LLCORNER
br	κάτω δεξιά	ACS_LRCORNER
ts	πάνω πλευρά	ACS_HLINE
bs	κάτω πλευρά	ACS_HLINE
ls	αριστερή πλευρά	ACS_VLINE
rs	δεξιά πλευρά	ACS_VLINE

[18] Keyboard and console utilities for Linux, Version 1.08. Βρείτε την τελευταία έκδοση στη διεύθυνση: `ftp://ftp.win.tue.nl/pub/home/aeb/linux-local/utils/kbd`

box() Η πιο απλή συνάρτηση για την κατασκευή περιγράμματος. Μπορούμε να ζητήσουμε να τυπώσει ένα χαρακτήρα για την αριστερή και δεξιά μεριά (πρώτη και τελευταία στήλη) και έναν άλλον ή τον ίδιο χαρακτήρα για την πάνω και κάτω πλευρά (πρώτη και τελευταία γραμμή). Οι χαρακτήρες που βρίσκονται στις τέσσερις γωνίες παίρνουνε την εξ'ορισμού τιμή που φαίνεται στο πινακάκι (ACS_??CORNER).

πρωτότυπο:

```
int box(WINDOW *win, chtype verch, chtype horch);
```

Η δεύτερη παράμετρος είναι ο χαρακτήρας για τις κάθετες πλευρές και η τρίτη για τις οριζόντιες πλευρές. Εάν δώσουμε την εντολή:

```
box(mywin, 0, 0);
```

θα πάρουμε το εξ'ορισμού περίγραμμα που περιγράφει το πινακάκι, και οπτικά το αποτέλεσμα είναι μια μονή συνεχής γραμμή.

Παραδείγματα:

```
box(title, '*'|A_DIM, '*'|A_BOLD );
box(plan, 0, '@'|COLOR_PAIR(2) );
box(result, 'R'|A_REVERSE, ' ' );
```

border() Φτιάχνει το περίγραμμα του παραθύρου stdscr επιτρέποντάς μας να ορίσουμε ένα χαρακτήρα με ιδιότητες όχι μόνο για τις πλευρές αλλά και για τις γωνίες.

πρωτότυπο:

```
int border(chtype ls, chtype rs, chtype ts, chtype bs,
chtype tl, chtype tr, chtype bl, chtype br);
```

Παραδείγματα:

```
border( '(', ')', '~', '~', '#', '%', '&', '@' );
border( 0, 0, 0, 0, 'i'|COLOR_PAIR(2)|A_BOLD, 'X'|A_DIM, 0, 0 );
```

wborder() Η γενική μορφή της προηγούμενης συνάρτησης για ένα παράθυρο.

πρωτότυπο:

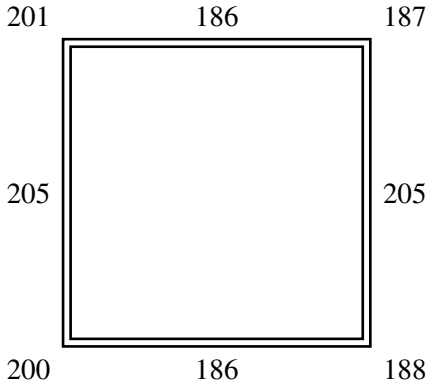
```
int wborder(WINDOW *win, chtype ls, chtype rs, chtype ts,
chtype bs, chtype tl, chtype tr, chtype bl, chtype br);
```

Παράδειγμα:

```
wborder(mainwin, 186|A_ALTCHARSET, 186|A_ALTCHARSET,
205|A_ALTCHARSET, 205|A_ALTCHARSET, 201|A_ALTCHARSET,
187|A_ALTCHARSET, 200|A_ALTCHARSET, 188|A_ALTCHARSET);
```

Σε αυτό το παράδειγμα κάνω χρήση του OEM Extended ASCII για να κατασκευάσω περίγραμμα με διπλές γραμμές. Εάν παραλείπαμε την ιδιότητα A_ALTCHARSET θα τυπώνονταν οι αντίστοιχοι χαρακτήρες ANSI (βλέπε και [10]).

Οπτικά στην οθόνη θα πρέπει να δείτε κάτι σαν αυτό:



Ολοκληρώνοντας και αυτή τη παράγραφο, πρέπει να κάνω μια επισήμανση. Αφού οι περιμετρικές θέσεις του παραθύρου καλύπτονται με κάποιον χαρακτήρα, θα πρέπει να αποφύγουμε να τυπώσουμε κάτι επάνω σε αυτές γιατί θα καταστρέψουμε το περίγραμμα! Έτσι η πρώτη εντολή εκτύπωσης θα πρέπει να ξεκινάει από τη δεύτερη γραμμή και δεύτερη στήλη ενώ επίσης δε θα πρέπει να ξεπερνάει την προτελευταία γραμμή και στήλη. Ένας σίγουρος τρόπος για να εξασφαλίσουμε κάτι τέτοιο είναι να φτιάξουμε ένα υποπαράθυρο μέσα στο παράθυρο με το περίγραμμα και να τυπώνουμε μέσα σε αυτό. Δηλαδή ένα υποπαράθυρο κατά δύο διαστάσεις μικρότερο από το παράθυρο με το περίγραμμα που να βρίσκεται στο κέντρο του!

3.10.4 Όχι άλλο refresh!

Οι συνεχείς κλήσεις των εντολών *wrefresh()* για ανανέωση των περιεχομένων ενός παραθύρου δεν είναι η καλύτερη δυνατή μέθοδος. Έχει μερικά μειονεκτήματα: α) καθυστερεί την εκτέλεση του προγράμματος (περισσότερες συγκρίσεις, περισσότερα δεδομένα περνούν από τη μνήμη στην οθόνη), β) δεν είναι σε όλες τις περιπτώσεις αποτελεσματική (π.χ. παράθυρα που μοιράζονται τον ίδιο φυσικό χώρο της οθόνης), και γ) μπορεί να γίνει δύσκολη η παρακολούθηση των συνεχόμενων αλλαγών στην οθόνη· οπτικά θα έχουμε ένα αποτέλεσμα που μοιάζει με αυτό της *flash()*[19].

Συναρτήσεις:	*Πρωτότυπα:*
refresh()	`int refresh(void);`
wrefresh()	`int wrefresh(WINDOW *win);`
wnoutrefresh()	`int wnoutrefresh(WINDOW *win);`
doupdate()	`int doupdate(void);`

Κάθε κλήση της εντολής *wrefresh()* δουλεύει ως εξής: Καλεί πρώτα τη *wnoutrefresh()* που ενημερώνει την εικονική οθόνη stdscr για τις αλλαγές που πρέπει να εφαρμοστούν και κατόπιν καλεί τη *doupdate()* η οποία συγκρίνει την εικονική οθόνη με τη φυσική οθόνη και εφαρμόζει αυτές τις αλλαγές πάνω στη curscr. Επομένως αντί να καλούμε για κάθε ένα παράθυρο ξεχωριστά τη *wrefresh()* και να γίνεται πολλές φορές όλη αυτή η διαδικασία, είναι προτιμότερο να καλούμε τη *wnoutrefresh()*, για κάθε παράθυρο που έχει αλλάξει το περιεχόμενό του, και στο τέλος μια μόνο φορά τη *doupdate()*.

Όταν το περιεχόμενο του παραθύρου ή μιας γραμμής έχει υποστεί μη επιθυμητές αλλαγές τότε μπορούμε να πούμε στην ncurses να το ξαναζωγραφίσει με τις ρουτίνες *redrawwin()* και *wredrawln()* αντίστοιχα:

[19] *Πρωτότυπο:* `int flash(void);`, `man curs_beep`.

Συναρτήσεις:	*Πρωτότυπα:*
redrawwin()	`int redrawwin(WINDOW *win);`
wredrawln()	`int wredrawln(WINDOW *win, int beg_line, int num_lines);`

Σε αυτή τη περίπτωση είναι πολύ καλύτερο αντί της *redrawwin()* να καλέσουμε την *wredrawln()* από την πρώτη γραμμή, *beg_line*, του παραθύρου μέχρι *num_lines* το πλήθος γραμμές, που φτάνουν ως τη τελευταία του γραμμή.

Εάν οι αλλαγές στο παράθυρο σχετίζονται με αλλαγή στις ιδιότητες ή στη περίπτωση που έχουμε επικαλυπτόμενα παράθυρα, τότε η *wrefresh()* δεν αρκεί. Αντ'αυτής πρέπει να καλέσουμε την *touchwin()*.

Συναρτήσεις:	*Πρωτότυπα:*
touchwin()	`int touchwin(WINDOW *win);`
touchline()	`int touchline(WINDOW *win, int start, int count);`
wtouchln()	`int wtouchln(WINDOW *win, int y, int n, int changed);`

Η *touchwin()* συμπεριφέρεται σαν να είχε μόλις δημιουργηθεί ολόκληρο το παράθυρο. Οποιαδήποτε πληροφορία υπάρχει για μέρη του παραθύρου που έχουνε πειραχτεί από άλλα επικαλυπτόμενα παράθυρα αγνοείται. Το ίδιο ισχύει και για τις άλλες δυο συναρτήσεις. Η *wtouchln()* μαρκάρει σαν αλλαγμένες (changed=1) ή μη (changed=0), έπειτα από την τελευταία κλήση της *wrefresh()*, *n* γραμμές ξεκινώντας από τη γραμμή *y*.

Στο `man curs_touch` θα βρείτε την περιγραφή των ρουτινών *is_linetouched()* και *is_wintouched()* που μας βοηθάνε να μάθουμε εάν το παράθυρο έχει «πειραχτεί» από την τελευταία κλήση της *wrefresh()* και μετά.

3.10.5 Σχόλιο: Ορθή χρήση

Όλα αυτά τα παράθυρα και υποπαράθυρα μπορεί να σας κάνουν να σκέφτεσθε πως τα πράγματα δεν είναι τόσο απλά, αντίθετα γίνονται ολοένα και πιο πολύπλοκα. Πιθανόν να προτιμήσετε για αυτό το λόγο να δουλεύετε στην απλότητα της πρότυπης οθόνης stdscr. Αυτό όμως είναι η μισή αλήθεια. Τα παράθυρα που φτιάχνουμε σπάνια τα χρησιμοποιούμε μια και μοναδική φορά κατά τη διάρκεια της εκτέλεσης ενός προγράμματος. Σκεφτείτε για παράδειγμα ένα παράθυρο που αναδύεται στο κέντρο της οθόνης και λέει στο χρήστη ένα μήνυμα λάθους ή μια πληροφορία και του ζητάει να πατήσει (Y)ES, (N)O, (C)CANCEL. Πολλά διαφορετικά μηνύματα μπορείτε να τυπώσετε στο ίδιο παράθυρο ενώ σας χρειάζονται δύο μόνο συναρτήσεις. Μια να δημιουργεί το παράθυρο και μια να το καταστρέφει. Είπα όμως ότι είναι η μισή αλήθεια. Σωστά. Γιατί εάν τώρα έχετε ένα μικρό πρόγραμμα που δεν χρειάζεται όλες αυτές τις πολυτέλειες για την αλληλεπίδραση του χρήστη, ίσως σας αρκεί μια απλή ακολουθία συναρτήσεων *printw(), refresh(), scanw()* για να κάνετε την δουλειά σας. Επομένως σε κάθε περίπτωση οφείλετε να εκτιμήσετε τις απαιτήσεις της εφαρμογής που προγραμματίζετε και να πράξετε δεόντως.

3.11 Συναρτήσεις χειρισμού ποντικιού (mouse interface)

3.11.1 Περιορισμοί

Όπως ήδη έχουμε αναφέρει η βιβλιοθήκη ncurses μας δίνει τη δυνατότητα να κάνουμε χρήση του ποντικιού του υπολογιστή μέσα στα προγράμματά μας. Υπάρχουνε όμως δύο βασικοί περιορισμοί. Πρώτον, αυτό μπορεί να γίνει μόνο μέσα στο xterm σε παραθυρικό περιβάλλον ή στη κονσόλα του Linux, εφόσον τρέχουμε τον gpm server[20] για υποστήριξη του ποντικιού. Και δεύτερον, η δυνατότητα υποστήριξης του ποντικιού είναι αποκλειστική λειτουργία στις ncurses. Με άλλα λόγια η μεταφερσιμότητα ενός προγράμματος που χρησιμοποιεί το ποντίκι δεν είναι δυνατή σε άλλα συστήματα διότι δεν υποστηρίζεται από τις δικές τους curses, δηλαδή του BSD ή του System V R4 ή σε σύστημα με το XSI Curses Standard[21].

Για να κάνουμε λοιπόν τα προγράμματά μας μεταφέρσιμα σε άλλα συστήματα θα πρέπει να χρησιμοποιούμε τον προεπεξεργαστή αφαιρώντας όσο τμήμα του κώδικα αφορά το ποντίκι, ως εξής:

```
#ifdef NCURSES_MOUSE_VERSION
       [mouse interface related code]
#endif
```

Η συμβολική σταθερά NCURSES_MOUSE_VERSION προς το παρόν έχει την τιμή 1. Όταν στο μέλλον προστεθεί υποστήριξη και για άλλες συσκευές κατάδειξης (touchscreen, joystick), ο αριθμός αυτός θα αυξηθεί.

Κατά συνέπεια λογικό είναι, να μην θέτουμε ως απαραίτητη προϋπόθεση την ύπαρξη του ποντικιού για την εκτέλεση ενός προγράμματος. Αντίθετα μπορούμε κάλλιστα να το χρησιμοποιούμε σαν εναλλακτική λύση για κάποια πληκτρολόγηση που ζητά το πρόγραμμα, π.χ. να καταδείξουμε και να πατήσουμε σε μια επιλογή (Y)ES, (N)O, (C)ANCEL, αντί των αντίστοιχων γραμμάτων στο πληκτρολόγιο.

3.11.2 Αρχικοποίηση συμβάντων ποντικιού

Πριν αρχίσουμε να λαμβάνουμε τα σήματα με τα συμβάντα του ποντικιού (αριστερό ή δεξί πλήκτρο πατημένο, κ.τ.λ.) πρέπει να ενεργοποιήσουμε τα συμβάντα που επιθυμούμε να χρησιμοποιήσουμε στο πρόγραμμα. Ακριβώς όπως κάναμε και με τα χρωματικά ζευγάρια. Εκεί είδαμε ότι το ζευγάρι που αντιστοιχεί στο COLOR_PAIR(0) (άσπρα γράμματα, μαύρο φόντο) ήταν προκαθορισμένο. Αντίθετα εδώ όλα τα συμβάντα είναι ανενεργά και γι' αυτό πρέπει να καλέσουμε τη συνάρτηση *mousemask()* με τις κατάλληλες παραμέτρους:

mousemask() *πρωτότυπο*:
`mmask_t mousemask(mmask_t newmask, mmask_t *oldmask);`

Η πρώτη παράμετρος είναι η bit mask που θέλουμε να ενεργοποιήσουμε, ενώ σαν δεύτερη παράμετρο μπορούμε να δώσουμε την τιμή NULL. Πληροφοριακά να αναφέρω ότι το *mmask_t* είναι δηλωμένο σαν `typedef unsigned long`. Η συνάρτηση θα μας επιστρέψει μια bit mask που θα αντιστοιχεί σε όσα συμβάντα ενεργοποιήθηκαν επιτυχώς, ενώ θα επιστρέψει την τιμή μηδέν σε περίπτωση ολικής αποτυχίας. Για παράδειγμα εάν θέλουμε να διαβάζουμε αριστερό και δεξί κλικ δίνουμε:

```
mousemask(BUTTON1_CLICKED|BUTTON3_CLICKED, NULL);
```

[20] General Purpose Mouse, πληροφορίες στη διεύθυνση: `http://linux.schottelius.org/gpm`

[21] *X/Open Curses, Issue 4, Version 2*, διαβάστε το δωρεάν σε μορφή PDF από το δικτυακό τόπο:
`http://www.opengroup.org/products/publications/catalog/c610.htm`

Ή για ενεργοποιήσουμε όλες τις δυνατές περιπτώσεις συμβάντων του ποντικιού, θα πρέπει να γράψουμε:

```
mousemask(ALL_MOUSE_EVENTS, NULL);
```

Τη συνάρτηση *mousemask()* μπορούμε να τη χρησιμοποιούμε και στη συνέχεια του κώδικά μας, αλλάζοντας κάθε φορά τη bit mask που θέλουμε να είναι ενεργή. Η πλήρης λίστα με όλες τις δυνατές τιμές για τη bit mask είναι αυτή που ακολουθεί όμως πρέπει να έχετε υπόψη πως δεν είναι απαραίτητα όλες αξιοποιήσιμες με τη συσκευή που έχετε.

bit mask	Περιγραφή
BUTTON1_PRESSED	mouse button 1 down
BUTTON1_RELEASED	mouse button 1 up
BUTTON1_CLICKED	mouse button 1 clicked
BUTTON1_DOUBLE_CLICKED	mouse button 1 double clicked
BUTTON1_TRIPLE_CLICKED	mouse button 1 triple clicked
BUTTON2_PRESSED	mouse button 2 down
BUTTON2_RELEASED	mouse button 2 up
BUTTON2_CLICKED	mouse button 2 clicked
BUTTON2_DOUBLE_CLICKED	mouse button 2 double clicked
BUTTON2_TRIPLE_CLICKED	mouse button 2 triple clicked
BUTTON3_PRESSED	mouse button 3 down
BUTTON3_RELEASED	mouse button 3 up
BUTTON3_CLICKED	mouse button 3 clicked
BUTTON3_DOUBLE_CLICKED	mouse button 3 double clicked
BUTTON3_TRIPLE_CLICKED	mouse button 3 triple clicked
BUTTON4_PRESSED	mouse button 4 down
BUTTON4_RELEASED	mouse button 4 up
BUTTON4_CLICKED	mouse button 4 clicked
BUTTON4_DOUBLE_CLICKED	mouse button 4 double clicked
BUTTON4_TRIPLE_CLICKED	mouse button 4 triple clicked
BUTTON_SHIFT	shift was down during button state change
BUTTON_CTRL	control was down during button state change
BUTTON_ALT	alt was down during button state change
ALL_MOUSE_EVENTS	report all button state changes
REPORT_MOUSE_POSITION	report mouse movement

3.11.3 Χειρισμός συμβάντων ποντικιού

Ο χειρισμός των συμβάντων γίνεται αρχικά με το γνωστό απλό τρόπο της παραγράφου 3.7.1, δηλαδή μέσω των συναρτήσεων της ομάδας *getch()* και του function key KEY_MOUSE. Στη συνέχεια γίνεται ο έλεγχος για να βρούμε ποιο ακριβώς bit mask συνέβει. Για τη δική μας ευκολία, η βιβλιοθήκη ncurses ορίζει μια ακόμη δομή με όνομα MEVENT και μας παρέχει τη συνάρτηση *getmouse()* για να τη δουλέψουμε.

Η δομή έχει ως εξής:

```
typedef struct
    {
      short id;      /* ID to distinguish multiple devices */
      int x, y, z;    /* event coordinates */
      mmask_t bstate;  /* button state bits */
    }
    MEVENT;
```

Το *id* είναι για να γίνεται διαχωρισμός μεταξύ διαφορετικών συσκευών κατάδειξης όμως προς το παρόν δε χρησιμοποιείται. Επίσης δε χρησιμοποιείται και η συντεταγμένη *z*, η οποία θα μπορούσε για παράδειγμα σε μια μελλοντική χρήση να μετρά τη διάρκεια της πίεσης που ασκείται σε μια συσκευή touchscreen. Οι συντεταγμένες *x* και *y* αντιστοιχούν στη θέση του ποντικιού πάνω στην οθόνη. Το τελευταίο πεδίο της δομής είναι και το πλέον χρήσιμο για εμάς, αφού μας δίνει τα συμβάντα του ποντικιού με μορφή bit mask.

getmouse() Διαβάζει το συμβάν που μόλις ανιχνεύσαμε με την *wgetch()* σε ένα παράθυρο, αφαιρώντας το από τον buffer του ποντικιού. Θα επιστρέψει την τιμή OK εάν το συμβάν είναι πράγματι ορατό μέσα στο παράθυρο διαφορετικά την τιμή ERR. Όταν επιστρέφεται η τιμή OK τότε οι συντεταγμένες *x, y* γίνονται οι γνωστές μας (y,x) συντεταγμένες με σύστημα αναφοράς την οθόνη και όχι το παράθυρο. Επίσης στο πεδίο bstate έχει ενεργοποιηθεί ένα bit που μαρτυρεί την κατάσταση του ποντικιού.
πρωτότυπο: `int getmouse(MEVENT *event);`

ungetmouse() Επιστρέψει στον buffer του ποντικιού μια δομή τύπου MEVENT. Είναι αντίστοιχη της *ungetch()*.
πρωτότυπο: `int ungetmouse(MEVENT *event);`

Παράδειγμα:

```
MEVENT event;
mousemask(ALL_MOUSE_EVENTS, NULL);

ch=wgetch(question);
if(ch == KEY_MOUSE)
     if( getmouse(&event) == OK )
          if ( (event.bstate & BUTTON1_CLICKED) &&
               (10 == event.y) &&
               (37 <= event.x) && (39 >= event.x) )
          mvwprintw(question, 12, 30, "You said YES!");
```

Με δυο λόγια κάναμε τα εξής: Αφού ενεργοποιήσαμε το ποντίκι, παγιδεύουμε το παράθυρο question. Μόλις πατήσουμε το αριστερό πλήκτρο θα γίνει ο έλεγχος εγκυρότητας με τη δεύτερη if. Στη συνέχεια ζητούμε μια σειρά από συνθήκες να πληρούνται όπου για χάριν παραδείγματος υποθέτουμε ότι είναι το κλικ πάνω στη λέξη YES που βρίσκεται στη θέση $(10, 37)$ της οθόνης (και μέσα στο παράθυρο question).

3.11.4 Οι υπόλοιπες ποντικοσυναρτήσεις

Έχουμε μια συνάρτηση με την οποία ελέγχουμε εάν ένα συμβάν έγινε μέσα στη περιοχή κάποιου παραθύρου *wenclose()*. Δύο ακόμα συναρτήσεις που μετατρέπουν τις συντεταγμένες της θέσης του ποντικιού σε συντεταγμένες οθόνης *mouse_trafo(), wmouse_trafo()*. Και μια τελευταία συνάρτηση που μας επιτρέπει να ορίζουμε το διάστημα που μεσολαβεί μεταξύ press & release ενός κουμπιού (διάρκεια κλικ) *mouseinterval()*. Λεπτομέρειες μπορείτε να βρείτε στο `man curs_mouse`.

Συναρτήσεις:	Πρωτότυπα:
wenclose()	`bool wenclose(const WINDOW *win, int y, int x);`
mouse_trafo()	`bool mouse_trafo(int* pY, int* pX, bool to_screen);`
wmouse_trafo()	`bool wmouse_trafo(const WINDOW* win, int* pY, int* pX, bool to_screen);`
mouseinterval()	`int mouseinterval(int erval);`

3.12 Διάφορες άλλες χρήσιμες συναρτήσεις

Μέχρι τώρα είδαμε μια πληθώρα βασικών συναρτήσεων και μάθαμε να κάνουμε πάρα πολλά πράγματα με τη βιβλιοθήκη ncurses. Στις σελίδες που ακολουθούν αποφάσισα να συμπεριλάβω μερικές ακόμη χρήσιμες συναρτήσεις με σκοπό να σας φέρω σε επαφή και με άλλες πτυχές της βιβλιοθήκης. Ωστόσο, θα πρέπει να γνωρίζετε ότι οι δυνατότητες που μας δίνει η ncurses δεν εξαντλούνται εδώ. Ειδικά για αυτούς που θα αποφασίσουν να χρησιμοποιούν το API της βιβλιοθήκης για τις εφαρμογές τους είναι απαραίτητη η αναλυτική ανάγνωση των [1] και [9], καθώς και η εκτενής επίσκεψη στον ιστοτόπο `http://dickey.his.com/ncurses`. Για τους υπόλοιπους, που μπορεί απλώς να επιθυμούν να δώσουν έναν άλλον αέρα στα προγράμματά τους, οι γνώσεις που απέκτησαν μέχρι τώρα μαζί με όσα ακολουθούν τους είναι υπεραρκετά. Σε κάθε περίπτωση, σε όποια κατηγορία αναγνωστών και εάν ανήκετε, εμένα δε μου μένει παρά να σας ευχηθώ: Καλή επιτυχία!

3.12.1 Συναρτήσεις μορφοποίησης παρασκηνίου (window background manipulation routines)

Στην παράγραφο 3.8 είδαμε πως να αλλάζουμε το παρασκήνιο για τους χαρακτήρες που τυπώνουμε. Η επιλογή του χρώματος γινόταν μέσα από το χρωματικό ζευγάρι που αρχικοποιούμε κάπου στην αρχή του προγράμματος και αναφερόμαστε σε αυτό με το COLOR_PAIR(). Μπορούμε παρόλα αυτά να φτάσουμε στο ίδιο αποτέλεσμα και με άλλο τρόπο. Με τις συναρτήσεις που θα γνωρίσουμε σε αυτή την παράγραφο, θα δούμε πως γίνεται να χρωματίσουμε το παρασκήνιο ενός ολόκληρου παραθύρου. Υπάρχει όμως μια επιπλέον ιδιότητα που κάνει ενδιαφέρουσες αυτές τις συναρτήσεις. Αφού κάθε θέση της οθόνης είναι τύπου chtype, δηλαδή χαρακτήρας με ιδιότητες, οι συναρτήσεις αυτές μας δίνουν τη δυνατότητα να παράγουμε το οπτικό εφέ μιας ταπετσαρίας, που στην ουσία είναι η εκτύπωση ενός επαναλαμβανόμενου μοτίβου ενός χαρακτήρα σε όλη την έκταση του παραθύρου.

bkgdset() Δίνει στο παρασκήνιο του παραθύρου stdscr τις ιδιότητες που ζητάμε, οι οποίες όμως δεν εφαρμόζονται αμέσως.
Π.χ. `bkgdset(COLOR_PAIR(1));`

πρωτότυπο: `void bkgdset(chtype ch);`

wbkgdset() Δίνει στο παρασκήνιο του παραθύρου *win* κάποιες ιδιότητες, χωρίς να τις εφαρμόζει.
Π.χ. `wbkgdset(money,'$'|A_BOLD|COLOR_PAIR(2));`

πρωτότυπο: `void wbkgdset(WINDOW *win, chtype ch);`

bkgd() Το παρασκήνιο της πρότυπης οθόνης αλλάζει στις νέες ιδιότητες που δίνουμε σαν όρισμα εφαρμόζοντάς τες την ίδια στιγμή.
Π.χ. `bkgd('*');`

πρωτότυπο: `int bkgd(chtype ch);`

wbkgd() Κάνει το ίδιο με την προηγούμενη αλλά για το παράθυρο *win*.
Π.χ. `wbkgd(empty_field, '_'|A_DIM);`

πρωτότυπο: `int wbkgd(WINDOW *win, chtype ch);`

Οι παραπάνω συναρτήσεις έχουν κάποιες παρενέργειες που πρέπει να γνωρίζουμε: Κάθε φορά που τυπώνουμε κάτι μέσα στο παράθυρο, οι εκτυπώσιμοι χαρακτήρες συνδυάζονται με το παρασκήνιο ως εξής: Οι μη λευκοί εκτυπώσιμοι χαρακτήρες (τα γράμματα, οι αριθμοί και τα σημεία στί-

ξης), παίρνουν την ιδιότητα χρώματος, και στυλ του παρασκηνίου (έντονα γράμματα, υπογράμμιση κ.τ.λ.), ενώ οι λευκοί χαρακτήρες, τα κενά και το TAB, αφήνουν το παρασκήνιο ανεπηρέαστο με αποτέλεσμα να φαίνεται και ο χαρακτήρας-μοτίβο που ενδεχομένως να έχουμε χρησιμοποιήσει για να καλύψουμε την επιφάνεια του παραθύρου. Κατά συνέπεια προτείνω να περιοριστούμε στο χρωματισμό του παρασκηνίου, αφού η απερίσκεπτη χρήση του μοτίβου θα καθιστά το κείμενό μας δυσανάγνωστο. Μια ενδεδειγμένη χρήση του μοτίβου θα ήταν στις περιπτώσεις που τυπώνουμε γραφήματα στην οθόνη, οπότε και για κάθε κατηγορία αποτελεσμάτων επιλέγουμε ένα διαφορετικό χαρακτήρα για μοτίβο. Μάλιστα η συγκεκριμένη χρήση είναι επιβεβλημένη για τα τερματικά που δεν υποστηρίζουν χρώμα, αφού δεν υπάρχει άλλος εύκολος τρόπος για να ζωγραφίσουμε την οριοθετημένη περιοχή ενός παραθύρου. (Εννοείται πως για τη χρήση του μοτίβου σε γραφήματα είμαστε αναγκασμένοι να χρησιμοποιούμε υποπαράθυρα για κάθε μια από τις κατηγορίες αποτελεσμάτων που θέλουμε να «χρωματίσουμε»).

Επίσης, αν θυμάστε, είχαμε πει ότι με τις συναρτήσεις χειρισμού ιδιοτήτων, για παράδειγμα *wattrset()*, μπορούμε να επιλέξουμε κάποιες ιδιότητες που θέλουμε να έχει το κείμενο που θα εμφανίσουμε μέσα στο παράθυρο, μαζί και το χρώμα. Η διαφορά της *wattrset()* με την *wbkgd()*, είναι ότι η δεύτερη πραγματοποιείται αμέσως, χρωματίζοντας την επιφάνεια του παραθύρου, ενώ η πρώτη εφαρμόζει την ιδιότητα χρώμα (ή όποια άλλη) απευθείας πάνω στο κείμενο που θα εκτυπώσουμε εφεξής μέσα στο παράθυρο, αφήνοντας όλο τον υπόλοιπο χώρο του παραθύρου ανεπηρέαστο.

Για να γίνουν όλα τα παραπάνω σαφή, άλλος δρόμος δεν υπάρχει, εκτός από τον πειραματισμό: Δοκιμάστε να τυπώσετε σε ένα παράθυρο πριν και μετά την εφαρμογή των συναρτήσεων *wattrset()* και *wbkgd()* χρησιμοποιώντας κάποιες ιδιότητες.

3.12.2 Σχεδιασμός γραμμών

Όπως είδαμε οι line and box drawing characters του OEM Extended ASCII Code μπορούν να χρησιμοποιηθούν για να περιγράψουν ένα παράθυρο. Η χρήση τους όμως δεν περιορίζεται σε αυτό. Η βιβλιοθήκη curses μας παρέχει ακόμη μερικές συναρτήσεις για να χρησιμοποιήσουμε αυτούς ή και άλλους χαρακτήρες έτσι ώστε να σχηματίζουνε οριζόντιες και κάθετες γραμμές.

Θα τυπώσουμε οριζόντιες γραμμές με τις συναρτήσεις:

hline() *πρωτότυπο*:
```
int hline(chtype ch, int n);
```
whline() *πρωτότυπο*:
```
int whline(WINDOW *win, chtype ch, int n);
```
mvhline() *πρωτότυπο*:
```
int mvhline(int y, int x, chtype ch, int n);
```
mvwhline() *πρωτότυπο*:
```
int mvwhline(WINDOW *, int y, int x, chtype ch, int n);
```

Για να πάρουμε τις αντίστοιχες συναρτήσεις για τις κάθετες γραμμές αλλάζουμε μόνο το γράμμα *h* σε *v*, ενώ η λίστα με τις παραμέτρους παραμένει η ίδια, δηλαδή είναι *vline()*, *wvline()*, *mvvline()*, *mvwvline()*.

Οι συναρτήσεις *hline()* και *whline()*, χωρίς να αλλάζουνε τη τρέχουσα θέση του δρομέα, τυπώνουν για τις επόμενες *n* θέσεις, (το τέλος της γραμμής είναι φράγμα και σταματάει), τον χαρακτήρα *ch*. Οι συναρτήσεις *mvhline()* και *mvwhline()* κάνουν το ίδιο μόνο που πραγματοποιούν πρώτα μετακίνηση του δρομέα στη θέση (y,x).

Ασφαλώς τα αντίστοιχα ισχύουν για τις κάθετες γραμμές. Η εκτύπωση για τις οριζόντιες γραμμές γίνεται από αριστερά προς τα δεξιά και για τις κάθετες από πάνω προς τα κάτω. Ας δούμε λίγα παραδείγματα:

```
whline(top, '+', 10);
mvwhline(bottom, LINES-3, 0, ACS_HLINE|COLOR_PAIR(1), COLS);
```

```
wvline(right, ACS_VLINE, 23);
mvwvline(temperature, 5, 2, '#'|COLOR_PAIR(2), 14);
mvhline(5,COLS/2-15,0,30);
```

Το μηδέν στο τελευταίο παράδειγμα χρησιμοποιείται για να δηλώσει ότι θέλουμε να χρησιμοποιήσει το προκαθορισμένο χαρακτήρα ACS_HLINE, με άλλα λόγια είτε μηδέν βάλουμε είτε ACS_HLINE θα παράγουμε το ίδιο αποτέλεσμα.

Οι συναρτήσεις αυτές μπορούνε κάλλιστα να χρησιμοποιηθούν για να χωρίσουμε τμήματα της οθόνης, να φτιάξουμε λίστες, να υπογραμμίσουμε τίτλους ή να κατασκευάσουμε ραβδογράμματα, και γενικά ότι άλλο δημιουργικό σκεφτείτε!

3.12.3 Συναρτήσεις για τις συντεταγμένες

Καθώς η θέση κάθε χαρακτήρα στην οθόνη καθορίζεται από ένα ζευγάρι συντεταγμένων, αναμενόμενο είναι πως θα υπάρχουν και μερικές συναρτήσεις ειδικά για αυτές. Έτσι λοιπόν, έχουμε συναρτήσεις που μεταφέρουν τον κέρσορα σε μια θέση της οθόνης και άλλες που μας αναφέρουν αυτή τη θέση ανά πάσα στιγμή.

move() Αυτή η συνάρτηση τοποθετεί τον κέρσορα στη γραμμή *y* και στη στήλη *x* της οθόνης stdscr. Η θέση (0,0) βρίσκεται στην πάνω αριστερή γωνία της οθόνης. (Συντεταγμένες σχετικές στην οθόνη).

πρωτότυπο:

```
int move(int y, int x);
```

wmove() Μετακινεί τον κέρσορα στη θέση (y,x) του παραθύρου *win*. Εδώ, η θέση (0,0) είναι ο πρώτος χαρακτήρας του παραθύρου στην πάνω αριστερή του γωνία. (Συντεταγμένες σχετικές στο παράθυρο).

πρωτότυπο:

```
int wmove(WINDOW *win, int y, int x);
```

getyx() Οι συντεταγμένες της τρέχουσας θέσης του κέρσορα ανατίθενται στις ακέραιες μεταβλητές *y* και *x*. Οι συντεταγμένες (0,0) δηλώνουν την πρώτη θέση του παραθύρου *win*. (Συντεταγμένες σχετικές στο παράθυρο).

πρωτότυπο:

```
void getyx(WINDOW *win, int y, int x);
```

getparyx() Το ίδιο, όμως τώρα οι συντεταγμένες (0,0) βρίσκονται στην πρώτη θέση του πατρικού παραθύρου του παραθύρου *win*. (Συντεταγμένες σχετικές στο πατρικό παράθυρο).

πρωτότυπο:

```
void getparyx(WINDOW *win, int y, int x);
```

getbegyx() Μας δίνει τις συντεταγμένες της αρχής του παραθύρου. Πρόκειται για τις δύο τελευταίες παραμέτρους της συνάρτησης *newwin()* (ή *subwin()*, αλλά όχι της *derwin()*!). (Συντεταγμένες σχετικές στην οθόνη).

πρωτότυπο:

```
void getbegyx(WINDOW *win, int y, int x);
```

getmaxyx() Μας δίνει τη διάσταση του παραθύρου. Είναι οι διαστάσεις που περνάμε στη συνάρτηση *newwin()* ως πρώτο και δεύτερο όρισμα ή αλλιώς η δεύτερη και τρίτη παράμετρος της συνάρτησης *subwin()* ή ακόμα οι δύο τελευταίες παράμετροι της συνάρτησης *derwin()*.

πρωτότυπο:

```
void getmaxyx(WINDOW *win, int y, int x);
```

Οι μεταβλητές *y*, *x* χρησιμοποιούνται χωρίς το & ως εξής:

```
WINDOW *table;
int rows=0, cols=0;
[...]
getmaxyx(table, rows, cols);
printw("The table has %d rows and %d columns.", rows, cols);
```

Για τις συναρτήσεις αυτές συμβουλευτείτε το `man curs_move` και `man curs_getyx`.

3.12.4 Αποθήκευση της οθόνης σε αρχείο & ανάκτησή της

Μπορούμε να αποθηκεύσουμε στιγμιότυπα της οθόνης σε αρχείο και να τα ανακτήσουμε αργότερα:

scr_dump() Αποθηκεύει την οθόνη στο αρχείο *filename*. (Βεβαιωθείτε ότι έχετε κάνει όλα τα απαραίτητα *wrefresh()* και *doupdate()* προηγουμένως...)

πρωτότυπο:

```
int scr_dump(const char *filename);
```

scr_restore() Ανακτά από το αρχείο *filename* τα δεδομένα που αποθηκεύσαμε με την *scr_dump()*. Θα πρέπει να καλέσουμε την *doupdate()* για να τα εμφανίσουμε στην οθόνη.

πρωτότυπο:

```
int scr_restore(const char *filename);
```

Η χρήση αυτής της δυνατότητας είναι διαδεδομένη σε πολλά παιχνίδια για κονσόλα όπου εναλλάσσουν συχνά τις οθόνες (πηγαίνοντας μπρος-πίσω) και έτσι επιτυγχάνουν γρηγορότερη εκτύπωση. Επίσης μπορεί να είναι χρήσιμη λειτουργία για την επαναφορά της εφαρμογής μας στην κατάσταση που βρισκόταν πριν μας κόψει η Δ.Ε.Η. το ρεύμα. Στο `man curs_scr_dump` θα βρείτε δυο ακόμα συναρτήσεις που θα σας βοηθήσουν να υλοποιήσετε μια στοιχειώδη προστασία απέναντι σε αυτό το σύνηθες φαινόμενο.

3.12.5 Αποθήκευση ενός παραθύρου σε αρχείο & ανάκτησή του

Μας παρέχονται αυτές οι δυο συναρτήσεις για αποθηκεύουμε ένα παράθυρο σε αρχείο και να το διαβάζουμε:

putwin() Αποθηκεύει στο αρχείο που δείχνει το *filep*, ολόκληρη τη δομή WINDOW του παραθύρου *win*.

πρωτότυπο:

```
int putwin(WINDOW *win, FILE *filep);
```

getwin() Χρησιμοποιείται για να ανακτούμε το περιεχόμενο του παραθύρου από ένα αρχείο. Είναι ανάλογη με την *newwin()* που όμως επιπροσθέτως αρχικοποιεί και τα κελιά του παραθύρου με τα περιεχόμενα που βρήκε στο αρχείο.

πρωτότυπο:

```
WINDOW *getwin(FILE *filep);
```

Αυτές οι συναρτήσεις βρίσκονται στο `man curs_util` μαζί με κάποιες ακόμη που μπορεί να θέλετε να χρησιμοποιήσετε.

3.12.6 Ρυθμίσεις για τον τρόπο που χειρίζεται η βιβλιοθήκη την οθόνη

Με τις συναρτήσεις που ακολουθούν μπορούμε να κάνουμε κάποιες ρυθμίσεις για τον τρόπο με τον οποίο η βιβλιοθήκη curses χειρίζεται την οθόνη. Με το που ξεκινάμε το API της ncurses, αμέσως μετά την κλήση της *initscr()*, όλες αυτές έχουν αρχικά την τιμή FALSE. Μέσα στο πρόγραμμα μπορούμε να εναλλάσσουμε τα TRUE και FALSE όσες φορές θέλουμε. Επίσης, δεν είναι απαραίτητο να επαναφέρουμε μια ιδιότητα, που αλλάξαμε σε TRUE, πριν καλέσουμε την *endwin()*, καθώς αυτό γίνεται αυτόματα από τη βιβλιοθήκη.

clearok() Όταν αυτή η συνάρτηση καλείται με δεύτερη παράμετρο TRUE, τότε κάθε εντολή *wrefresh(win)* που ακολουθεί αναγκάζει τη βιβλιοθήκη ncurses να «ζωγραφίζει» από την αρχή ολόκληρο το παράθυρο *win* και όχι μόνο τα σημεία που έχουν αλλάξει.

πρωτότυπο:

```
int clearok(WINDOW *win, bool bf);
```

Σημείωση: Αυτή η συνάρτηση είναι μια από τις ελάχιστες εξαιρέσεις όπου επιτρέπεται η χρήση της μεταβλητής *curscr* σαν πρώτο όρισμα. Όταν λοιπόν καλέσουμε `clearok(curscr, TRUE)` κάθε *wrefresh()* που ακολουθεί και για *οποιοδήποτε παράθυρο* θα ζωγραφίσει εκ νέου *ολόκληρη την οθόνη* γραμμή προς γραμμή.

immedok() Η ενεργοποίηση αυτής της συνάρτησης για το παράθυρο *win* θα προκαλέσει —πανικό στην οθόνη μας—. Συγκεκριμένα, με κάθε κλήση εκτύπωσης *waddch(), wprintw()*, κ.τ.λ. αυτομάτως θα καλείται και μια *wrefresh(win)* ανανεώνοντας συνεχώς το περιεχόμενο του παραθύρου μας. Εκτός από την ενοχλητική για το μάτι μας κατάσταση, μειώνεται δραματικά και η ταχύτητα εκτέλεσης του προγράμματος. Δοκιμάστε την σε τερματικό με ρυθμό μετάδοσης 9600 bps και είμαι βέβαιος πως θα πειστείτε να την αποφεύγετε.

πρωτότυπο:

```
void immedok(WINDOW *win, bool bf);
```

scrollok() Ρυθμίζει την ολίσθηση του παραθύρου. Με το FALSE, όταν ο κέρσορας βρεθεί στη τελευταία θέση του παραθύρου θα παραμείνει εκεί. Με το TRUE, οι γραμμές του παραθύρου ολισθαίνουν προς τα πάνω επιτρέποντας έτσι στο κέρσορα να αλλάξει γραμμή.

πρωτότυπο:

```
int scrollok(WINDOW *win, bool bf);
```

setscrreg() Επιτρέπει μόνο στην περιοχή που οριοθετείται από τη γραμμή *top* έως τη γραμμή *bot* να ολισθαίνει. (Φυσικά πρέπει να έχουμε καλέσει και την *scrollok()*).

πρωτότυπο:

```
int setscrreg(int top, int bot);
```

wsetscrreg() Είναι η ίδια με την προηγούμενη όμως στη γενική της μορφή, για οποιοδήποτε παράθυρο και όχι μόνο για τη stdscr.

πρωτότυπο:

```
int wsetscrreg(WINDOW *win, int top, int bot);
```

Περισσότερες λεπτομέρειες για τα παραπάνω διαβάστε στο `man curs_outopts`.

3.13 Παραδείγματα

Αφού λοιπόν γνωρίσαμε όλες αυτές τις υπέροχες δυνατότητες της βιβλιοθήκης έφτασε αισίως η ώρα να δούμε μέσα από ολοκληρωμένα προγράμματα πως μπορούμε να τις συνθέσουμε για να δημιουργήσουμε στη κονσόλα μας ένα ελκυστικότερο περιβάλλον για τους χρήστες των εφαρμογών μας.

Ξεκινώντας με απλά προγράμματα προχωρώ σε πιο σύνθετα προσπαθώντας έτσι να καλύψω όλες τις συναρτήσεις της βιβλιοθήκης στις οποίες αναφέρθηκα. Παράλληλα όμως, θέλω να πιστεύω ότι δίνοντάς σας αυτές τις εφαρμογές, θα σας κάνω να ερωτευτείτε τη βιβλιοθήκη και να αρχίσετε να την χρησιμοποιείτε και εσείς στα προγράμματά σας.

Τώρα εάν θέλετε να ξεκινήσετε μελετώντας πολύ πιο απλό κώδικα, δείτε στο [1] πολλά παραδείγματα που εξυπηρετούν άριστα την αποστολή τους: Να σας εισάγουν στο προγραμματισμό με το API της βιβλιοθήκης ncurses. Σημειώστε ότι οι πηγαίοι κώδικες διατίθενται μαζί με το κείμενο και σας ενθαρρύνω να πειραματιστείτε πάνω τους δοκιμάζοντας τις συναρτήσεις.

Στη συνέχεια, αξίζει πραγματικά να δείτε τα παραδείγματα που έρχονται μαζί με το πακέτο εγκατάστασης της ίδιας της βιβλιοθήκης και μάλιστα ανάμεσά τους ξεχωρίζω τα: `bs, dots, firework, gdc, hanoi, knight` και `worm`. Μελετήστε τον πηγαίο κώδικα, μεταγλωττίστε τα και τρέξτε τα εκτελέσιμα.

3.13.1 Παράδειγμα 1: Το πρόγραμμα `program1.c` (σελ. 4)

Ορίστε λοιπόν πως θα πρέπει να γράψουμε το πρόγραμμα `program1.c` για να πετύχουμε το ίδιο αποτέλεσμα με τη βελτιωμένη εκδοχή της βιβλιοθήκης asciiart (`program3.c`). Τα κύρια σημεία του προγράμματος αυτού είναι η εναλλαγή του echo-noecho mode, η ενεργοποίηση των χρωμάτων και η ολίσθηση της οθόνης. Το πρόγραμμα προσφέρεται για πειραματισμούς: Δοκιμάστε να σχολιάσετε τη γραμμή 18, κατόπιν αλλάξτε τις *printw()* με *addstr()* και αφαιρέστε την *noecho()* στη γραμμή 38. Δείτε πώς συμπεριφέρεται το πρόγραμμα σε κάθε μια από αυτές τις περιπτώσεις.

Μεταγλώττιση:

(george@earth) ~/asnc-source-1.0/ncurses$ **gcc -o program8 program8.c -lncurses**

Πρόγραμμα 8: asnc-source-1.0/ncurses/program8.c

```
1  /* program8.c :
     A simple guessing game: Guess an integer between 1-10.
     (program1.c with ncurses look & feel) */

5  #include <ncurses.h>
   #include <stdlib.h>
   #include <time.h>

   int main(void) {
10   int num=0, guess=0, c=0;

     initscr();
     cbreak();

15   start_color(); /* request colors */
     init_pair(1,COLOR_RED,COLOR_BLACK); /* pair 1 */
     init_pair(2,COLOR_GREEN,COLOR_BLACK); /* pair 2 */

     scrollok(stdscr,TRUE);
```

```
20   /* uncomment the next line if scrolling has bad behaviour */
     /* idlok(stdscr, TRUE); */
       srand(time(NULL)); /* randomization seed */

       do { addstr("Guess a number between 1-10.");
25         num = rand()%10 +1;  /* 1, 2, ..., 10 */

           do { printw("\nYour guess is: ");
               scanw("%d", &guess);
               if( guess!=num) { color_set(1, NULL);
30                         printw("Sorry, try again!");
                           color_set(0, NULL); } /* white on black */

           } while( num!=guess );

35         color_set(2, NULL);
           printw("Yes, that's it!\n");
           color_set(0, NULL); /* white on black again */
           printw("Play again? (Q: quits)\n");
           noecho();
40         c=getch();
           echo();

       } while( c!='Q' && c!='q' );

45     endwin();

    return EXIT_SUCCESS; }
```

Σημείωση: Οι εντολές στις γραμμές 31 και 37, όπως ανακάλυψα παίζοντας μαζί τους, δε λειτουργούν όπως αναμενόταν. Το χρώμα δεν αλλάζει στο προκαθορισμένο χρωματικό ζευγάρι 0 (λευκό/μαύρο). Πρόκειται για ένα bug στην έκδοση 5.3 της βιβλιοθήκης ncurses για το οποίο ενημέρωσα τον Dickey και συμπεριέλαβε μια διόρθωση στο patch 20031206. Έτσι από την έκδοση 5.4 και μετά το bug έχει απαλειφθεί. Τώρα για να δουλέψει σωστά το πρόγραμμα σε όσους έχουνε ακόμα παλαιότερες εκδόσεις της βιβλιοθήκης θα πρέπει να κάνετε τα εξής:

Μετά τη γραμμή 17 προσθέτουμε:

```
init_pair(3, COLOR_WHITE, COLOR_BLACK);
```

Και οι γραμμές 31 και 37 θα συμμορφωθούν με την αλλαγή:

```
color_set(3, NULL);
```

3.13.2 Παράδειγμα 2: Ανάλυση αρχείου κειμένου (textfile).

Το πρόγραμμα φτιάχνει το ιστόγραμμα ενός αρχείου κειμένου και τυπώνει οριζόντια και κατακόρυφα ραβδογράμματα με τα μήκη των λέξεών του. Έχει επιλογή να δεχτεί κείμενο από το χρήστη και άλλη επιλογή που αλλάζει την οπτική παράσταση του ραβδογράμματος. Η ιδέα του προγράμματος προέρχεται από την άσκηση 1.13 του [3]. Ο αλγόριθμος της συνάρτησης *process()* είναι δανεισμένος από το πρόγραμμα που προτείνει ως λύση για την άσκηση 1.13 ο Richard Heathfield, στη σελίδα `http://users.powernet.co.uk/eton/kandr2/krx113.html`.

Μεταγλώττιση:

(george@earth) ~/asnc-source-1.0/ncurses$ **gcc -o program9 program9.c -lncurses**

Πρόγραμμα 9: asnc-source-1.0/ncurses/program9.c

```
#include <ncurses.h>

/*    prototypes    */
void draw_options(void);
void textfield(void);
void draw_welcome(void);
void message(int);
void process(FILE *);
int checkfordata(void);
void horizontal(void);
void vertical(void);
int count_cols(long);
int count_lines(long);
void cleanup(void);
void clear_ist(void);
void textfeed(void);

WINDOW *options=NULL,
          *hor=NULL,
          *ver=NULL,
          *right_side=NULL,
          *ffield=NULL,
          *ih[11]={NULL},
          *iv[11]={NULL};

FILE  *textfile=NULL;

int   max_y, max_x;
char  fname[31];
bool  style[4]={ TRUE, FALSE, TRUE, FALSE};
long  lengtharr[11];

int main() {

chtype ch=0;

/* initialisation */
initscr(); raw(); noecho(); curs_set(0); start_color();

init_pair(1,COLOR_WHITE,COLOR_BLUE);
init_pair(2,COLOR_YELLOW,COLOR_BLACK);

draw_options();
textfield();
draw_welcome();
wrefresh(options);

/* keyboard trap */
do { switch(ch=wgetch(options)) {
     case 'q': /* fall through */
     case 'Q': break;
     case 'f': /* fall through */
     case 'F':
```

```
            echo(); curs_set(1);
            wmove(ffield,0,0);
            wclrtoeol(ffield);
            wrefresh(ffield);
            scrollok(ffield,TRUE);
            wgetnstr(ffield,fname,30);
            noecho(); curs_set(0);
            if (NULL!=textfile) fclose(textfile);
            clear_ist();
            if (NULL==(textfile=fopen(fname,"r"))) {
              message(2);
              wgetch(right_side);
              textfield();
              draw_welcome();
              break;
            }
            process(textfile);
            (TRUE==style[0]) ? horizontal() : vertical();
            break;
          case 'u': /* fall through */
          case 'U':
            if (NULL!=textfile) fclose(textfile);
            textfile=tmpfile();
            clear_ist();
            textfeed();
            process(textfile);
            (TRUE==style[0]) ? horizontal() : vertical();
            break;
          case 'v': /* fall through */
          case 'V':
            if (TRUE==style[0]) {
              style[0]=FALSE; style[1]=TRUE;
              mvwchgat(options,8,2,10,A_NORMAL,0,NULL);
              mvwchgat(options,9,2,8,A_REVERSE,2,NULL);
            } else {
              style[0]=TRUE; style[1]=FALSE;
              mvwchgat(options,8,2,10,A_REVERSE,2,NULL);
              mvwchgat(options,9,2,8,A_NORMAL,0,NULL);
            }
            wtouchln(options,8,2,1);
            wrefresh(options);
            if(FALSE==checkfordata()) break;
            (TRUE==style[0]) ? horizontal() : vertical();
            break;
          case 'd': /* fall through */
          case 'D':
            if (TRUE==style[2]) {
              style[2]=FALSE; style[3]=TRUE;
              mvwchgat(options,12,2,5,A_NORMAL,0,NULL);
              mvwchgat(options,13,2,7,A_REVERSE,2,NULL);
            } else {
              style[2]=TRUE; style[3]=FALSE;
              mvwchgat(options,12,2,5,A_REVERSE,2,NULL);
              mvwchgat(options,13,2,7,A_NORMAL,0,NULL);
            }
            wtouchln(options,12,2,1);
            wrefresh(options);
```

```
            if(FALSE==checkfordata()) break;
            (TRUE==style[0]) ? horizontal() : vertical();
            break;
        default: /* something else */
            /* ignore, we could flash() or beep() */
            break;
        }
    } while ( ch != 'q' && ch != 'Q' );

    cleanup();
    endwin();
    return 0;
    } /* end of main() */

    /*============================*/
    void draw_options()
    /*============================*/
    {
    /* request a new window */
    options=newwin(16,17,LINES/2-8,2);

    /* set the color for the text and print it */
    wcolor_set(options,0,NULL);
    waddstr(options, "Options:");
    mvwaddstr(options, 2,0,"Input from file");
    mvwaddstr(options, 3,0,">");
    mvwaddstr(options, 5,0,"Input from user");
    mvwaddstr(options, 7,0,"Change view style");
    mvwaddstr(options, 8,2,"Horizontal");
    mvwaddstr(options, 9,2,"Vertical");
    mvwaddstr(options,11,0,"Change draw style");
    mvwaddstr(options,12,2,"Color");
    mvwaddstr(options,13,2,"Pattern");
    mvwaddstr(options,15,0,"Quit");

    /* hightlight the letters we'll use to access the options */
    mvwchgat(options,2,11,1,A_BOLD,2,NULL);
    mvwchgat(options,5,11,1,A_BOLD,2,NULL);
    mvwchgat(options,7,7,1,A_BOLD,2,NULL);
    mvwchgat(options,8,2,10,A_REVERSE,2,NULL);
    mvwchgat(options,11,7,1,A_BOLD,2,NULL);
    mvwchgat(options,12,2,5,A_REVERSE,2,NULL);
    mvwchgat(options,15,0,1,A_BOLD,2,NULL);

    return;
    } /* end draw_options() */


    /*============================*/
    void textfield(void)
    /*============================*/
    {
    /* create the text field */
    if (NULL==ffield) ffield=derwin(options,1,16,3,1);
    wclear(ffield);
    wbkgd(ffield, '_');
    mvwaddstr(ffield, 0,0,"filename");
```

```
    mvwchgat(ffield,0,0,8,A_REVERSE,2,NULL);
    wrefresh(ffield);
170
    return;
    } /* end textfield() */


175 /*============================*/
    void draw_welcome()
    /*============================*/
    {

180 getmaxyx(stdscr,max_y,max_x);
    if (NULL==right_side) right_side=newwin(max_y-2,max_x-22,1,20);

    message(1);

185 return;
    } /* end draw_welcome() */


    /*============================*/
190 void message(int chiffre)
    /*============================*/
    {
      wclear(right_side);

195   getmaxyx(right_side,max_y,max_x);

      switch(chiffre) {
             case 1:
                      mvwaddstr(right_side,max_y/2-1,max_x/2-20,
200                        "Use the highlighted keys from the options");
                      mvwaddstr(right_side,max_y/2,max_x/2-17,
                           "menu to interact with the program.");
                      break;
             case 2:
205                   mvwprintw(right_side,max_y/2-1,max_x/2-14,
                           "Unable to process the file.");
                      mvwprintw(right_side,max_y/2,max_x/2-13,
                           "The file was not found or");
                      mvwprintw(right_side,max_y/2+1,max_x/2-15,
210                        "some other error has occurred.");
                      break;
             case 3:
                      mvwprintw(right_side,max_y/2-1,max_x/2-9,
                           "Nothing to display.");
215                   break;
             default:
                      mvwprintw(right_side,max_y/2,max_x/2-16,
                           "You should never read this line.");
             break;
220   }

    wrefresh(right_side);
    return;
    } /* end message() */
```

```
225
    /*===========================*/
    void process(FILE *fn)
    /*===========================*/
    {
230  int c;
     int inspace = 0;
     int wordlen = 0;

     int firstletter = 1;
235  long thisval = 0;
     long maxval = 0;
     int thisidx = 0;
     int done = 0;

240  for(thisidx = 0; thisidx <= 10; thisidx++)
     {
      lengtharr[thisidx] = 0;
     }

245  rewind(fn);  /* needed when we use a tempfile */

     while(done == 0)
     {
      c = fgetc(fn);
250
      if(c == ' ' || c == '\t' || c == '\n' || c == EOF)
      {
       if(inspace == 0)
       {
255     firstletter = 0;
        inspace = 1;

        if(wordlen <= 10)
        {
260      if(wordlen > 0)
         {
          thisval = ++lengtharr[wordlen - 1];
          if(thisval > maxval)
          {
265        maxval = thisval;
          }
         }
        }
        else
        {
270      thisval = ++lengtharr[10];
         if(thisval > maxval)
         {
          maxval = thisval;
         }
        }
       }
       if(c == EOF)
       {
280     done = 1;
       }
```

```
      }
      else
      {
285    if(inspace == 1 || firstletter == 1)
       {
        wordlen = 0;
        firstletter = 0;
        inspace = 0;
290    }
       ++wordlen;
      }
     }
    return;
295 } /* end of process() */

    /*============================*/
    int checkfordata(void)
    /*============================*/
300 {
     if (NULL==textfile) { message(3); return FALSE; }
     return TRUE;
    } /* end of checkfordata() */

305 /*============================*/
    void horizontal(void)
    /*============================*/
    {
    long i;
310 int cols=0;

    wclear(right_side);wrefresh(right_side);
    getmaxyx(right_side,max_y,max_x);
    if (NULL==hor) hor=derwin(right_side,14,44,max_y/2-7,max_x/2-22);
315
    mvwprintw(right_side,max_y-1,max_x/2-14,"H rule: words, V rule: length");
    wrefresh(right_side);

    mvwhline(hor,12,4,0,41);
320 mvwvline(hor,0,3,0,12);
    mvwaddch(hor,12,3,ACS_LLCORNER);
    wmove(hor,13,4);
    for(i=1;i<16;i++) wprintw(hor,(i<10) ? "%2d" : "%3d",i);
    wprintw(hor," >15");
325 mvwprintw(hor,0,0,">10");
    for(i=0;i<10;i++) mvwprintw(hor,2+i,1,"%2d",10-i);
    mvwhline(hor,1,4,'-',40);
    mvwvline(hor,0,40,'|',12);

330 for(i=0;i<=10;i++) {
        if (10==i) {
            if (0!=lengtharr[i]) {
                cols=count_cols(i);
                if (NULL==ih[i]) ih[i]=derwin(hor,1,cols,0,4);
335             if (TRUE==style[2])
                    wbkgd(ih[i],COLOR_PAIR(1));
                else
                    wbkgd(ih[i],COLOR_PAIR(0)|'#');
```

```
                if (40==cols)
340                  mvwprintw(ih[i],0,cols-10,"%10ld",lengtharr[i]);
            }
        }
        else {
            if (0!=lengtharr[i]) {
345             cols=count_cols(i);
                if (NULL==ih[i]) ih[i]=derwin(hor,1,cols,11-i,4);
                if (TRUE==style[2])
                    wbkgd(ih[i],COLOR_PAIR(1));
                else
350                 wbkgd(ih[i],COLOR_PAIR(0)|'#');
                if (40==cols)
                    mvwprintw(ih[i],0,cols-10,"%10ld",lengtharr[i]);
            }
        }
355 } /* end for loop */

    touchwin(hor);
    wrefresh(hor);

360 return;
    } /* end horizontal() */

    /*============================*/
    void vertical(void)
365 /*============================*/
    {
    long i;
    int lines=0;

370 wclear(right_side);wrefresh(right_side);
    getmaxyx(right_side,max_y,max_x);
    if (NULL==ver) ver=derwin(right_side,19,29,max_y/2-10,max_x/2-14);

    mvwprintw(right_side,max_y-1,max_x/2-14,"H rule: length, V rule: words");
375 wrefresh(right_side);

    mvwhline(ver,17,4,0,25);
    mvwvline(ver,0,3,0,17);
    mvwaddch(ver,17,3,ACS_LLCORNER);
380 wmove(ver,18,4);
    for(i=1;i<11;i++) wprintw(ver,(i<10) ? "%2d" : "%3d",i);
    wprintw(ver," >10");
    mvwprintw(ver,0,0,">15");
    for(i=0;i<15;i++) mvwprintw(ver,2+i,1,"%2d",15-i);
385 mvwhline(ver,1,4,'-',25);
    mvwvline(ver,0,25,'|',17);

    for(i=0;i<=10;i++) {
        if (10==i) {
390         if (0!=lengtharr[i]) {
                lines=count_lines(i);
                if (NULL==iv[i]) iv[i]=derwin(ver,lines,2,17-lines,26);
                if (TRUE==style[2])
                    wbkgd(iv[i],COLOR_PAIR(1));
395             else
```

```
                    wbkgd(iv[i],COLOR_PAIR(0)|'#');
               if (17==lines)
                    mvwprintw(iv[i],0,0,"!!");
          }
     }
     else {
          if (0!=lengtharr[i]) {
               lines=count_lines(i);
               if (NULL==iv[i]) iv[i]=derwin(ver,lines,2,17-lines,4+2*i);
               if (TRUE==style[2])
                    wbkgd(iv[i],COLOR_PAIR(1));
               else
                    wbkgd(iv[i],COLOR_PAIR(0)|'#');
               if (17==lines)
                    mvwprintw(iv[i],0,0,"!!");
          }
     }
} /* end for loop */

touchwin(ver);
wrefresh(ver);

return;
} /* end vertical() */

/*============================*/
int count_cols(long i)
/*============================*/
{
     if (lengtharr[i]<=9)   return lengtharr[i]*2;
     if (lengtharr[i]<=15)  return 9*2+(lengtharr[i]-9)*3;
     return 40;
} /* end count_cols() */

/*============================*/
int count_lines(long i)
/*============================*/
{
     if (lengtharr[i]<=15)  return lengtharr[i];
     return 17; /* more than 15 words */
} /* end count_lines() */

/*============================*/
void cleanup(void)
/*============================*/
{
int i;

if (NULL!=textfile) fclose(textfile);

for(i=0;i<11;i++) {
     if (NULL!=ih[i]) delwin(ih[i]);
     if (NULL!=iv[i]) delwin(iv[i]);
}
if (NULL!=hor) delwin(hor);
if (NULL!=ver) delwin(ver);
delwin(right_side);
```

```
    delwin(ffield);
    delwin(options);

    clear();
    refresh();
    return;

} /* end cleanup */

/*============================*/
void clear_ist(void)
/*============================*/
{
int i;

for(i=0;i<11;i++) {
     if (NULL!=ih[i]) {
          delwin(ih[i]);
          ih[i]=NULL;
     }
     if (NULL!=iv[i]) {
          delwin(iv[i]);
          iv[i]=NULL;
     }
}

return;
} /* end clear_ist() */

/*============================*/
void textfeed(void)
/*============================*/
{
WINDOW *textwinborder,*textwin;
chtype ch;

wclear(right_side);
getmaxyx(right_side,max_y,max_x);
mvwaddstr(right_side,1,max_x/2-16,"Enter your text below and press");
mvwaddstr(right_side,2,max_x/2-9,"Ctrl+D when done.");
wrefresh(right_side);
textwinborder=derwin(right_side,max_y-4,max_x-8,3,max_x/2-(max_x-8)/2);
getmaxyx(textwinborder,max_y,max_x);
textwin=derwin(textwinborder,max_y-2,max_x-2,1,1);
box(textwinborder,0,0);
wbkgd(textwin,COLOR_PAIR(1));
wbkgd(textwinborder,COLOR_PAIR(1));
wrefresh(textwinborder);
wrefresh(textwin);

scrollok(textwin, TRUE);
curs_set(1);
keypad(textwin,TRUE);
while(0x04!=(ch=wgetch(textwin))) {
     if ( (32<=ch && ch<=126) || '\n'==ch) {
          wechochar(textwin,ch);
          fputc(ch,textfile);
```

```
510         }
            else if (KEY_BACKSPACE==ch) {
                 fseek(textfile,-1L,SEEK_CUR);
                 if ( '\n'!=fgetc(textfile) ) {
                      waddstr(textwin,"\b \b");
515                   wrefresh(textwin);
                      fseek(textfile,-1L,SEEK_CUR);
                      fputc(' ',textfile);
                      fflush(textfile);
                      fseek(textfile,-1L,SEEK_CUR);
520              }
                 else
                      continue; /* do nothing */
            }
            else
525              continue; /* ignore input */
       }
       fflush(textfile);
       curs_set(0);

530    return;
       } /* end textfeed() */
```

4 Παραρτήματα

4.1 Αρχεία asciiart.h, asciiart.c

Σημείωση 1: *Τα σχόλια των δύο αυτών αρχείων που αποτελούν τη βιβλιοθήκη asciiart έχουνε αποδοθεί στα αγγλικά. Αυτή τη μεταφρασμένη εκδοχή θα τη βρείτε στην ιστοσελίδα του Γιώργου Ανδρέου (βλέπε 4.2).*

το αρχείο asciiart.h

```
#ifndef ASCII_ART_H
#define ASCII_ART_H

/*    This collection of functions makes possible to know
      the characters on the input stream immediately as they
      are typed. It also let us access some terminal
      attributes like the color. */

/* get one character without echo */
char getch(void);
/* get one character with echo */
char getche(void);
/* use this function to demostrate the usage of this module */
void test_asciiart(void);
/* change the text color; valid values are 0<=i<=7 */
void textcolor(int i);
/* 1 sets and 0 unsets the blinking text (experimental) */
void textblink(int val);
/* reinitialization of the console */
void textreset(void);

/* usage example:
     #include "asciiart.h"
     int main(void) {
     test_asciiart();
     return 0; }
*/
#endif
```

Σημείωση 2: *Στο αρχικό αρχείο asciiart.c η συνάρτηση textreset είναι γραμμένη* `void textreset(voir)` *και πρόκειται για ορθογραφικό λάθος. Εδώ έχει αλλαχτεί σε* `void textreset(void)` *(γραμμή 59).*

το αρχείο asciiart.c

```
1   #include <termios.h>
    #include <unistd.h>
    #include <stdio.h>
    /*-------------------------------------------------------------------------*/
5   /* Reserved to store the terminal attributes */
    struct termios sattr;
    /* Initialize mode : 0=normal 1=no buffer */
    int mode_clavier=0;
    char escape_val=27;
10  void initialisation_clavier(int val );
    char getcharacter(void);
    void restauration_clavier(void);
    /*-------------------------------------------------------------------------*/
    char getch(void)
15  {
        char c;
        if (mode_clavier==1) return getcharacter();
        else
        {
20          initialisation_clavier(0);
            c=getcharacter();
            restauration_clavier();
            return c;
        }
25  }
    /*-------------------------------------------------------------------------*/
    char getche(void)
    {
        char c;
30      if (mode_clavier==1) return getcharacter();
        else
        {
            initialisation_clavier(1);
            c=getcharacter();
35          restauration_clavier();
            return c;
        }
    }
    /*-------------------------------------------------------------------------*/
40  void textcolor(int i)
    {
        if (i<0 || i> 7) return;
        printf("%c[3%1dm",escape_val,i);

45  }
    /*-------------------------------------------------------------------------*/
    void textblink(int val)
    {
     if(val)
```

```
50      {
         printf("%c[5m",escape_val);
        }
       else
        {
55       printf("%c[25m",escape_val);
        }
      }
      /*-------------------------------------------------------------------------*/
      void textreset(void)
60    {
       printf("%c[0m",escape_val);
      }
      /*-------------------------------------------------------------------------*/
      void test_asciiart(void)
65    {
           char c=0;
           int i;
           printf("Testing getch()\n");
           while (c!='q')
70         {
                printf("Press a key (q to quit)....\n");
                c=getch();
                printf("You have entered %c\n",c);
           }
75         c=0;
           printf("Testing getche()\n");
           while (c!='q')
           {
                printf("Press a key (q to quit)....\n");
80              c=getche();
                printf("You have entered %c\n",c);
           }
           printf("Testing colors...\n");
           for (i=0;i<8;i++)
85          {
            textcolor(i);
            printf("============== Color %d\n",i);
           }
           textreset();
90         getch();
      }
      /*=========================================================================*/
      /*              PRIVATE PART                       */
      /*=========================================================================*/
95    /*-------------------------------------------------------------------------*/
      /* changes mode into no-buffered: val=1:no echo */
      void initialisation_clavier(int val )
      {
           struct termios attr;
100        /* store current terminal configuration in sattr */
           if (mode_clavier==0) tcgetattr(STDIN_FILENO,&sattr);
           tcgetattr(STDIN_FILENO,&attr);
           /* now, perform the modifications */
           if (val)
105         attr.c_lflag &= ~ (ICANON);
           else
```

```
        attr.c_lflag &= ~ (ICANON | ECHO);
        attr.c_cc[VMIN]=1;
        attr.c_cc[VTIME]=0;
110     tcsetattr(STDIN_FILENO,TCSAFLUSH,&attr);
        mode_clavier=1;
    }
    /*-------------------------------------------------------------------------*/
    /* Restore terminal configuration */
115 void restauration_clavier(void)
    {
        if (mode_clavier==1)
        tcsetattr(STDIN_FILENO,TCSAFLUSH,&sattr);
        mode_clavier=0;
120 }
    /*-------------------------------------------------------------------------*/
    char getcharacter(void)
    {
        char c;
125     read(STDIN_FILENO, &c,1);
        return c;
    }
```

4.2 Σύνδεσμοι

♢ `http://www.tldp.org`

The Linux Documentation Project
Εδώ θα βρείτε οδηγούς και πληροφορίες για οτιδήποτε σχετίζεται με το λειτουργικό σύστημα Linux. Τα HOW-TO, γραμμένα όλα από εθελοντές λινουξάδες, είναι η σημαντικότερη πηγή εκμάθησης του δημοφιλούς λειτουργικού για τους καινούριους χρήστες του. Βρείτε και διαβάστε στο [1] πολλά παραδείγματα που θα σας βοηθήσουν στα πρώτα προγραμματιστικά σας βήματα με τη βιβλιοθήκη ncurses.

♢ `http://lesouriciergris.free.fr`

Προσωπική ιστοσελίδα του Eric Berthomier
Είναι καθηγητής πληροφορικής, δημιουργός της βιβλιοθήκης asciiart. Για όσους γνωρίζουν γαλλικά, προτείνω να διαβάσουνε και το [5] που περιέχει παραδείγματα και για τις δυο βιβλιοθήκες asciiart και ncurses.

♢ `http://www.tem.uoc.gr/~gbandreo`

Προσωπική ιστοσελίδα του Γιώργου Ανδρέου
Φοιτητής του Τμήματος Εφαρμοσμένων Μαθηματικών της σχολής Θετικών Επιστημών του Πανεπιστημίου Κρήτης. Το περιεχόμενο της σελίδας ανανεώνεται σε άτακτα χρονικά διαστήματα. Εδώ θα βρείτε και νεώτερες εκδόσεις αυτού του κειμένου.

♢ `http://dickey.his.com/ncurses`

Επίσημος δικτυακός τόπος της βιβλιοθήκης ncurses
Ιστορικά στοιχεία, πηγαίος κώδικας, συνήθεις τιθέμενες ερωτήσεις (F.A.Q.), θα τα βρείτε όλα εδώ.

4.3 Αναφορές

[1] *NCURSES Programming HOWTO*, Pradeep Padala, Version 1.7.1, (2002-06-25).

[2] *Programming languages — C*, ISO/IEC 9899:1999, Second edition, (1999-12-01).

[3] *The C Programming Language (ANSI C)*, Brian W. Kernighan, Dennis M. Ritchie, Prentice Hall, 2nd Edition.

[4] *Advanced Bash-Scripting Guide*, Mendel Cooper, Version 1.7, (5 January 2003).

[5] *Cours de C (sous Linux)*, Eric Berthomier, Laurent Signac, (23 April 2003), `http://lesouriciergris.free.fr/linux_c/cours.ps`

[6] *man pages section 3: Curses Library Functions*, Sun Microsystems, (February 2000), `http://lesouriciergris.free.fr/linux_c/curses.zip`

[7] *Practical C Programming*, Steve Oualline, O'Reilly, 3rd Edition.

[8] *man pages*, Linux on-line Documentation System, Version 1.5k.

[9] *Writing programs with ncurses*, Eric S. Raymond and Zeyd M. Ben-Halim, `http://www.cs.mun.ca/~rod/ncurses/ncurses.html`

[10] *OEM Extended ASCII Code*

`http://www.cplusplus.com/doc/papers/ascii.html`

`http://www.cdrummond.qc.ca/cegep/informat/Professeurs/Alain/files/`

`/ascii.htm`