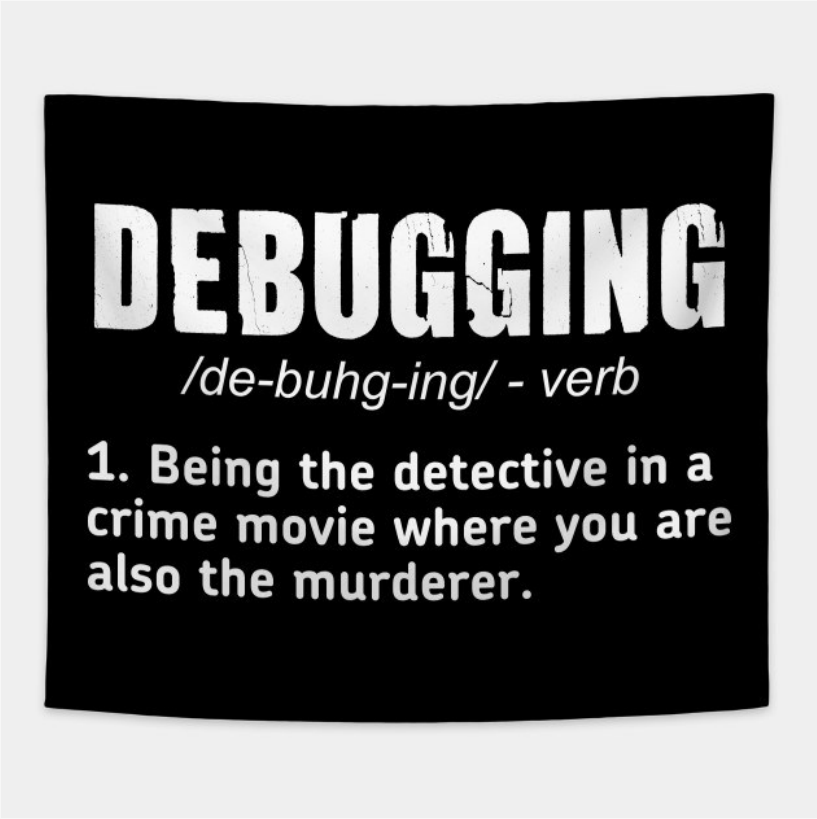
Debugging

- bug = λάθος στο πρόγραμμα το οποίο προκαλεί απρόσμενη συμπεριφορά ή λανθασμένη έξοδο.
- debugging = η διαδικασία εύρεσης ενός bug.



Τεχνικές αποφυγής σφαλμάτων

- Σχεδιάζουμε τον αλγόριθμο πριν αρχίσουμε να γράφουμε κώδικα
- Κατανοούμε τι θέλουμε να κάνει το πρόγραμμα
- Σχεδιάζουμε πώς θα πάμε από την είσοδο στην έξοδο.
- Βεβαιωνόμαστε ότι η μέθοδός μας θα λειτουργήσει για διάφορες περιπτώσεις
- Ιδιαίτερη έμφαση στις οριακές περιπτώσεις.

Τεχνικές αποφυγής σφαλμάτων

- Γράφουμε ευανάγνωστο κώδικα
- Σωστή στοίχιση, με συνεπή χρήση { }
- Αν κάνουμε αλλαγές στον κώδικα που επηρεάζουν τη στοίχιση, τη διορθώνουμε άμεσα
- Περιγραφικά ονόματα μεταβλητών/συναρτήσεων
- Συνετή χρήση κενών γραμμών
- Αποτελεσματικός σχολιασμός.

Τεχνικές αποφυγής σφαλμάτων

- Αναπτύσσουμε το πρόγραμμα σταδιακά.
- Δε γράφουμε όλη τη λύση από την αρχή, αλλά κομμάτι-κομμάτι
- Δεν προχωράμε στο επόμενο κομμάτι αν δεν είμαστε σίγουροι ότι τα προηγούμενα λειτουργούν σωστά.

Δυσκολίες

- Το σημείο στο οποίο τερμάτισε άδοξα το πρόγραμμα δεν είναι απαραίτητα το σημείο στο οποίο έγινε το λάθος
- Το πρόγραμμα κάποιες φορές φαίνεται να λειτουργεί σωστά, κάποιες όχι.
- Το πρόβλημα μπορεί να προέρχεται από συνδυασμό λαθών.

Τεχνικές εύρεσης σφαλμάτων

- Χρήση debugger
- Χρήση εργαλείων εξέτασης της μνήμης που χρησιμοποιεί το πρόγραμμα.
- Έλεγχος και αποκλεισμός επιμέρους κομματιών του προγράμματος έως ότου εντοπιστεί το κομμάτι στο οποίο βρίσκεται το σφάλμα.

Χρήση debugger

- debugger = εργαλείο που μας βοηθά να λύσουμε τα παρακάτω προβλήματα:
- Το πρόγραμμα τρέχει και τερματίζει ομαλά αλλά δεν παράγει σωστά αποτελέσματα
- Ο debugger μας επιτρέπει να εκτελέσουμε το πρόγραμμα γραμμή-γραμμή, εξετάζοντας τις τιμές των μεταβλητών που μας ενδιαφέρουν.
- Το πρόγραμμα τερματίζει με segmentation fault ή παρόμοιο λάθος.
- Ο debugger μας βοηθά να εντοπίσουμε ακριβώς το σημείο που προέκυψε το segmentation fault.

gdb

- Βασική προϋπόθεση
- -g όταν κάνουμε compile με gcc
- Παράδειγμα: `gcc -Wall -g test.c -o test`
- Απλή εκτέλεση:
- Γράφουμε `gdb test` όπου test το εκτελέσιμο
- Αγνοούμε τα εισαγωγικά μηνύματα μέχρι να δούμε:
- `Reading symbols from test...done`
- Γράφουμε `r` για να εκτελεστεί το πρόγραμμα μέσα από το περιβάλλον του gdb (r=run)
- Βλέπουμε το τελικό αποτέλεσμα.

Απλή εκτέλεση

```
vdoufexi@csl-venus:~$ gcc -Wall -g test.c -o test
vdoufexi@csl-venus:~$ gdb test
GNU gdb (Ubuntu 8.1-0ubuntu3.1) 8.1.0.201
Copyright (C) 2018 Free Software Foundati
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos w                ommands related to "word"...
Reading symbols from test...done.
(gdb) r
Starting program: /srv/
Enter n, k: 10 4
Result = 18.33
[Inferior 1 (process 3573) exited normally]
(gdb) q
vdoufexi@csl-venus:~$
```

Compile με -g

Εκκίνηση gdb για το εκτελέσιμο test

Εκτέλεση του test

Έξοδος προγράμματος (τα 10 4 εισήχθησαν από το χρήστη)

Το πρόγραμμα τερμάτισε ομαλά.

Έξοδος από gdb

Ειδικές περιπτώσεις εισόδου

- Εκτέλεση προγράμματος με ανακατεύθυνση της συμβατικής εισόδου:
- `r < a_in1` όπου a_in1 το αρχείο εισόδου.
- Εκτέλεση με ορίσματα προγράμματος από τη γραμμή εντολών:
- `r 15 hello` όπου 15 και hello τα ορίσματα προγράμματος.
- Εκτέλεση με ορίσματα και ανακατεύθυνση
- `r 15 hello < a_in1`

Διαχείριση segmentation fault

- Εκτελούμε κανονικά το πρόγραμμα μέσω gdb έως ότου τερματίσει με segmentation (ή άλλο) fault.
- Γράφουμε `bt` (backtrace) για να δούμε τα frames (πλαίσια στοίβας) μέχρι το σημείο που τερμάτισε το πρόγραμμα.
- Μας ενδιαφέρει το πιο πρόσφατο frame (κοιτώντας τα από το τέλος προς την αρχή) που αναφέρεται σε δικό μας κώδικα και όχι κώδικα βιβλιοθήκης.
- Αν χρειάζεται, γράφουμε `f X` όπου `X` ο αριθμός του frame για να μεταφερθούμε σε αυτό.

Παράδειγμα

```c
/*
Ypologizei to athroisma:
   n/1 + n/2 + n/3 + n/4 + ... + n/k
*/
#include<stdio.h>

double calc_term(int n, int m);
double calc_sum(int n, int k);

int main (int argc, char *argv[]) {
    int n, k;
    printf("Enter n, k: ");
    scanf("%d %d", &n, &k);
    double result = calc_sum(n, k);
    printf("Result = %.2lf\n", result);
    return 0;
}
```

```c
double calc_sum(int n, int k) {
    int i;
    double sum=0;
    for (i = 0; i < k; i++) {
        sum += calc_term(n, i);
    }
    return sum;
}

double calc_term(int n, int i) {
    double result;
    result = n/i;
    return result;
}
```

```
vdoufexi@csl-venus:~$ gcc -g -Wall test.c -o test
vdoufexi@csl-venus:~$ ./test
Enter n, k: 10 4
Floating point exception (core dumped)
vdoufexi@csl-venus:~$
```

Παράδειγμα

```
Reading symbols from test...done.
(gdb) r
Starting program: /srv/homes/vdoufexi/test
Enter n, k: 10 4

Program received signal SIGFPE, Arithmetic exception.
0x000055555554802 in calc_term (n=10, i=0) at test.c:30
30                result = n/i;
(gdb) bt
#0  0x000055555554802 in calc_term (n=10, i=0) at test.c:30
#1  0x00005555555547cf in calc_sum (n=10, k=4) at test.c:23
#2  0x00055555554774 in main (argc=1, argv=0x7fffffffd478) at test.c:15
(gdb)
```

Παρατηρήστε τη σειρά των κλήσεων:

main -> calc_sum -> calc_term -> crash

frame: #2 #1 #0

Μας ενδιαφέρει το frame #0

Παράδειγμα

- Έχοντας μεταβεί στο κατάλληλο frame, μπορούμε να δούμε πληροφορίες για τις εμπλεκόμενες μεταβλητές
- `p expr` όπου expr μια έκφραση, εκτυπώνει την τιμή της (p=print).

```
(gdb) f 0
#0  0x000055555554802 in calc_term (n=10, i=0) at test.c:30
30                result = n/i;
(gdb) p n
$3 = 10
(gdb) p i
$4 = 0
(gdb) p n/i
Division by zero
(gdb)
```

Εκτέλεση γραμμή-γραμμή

- Αν δε μπορούμε να βρούμε εύκολα ποιο είναι το πρόβλημα, μπορούμε να εκτελέσουμε το πρόγραμμά μας γραμμή-γραμμή και να εξετάζουμε σε κάθε γραμμή τις τιμές των μεταβλητών, σκεφτόμενοι ταυτόχρονα τι τιμές **θα έπρεπε** να έχουν, κι ελέγχοντας αν όντως τις έχουν.
- Για να το κάνουμε αυτό χρησιμοποιούμε breakpoints.
- breakpoint = σημείο στο πρόγραμμα που θέλουμε να σταματήσει η εκτέλεση και να περιμένει μέχρι να του πούμε εμείς να συνεχίσει.

Εκτέλεση γραμμή-γραμμή

- Η σύνταξη είναι `b s` όπου s το σημείο που θέλουμε να σταματήσει η εκτέλεση (b=break). Έχουμε αρκετές επιλογές για το s:
- `b main` σταματάει στην αρχή της main, πριν την πρώτη εντολή.
- `b 32` σταματάει στη γραμμή 32
- `b myfunc` σταματάει στην αρχή της συνάρτησης myfunc, πριν την πρώτη εντολή.
- `b test.c:32` σταματάει στη γραμμή 32 του αρχείου test.c (χρήσιμο αν το πρόγραμμά μας καταλαμβάνει περισσότερα από ένα αρχεία)

Εκτέλεση γραμμή-γραμμή

- Αφού σταματήσουμε σε μια εντολή, έχουμε δύο τρόπους να συνεχίσουμε στην επόμενη:
- `n` (δηλαδή next).
- Αν η επόμενη εντολή περιλαμβάνει κλήση σε συνάρτηση, τότε οι εντολές της συνάρτησης θα εκτελεστούν σε ένα βήμα και θα προχωρήσει στην εντολή που ακολουθεί.
- `s` (δηλαδή step).
- Αν η επόμενη εντολή έχει κλήση συνάρτησης, θα μας πάει στην πρώτη εντολή μέσα στη συνάρτηση.
- Σε κάθε περίπτωση, μετά από n ή s το gdb περιμένει ξανά μέχρι να του πούμε να συνεχίσει.

Εκτέλεση γραμμή-γραμμή

```
(gdb) b main
Breakpoint 1 at 0x729: file test.c, line 11.
(gdb) r
Starting program: /srv/homes/vdoufexi/test

Breakpoint 1, main (argc=1, argv=0x7fffffffd4a8) at test.c:11
warning: Source file is more recent than executable.
11              int n, k;
(gdb) n
13              scanf("%d %d", &n, &k);
(gdb)
```

ΠΡΟΣΟΧΗ: Η τελευταία εντολή που βλέπουμε (η scanf στη συγκεκριμένη περίπτωση) ΔΕΝ έχει εκτελεστεί ακόμη, αλλά είναι η επόμενη εντολή προς εκτέλεση (αν δοθεί n ή s).

Εκτέλεση γραμμή-γραμμή

```
13              scanf("%d %d", &n, &k);
(gdb)
Enter n, k: 10 4
14              double result = calc_sum(n, k);
(gdb)
```

Αν εδώ γράψουμε **n**, θα κληθεί η calc_sum, αν δεν συμβεί κάποιο σφάλμα θα αποθηκευτεί το αποτέλεσμά της στη μεταβλητή result,
και θα είμαστε έτοιμοι να προχωρήσουμε στην εντολή της γραμμής 15.
Με άλλα λόγια, η εκτέλεση της calc_sum αντιμετωπίζεται ως ένα βήμα στην εκτέλεση βήμα-βήμα που κάνουμε.

Αν όμως γράψουμε **s**, θα μεταβούμε στην πρώτη γραμμή της συνάρτησης calc_sum και αυτή θα είναι η επόμενη εντολή προς εκτέλεση. Με άλλα λόγια, κάθε εντολή της calc_sum αντιμετωπίζεται ως ένα βήμα στην εκτέλεση βήμα-βήμα που κάνουμε.

Εκτέλεση γραμμή-γραμμή

Παράδειγμα χρήσης n και s σε test.c χωρίς bugs

```
14              double result = calc_sum(n, k);
(gdb) s
calc_sum (n=10, k=4) at test.c:21
21              double sum=0;
(gdb) n
22              for (i = 1; i < k; i++) {
(gdb) n
23                      sum += calc_term(n, i);
(gdb) p sum
$4 = 0
(gdb) n
22              for (i = 1; i < k; i++) {
(gdb) p sum
$5 = 10
(gdb)
```

Με s "μπαίνουμε" στη συνάρτηση calc_sum. Η επόμενη εντολή προς εκτέλεση είναι η πρώτη εντολή της calc_sum στη γραμμή 21.

Είμαστε ακριβώς πριν την εκτέλεση της γραμμής 23 κι εκτυπώνουμε την τιμή του sum.

Με n εκτελέστηκε η γραμμή 23 σε ένα βήμα και προχωρήσαμε άμεσα στην επόμενη γραμμή. Το sum έχει πάρει νέα τιμή.

Ακολουθεί το ίδιο παράδειγμα αλλά με χρήση s στη γραμμή 23

Εκτέλεση γραμμή-γραμμή

```
14                  double result = calc_sum(n, k);
(gdb) s
calc_sum (n=10, k=4) at test.c:21
21                  double sum=0;
(gdb) n
22                  for (i = 1; i < k; i++) {
(gdb) n
23                          sum += calc_term(n, i);
(gdb) s
calc_term (n=10, i=1) at test.c:30
30                  result = (double)n/i;
(gdb) n
31                  return result;
(gdb) n
32          }
(gdb) n
calc_sum (n=10, k=4) at test.c:22
22                  for (i = 1; i < k; i++) {
(gdb) n
23                          sum += calc_term(n, i);
(gdb) n
22                  for (i = 1; i < k; i++) {
(gdb)
```

Με s, μεταβήκαμε στην αρχή της calc_sum (γρ. 21 του αρχείου).

Με s, μεταβήκαμε στην αρχή της calc_term (γρ. 30 του αρχείου).

Τέλος της calc_term επιστρέφουμε στη επόμενη εντολή του προγράμματος εντός της calc_sum.

Αυτή τη φορά επιλέξαμε n επομένως μεταβήκαμε

Άλλες χρήσιμες εντολές

- Η εντολή `c` μπορεί να χρησιμοποιηθεί αφότου έχουμε σταματήσει σε κάποιο σημείο, και ζητά από το gdb να συνεχίσει την εκτέλεση μέχρι είτε να συναντήσει το επόμενο breakpoint είτε να τερματίσει το πρόγραμμα (c = continue).
- Η εντολή `l` (μικρό el) μας δείχνει τις 5 γραμμές πριν και τις 5 γραμμές μετά την τρέχουσα εντολή, για να εξετάσουμε πιο συνολικά το σημείο του προγράμματος στο οποίο βρισκόμαστε (l = list).

Άλλες χρήσιμες εντολές

- Με την εντολή `watch var` ζητάμε από το gdb να παρακολουθήσει τη μεταβλητή με όνομα var και να διακόψει την εκτέλεση όταν αυτή αλλάξει τιμή.
- Με την continue μπορεί να ξανασυνεχίσει.
- Με την εντολή `q` τερματίζουμε το gdb. Αν το πρόγραμμα που έτρεχε μέσω gdb δεν είχε τερματίσει ομαλά, θα ζητά επιβεβαίωση για την έξοδο κι επιλέγουμε y για ναι, ή n για όχι (q = quit).

gdb vs. printf

- Μια απλή τακτική για debugging είναι να εισάγετε printf σε στρατηγικά σημεία του κώδικα και να εκτιμάτε το σημείο που πρέκυψε segmentation fault με βάση το ποιες printf εκτελέστηκαν.
- Αυτό δεν είναι αξιόπιστο, γιατί οι εκτυπώσεις από την printf δεν είναι απαραίτητα άμεσες.
- Μπορεί να έχει εκτελεστεί μια printf από το πρόγραμμα αλλά το λειτουργικό σύστημα να μην έχει ακόμη εμφανίσει το μήνυμα στην οθόνη.
- Να τελειώνετε **πάντα** την εκτύπωση με \n και μετά την printf προσθέστε την εντολή fflush(stdout);