I
Ανασκόπηση και εμβάθυνση
Μερικά πιο “προχωρημένα” θέματα

II
Τα αρχεία δεδομένων στη C

Διδάσκων: Β. Ράπτης

Τελεστές αύξησης/μείωσης (++, --)

Λειτουργία βάσει θέσης:

++x διάφορο του x++ !!!

Παράδειγμα 1 (postfix):

```
int x = 0, y = 1;
x = y++;
```

Ισοδυναμεί με...

```
int x = 0, y = 1;
x = y;
y=y+1;
```

Τελικές τιμές: x = 1, y = 2.

Παράδειγμα 2 (prefix):

```
int x = 0, y = 1;
x =++ y;
```

Ισοδυναμεί με...

```
int x = 0, y = 1;
y=y+1;
x = y;
```

Τελικές τιμές: x = 2, y = 2.

Παρόμοια και για τον τελεστή μείωσης (--)

Τελεστές αύξησης/μείωσης (++, --)

Συνηθίζεται η χρήση τους σε συνδυασμό με άλλες εντολές.

Παραδείγματα:

Ισοδυναμεί με...

```
if(x++ > 0) {
    ...
}
```

```
if(x > 0) {
    ...
}
x=x+1;
```

```
if(++x > 0) {
    ...
}
```

```
x=x+1;
if(x > 0) {
    ...
}
```

Για μεταβολή ≠ 1 υπάρχει μόνο postfix:

x += y;	<=>	x = x+y;	x -= y;	<=>	x = x-y;
x *= y;	<=>	x = x*y;	x /= y;	<=>	x = x/y;

Τελεστές αύξησης/μείωσης (++, --)

Τα διάφορα πρότυπα της C δεν προβλέπουν τα πάντα. Επομένως, υπάρχουν **απροσδιόριστες** περιπτώσεις, με διαφορετική συμπεριφορά για κάθε μεταγλωττιστή.

Συμβουλή: όχι παραφόρτωμα με τελεστές ++ ή --

Προσοχή στις πολύπλοκες εκφράσεις!
Δυσνόητες
Μπορεί να δίνουν άλλο από αυτό που νομίζουμε
Προσοχή σε αριθμοδείκτες πίνακα! Από Kernighan & Pike “The Practice of Programming”:

```
str[i++] = str[i++] = ' ';
```

θέλουμε να δώσουμε την τιμή ' ' *στα δύο επόμενα στοιχεία του* str. *Ανάλογα με το πότε ενημερώνεται το* i, *μπορεί να αφήσει μία θέση και το* i *να αυξηθεί μόνο κατά* 1.
Πιο καλά να σπάσει σε δύο:

```
str[i++] = ' ';
str[i++] = ' ';
```

Άλλο κακό παράδειγμα:

```
array[i++] = i;
```

Μπορεί να αλλάξει πρώτα το i *και μετά να γίνει η ανάθεση ή ανάποδα! Καλύτερα να σπάσει σε δύο, ανάλογα με το τι ακριβώς θέλουμε.*

Σύνθετες Εντολές (if, for, while, do...while)

Οι παρακάτω εντολές:

```
if (...) ;          for ( ...) ;          while ( ... ) ;
```

είναι **συντακτικά σωστές!** Αλλά, 99% λογικά λάθος (όχι αυτό που θέλουμε να κάνουμε)

Ο τελεστής "κόμμα" - παράδειγμα του 1%.

```
for(i=0; i<10; i++)        γράφεται και        for(i=0; i<10; i++, n++);
    n++;
```

Τι κοινό έχουν τα παρακάτω;

```
for(i=1; i >0; )          for( ; ; )          while ( 1 ) (ή κάτι άλλο ≠ 0)
```

Υλοποιούν έναν ατέρμονο βρόχο.

Μπορούμε να φύγουμε με break ή goto

```
if( ... ) break;
if( ... ) goto label;
```

Σύνθετες Εντολές (if, for, while, do...while)

Έξοδος από πολλαπλό βρόχο:

```
for ( i = 0; i < m; i++) {
    for ( j = 0 ; j < n; j ++) {
        ...
        break;
        ...
    }
}
```

Μας πάει από το εσωτερικό στο εξωτερικό for

```
        for ( i = 0; i < m; i++) {
            for ( j = 0 ; j < n; j ++) {
                ...
                goto there;
                ...
            }
        }
there:  printf("It is there!\n");
```

Έτσι βγαίνουμε και από τα δύο for
Ή με κατάλληλο συνδυασμό δύο break

Συναρτήσεις – μερικές παγίδες

Σύνηθες λάθος: αδήλωτο όνομα ακολουθούμενο από παρενθέσεις, π.χ.

y = energy(i); /* κάπου μέσα στο πρόγραμμα */

Ο compiler θα το εκλάβει ως όνομα συνάρτησης τύπου int!
Συνάρτηση χωρίς δήλωση τύπου είναι int και όχι void.

Πρέπει, ένα από τα δύο:

τουλάχιστον το πρότυπο της συνάρτησης να προηγείται της main

Τύπος Ονομα (*τύπος1 παράμετρος1*, *τύπος2 παράμετρος2* ...)

#include το κατάλληλο αρχείο επικεφαλίδας

Άλλη μία πηγή σφαλμάτων:
Αν δεν υπάρχουν παράμετροι πρέπει να γράφουμε τη λέξη void γιατί λόγοι συμβατότητας με παλιότερες εκδόσεις της C οδηγούν την κενή λίστα να εκλαμβάνεται ως δήλωση παλαιού τύπου και αναστέλλεται κάθε έλεγχος των ορισμάτων με αποτέλεσμα αυξημεη πιθανότητα λάθους κατά την ανάπτυξη του προγράμματος.

Σημείωση για το #include:

#include <όνομα.h> /* αρχείο πρότυπης βιβλιοθήκης */
#include"όνομα.h" /* αρχείο που δημιουργήσαμε εμείς */

Συναρτήσεις – κατάταξη μεταβλητών

Ως προς την εμβέλεια.
- Τοπικές: δηλωμένες μέσα στη συνάρτηση
  (*και... ακόμη πιο τοπικές, μέσα σε { και }*)
  "Ορατές" μόνο μέσα στη συνάρτηση (ή τα άγκιστρα).
  Χάνουν την τιμή τους με την έξοδο από την εμβέλειά τους
- Εξωτερικές: δηλωμένες έξω από συναρτήσεις.
  "Ορατές" από όλες τις συναρτήσεις που ακολουθούν.
  Διατηρούν την τιμή τους από συνάρτηση σε συνάρτηση

ΑΛΛΑ, τοπική μεταβλητή με ίδιο όνομα υπερβαίνει (overrides) την εξωτερική.

Ως προς την κλάση αποθήκευσης.
- Αυτόματες, (auto, προεπιλογή για όλες, άρα δε χρειάζεται αυτή η δήλωση)
- Στατικές (static).
  Διατηρούν την τιμή τους ακόμη και αν είναι τοπικές.
  Αόρατες από άλλα αρχεία.
- Καταχωρητή (register)
  Στο μικροεπεξεργαστή για ταχύτερη πρόσβαση και επεξεργασία

<u>Προσοχή!</u> Τοπικές αυτόματες πριν την αρχικοποίηση περιέχουν "σκουπίδια"
Στατικές και εξωτερικές αρχικά είναι μηδέν.

Συναρτήσεις – πέρασμα παραμέτρων

Πέρασμα μέσω τιμής (by value) – αν δε θέλουμε να αλλάξουμε τα ορίσματα

```
int absdiff(int x, int y) {
        if(x > y)
                return(x-y);
        else
                return(y-x);
}
```

```
int x, m = 3, n = 2;
...
x = absdiff(m,n);
printf("m=%d, n=%d, |m-n| =%d", m, n, x);
...
```

Θα τυπώσει: m=3, n=2, |m-n|=1

Πέρασμα με διεύθυνση ή αναφορά (by reference) – για να αλλάξουμε τα ορίσματα

```
int altadd(int *x, int *y) {
        (*x)++;
        (*y)--;
        return(*x+*y);
}
```

```
int x, m = 3, n = 2;
...
printf("m=%d, n=%d\n);
x = altadd(&m, &n);
printf("m=%d, n=%d, m+n=%d\n",m,n,x);
```

Θα τυπώσει: m=3, n=2
m=4, n=1, m+n=5

Επιτρέπεται κάθε παράμετρος να περνά με διαφορετικό τρόπο.

Συναρτήσεις – πέρασμα παραμέτρων

1. Επιτρέπεται κάθε παράμετρος να περνά με άλλο τρόπο.

```
void func(int x, int *y);          func(a, &b);
```

2. Οι πίνακες περνάνε μέσω διεύθυνσης
   (όνομα πίνακα συνώνυμο με δείκτη στο στοιχείο [0])

```
void func(int *);
...
int x[10]={1};                     func(x);
```

3. Και... συναρτήσεις περνάνε μέσω διεύθυνσης
   (όνομα συνάρτησης συνώνυμο με δείκτη σε αυτή)
   και έτσι, πάμε στους δείκτες σε συναρτήσεις...

Δείκτες σε συναρτήσεις

Κάθε συνάρτηση έχει διεύθυνση (την αρχή της περιοχής όπου αποθηκεύεται).
Η διεύθυνση μπορεί να αποθηκευτεί.
Άρα, μπορεί να οριστεί δείκτης σε συνάρτηση

Τύπος (**όνομα_δείκτη*) ( );

Παρενθέσεις απαραίτητες λόγω προτεραιότητας (αλλιώς θα οριζόταν συνάρτηση που επιστρέφει *Τύπος**)

Όπως και με τους πίνακες: το όνομα της συνάρτησης είναι συνώνυμο με τη διεύθυνσή της.

Έστω η συνάρτηση

```
int  intFc(int a, int b, int (*Func) ( ));
```

Βάσει προηγουμένων, μπορώ να περάσω συναρτήσεις ως το τρίτο όρισμα, π.χ.

```
intFc(x, y, someFunction);
intFc(a, b, someOtherFunction);
```

Επιτρέπονται και πίνακες δεικτών σε συναρτήσεις:

```
void (*fptr[ ]) = {func1, func2, func3};
```

Διάφορα

Εντολές πολλαπλής ανάθεσης.
Η ανάθεση x = y; είναι και παράσταση με τιμή την τελική τιμή του x.
Τότε, μπορούμε να γράψουμε

```
x = y = z = a;
```

γιατί σημαίνει

```
(x = (y = (z = a)));
```

Μετατροπή τύπου (type casting)

```
τύπος1 A;
τύπος2 B;
A = (τύπος1) B;
```

Παράδειγμα: πραγματική διαίρεση ακεραίων.

```
int m = 2, n = 3;
float x;
x = ( (float) m ) / ((float) n);
```

Παραστάσεις ως λογικές συνθήκες:
Ο,τι επιστρέφει 0 (int, long κλπ), 0. (float, double κλπ), NULL ή '\0' = ΨΕΥΔΗΣ
Διαφορετικά, ΑΛΗΘΗΣ

Διάφορα

Είσοδος/Έξοδος για ένα χαρακτήρα

putchar γνωστή...

getchar η συμμετρική αυτής.
(διαβάζει από πληκτρολόγιο)

"Πάγωμα" προγράμματος

```
printf("\nPress any key...") /* κάποιο μήνυμα */
getchar( );     /* περιμένει οποιοδήποτε χαρακτήρα */
```

Ορίσματα της main

```
int main (int argc, char **argv)
ή int main(int argc, char *argv[ ])
```

argc >= 1
argv[0] = όνομα εκτελέσιμου
argv[1], argv[2], ... argv[argc+1] = ορίσματα γραμμής εντολών

Επιστρεφόμενη τιμή προς το σύστημα· εξ ου και return 0; στο τέλος.

Προεπεξεργαστής

#include γνωστή...
#define, #ifdef, #ifndef, #undef, #endif χρήσιμες!

#define LALA 25 όπου υπάρχει LALA στο πρόγραμμα, αντικαθίσταται από 25, *πριν* τη μεταγλώττιση.

#define SIZE(x) sizeof(x) / sizeof(x[0])
χρήσιμη μακροεντολή για μέγεθος πίνακα

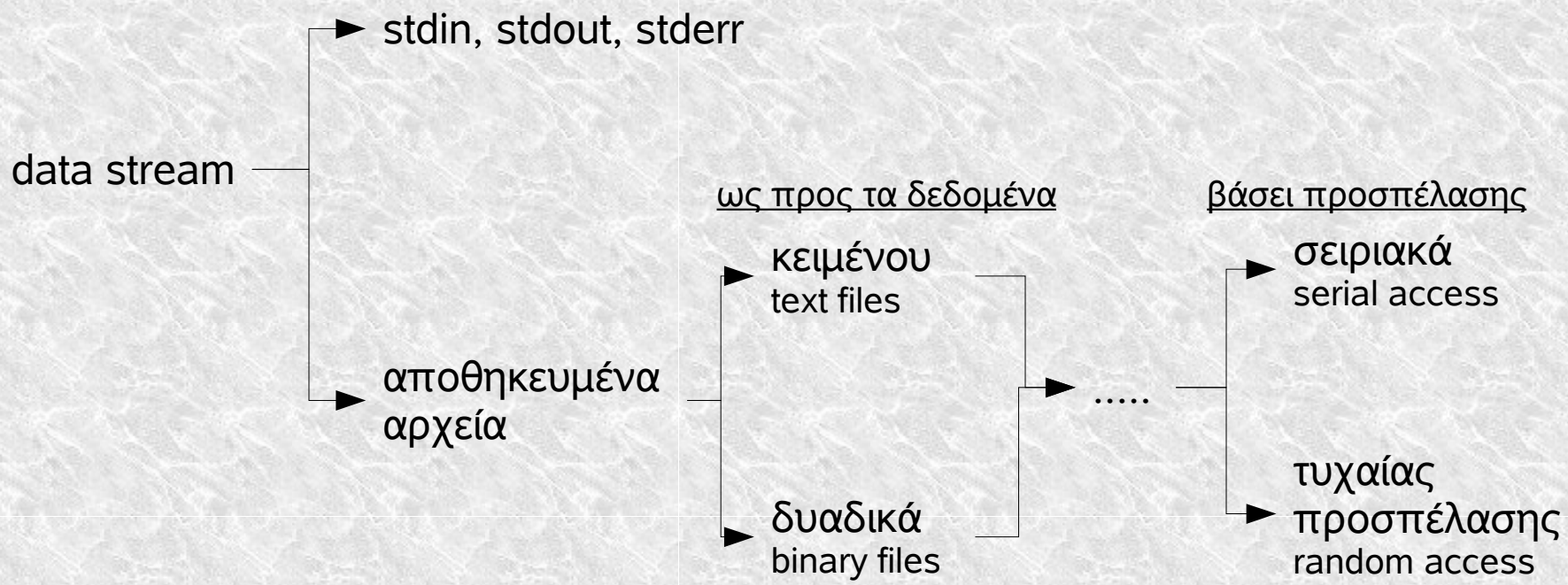
Έστω αρχείο something.h

```
#ifndef SOMETHING_H    (αν δεν έχει οριστεί αυτή η συμβολική σταθερά)
#define SOMETHING_H    (όρισέ την)
...                    (δήλωσε τις συναρτήσεις που θέλεις κλπ)
#endif                 (τέλος του if)
```

για να αποφύγουμε πολλαπλά include του ίδιου αρχείου σε μεγάλους κώδικες με πολλά αρχεία.
Αν είναι ήδη defined η συμβολική σταθερά, τότε δε θα διαβάσει το υπόλοιπο αρχείο.

Αρχεία

Το γενικό πλαίσιο: data stream (ρεύμα δεδομένων).



Αρχεία κειμένου: δεδομένα τύπου char
Δυαδικά: δεδομένα όπως ακριβώς αναπαρίστανται στη μνήμη του ΗΥ
Τρόπος προσπέλασης:

Αρχεία

Τι είναι αρχείο;
“Λογικά οργανωμένη πληροφορία αποθηκευμένη σε μαγνητικά μέσα”.
Ουσιαστικά, bytes οργανωμένα με κάποιο τρόπο. Ποιον ακριβώς;

Πιθανά περιεχόμενα ενός “αρχείου κειμένου”: bytes χωρισμένα από '\n'

'K'	'o'	'z'	'a'	'n'	'i'	'\n'	'l'	'o'	'a'	'n'	'n'	'i'	'n'	'a'	'\n'	'A'	't'	'h'	'i'	'n'	'a'	'\n'	'T'	'h'	'e'

Μπορούμε να τα δούμε και έτσι:

'K'	'o'	'z'	'a'	'n'	'i'	'\n'						
'l'	'o'	'a'	'n'	'n'	'i'	'n'	'a'	'\n'				
'A'	't'	'h'	'i'	'n'	'a'	'\n'						
'T'	'h'	'e'	's'	's'	'a'	'l'	'o'	'n'	'i'	'k'	'i'	EOF

Γραμμές = “εγγραφές”.
Ξεχωρίζουν με '\n'
Τελευταίο byte: EOF = End Of File

Από το stdio.h

```
/* End of file character.
   Some things throughout the
library rely on this being -1.  */
#ifndef EOF
# define EOF (-1)
#endif
```

Αρχεία

Πώς θα μπορούσαμε να χειριστούμε το προηγούμενο αρχείο;
Τύπος FILE - τα αρχεία ορίζονται ως δείκτες, π.χ.

```
FILE *f1, *f2, f3;
```

Άνοιγμα αρχείων.

```
f1 = fopen("arxeio1", "r");
f2 = fopen("arxeio2", "w");
f3 = fopen("arxeio3", "a");
```

"read", για ανάγνωση

"write", για εγγραφή (αν προϋπάρχει, **σβήνονται** τα προηγούμενα!)

"append", για προσθήκη δεδομένων *χωρίς να σβήνει τα υπάρχοντα*.

Προκαθορισμένος τύπος: κειμένο.
Για δυαδικά: κωδικός "b", π.χ. fp = fopen("arxeio", "wb");

Κλείσιμο αρχείων: int fclose (FILE *);
π.χ. fclose(f1) κλπ

<u>Σημείωση:</u> stdin, stdout = *σταθερές* τύπου FILE*

Αρχεία

Ανάγνωση και εγγραφή χαρακτήρων.

```
int fgetc(FILE* f)
int getc (FILE* f)
```

```
int getchar( void )
```

getchar( ) <=> fgetc(stdin)

```
int fputc(int c, FILE *f);
int putc (int c, FILE *f);
```

```
int putchar(int c);
```

putchar (c) <=> fputc(c, stdin);

Παραδείγματα:

```
FILE *fp=fopen("file1", "r");
...
while((c = fgetc(fp)) != EOF)
    putchar(c);
...
fclose(fp);
```

```
FILE *gp=fopen("file2", "w");
...
while((c = getchar( )) != '\n')
    fputc(c, gp);
...
fclose(gp);
```

Αρχεία

Ανάγνωση και εγγραφή αλφαριθμητικών.

```
int fprintf(FILE* f, const char *format, ...);      int printf(const char*format, ...);
int fscanf(FILE* f, const char *format, ...);       int scanf(const char *format, ...);
```

Το format επιτρέπει καταχώρηση εγγραφών *σταθερού* μήκους.

Εγγραφές ποικίλου μήκους με τερματικό byte '\n' → αναγκαστικά, σειριακά
Εγγραφές σταθερού μήκους → επιτρέπουν άμεση προσπέλαση κάθε εγγραφής
Πώς;

```
int fseek(FILE *fp, long int offset, int flag);
```

μετατοπίζει "κεφαλή" κατά offset *bytes*
όπου flag

```
#define SEEK_SET    0        (σε σχέση με την αρχή του αρχείου)
#define SEEK_CUR    1        (σε σχέση με τρέχουσα θέση)
#define SEEK_END    2        (σε σχέση με το τέλος του αρχείου)
```

Επίσης,

```
int fputs(const char*s, FILE *f);          int puts(const char *s);
char *fgets(char*s, int n, FILE *f);       char *gets(char*s);
```