ΕΡΓΑΣΤΗΡΙΟ 4:
Μεταβλητές, Δομές Ελέγχου και Επανάληψης

Στο εργαστήριο αυτό, θα εξοικειωθούμε με τους τύπους δεδομένων που μας παρέχει η γλώσσα C, θα χρησιμοποιήσουμε τις δομές επανάληψης (`for`, `while`, `do...while`), την δομή ελέγχου (`if...else`) και θα μάθουμε πώς να διαχειριζόμαστε την έξοδο του προγράμματος μας, μέσω της συνάρτησης `printf()`. Επίσης θα δούμε κάποια βασικά πράγματα για τις συναρτήσεις

Άσκηση 1: Υπολογισμός σειράς

1.1. Γράψτε πρόγραμμα C που να υπολογίζει το άθροισμα της σειράς

$$\sum_{i=1}^{100} i$$

και να το εκτυπώνει στην οθόνη. Να χρησιμοποιηθεί η δομή επανάληψης `do...while`.

Δομές επανάληψης `while` και `do...while`:

WHILE	DO...WHILE

`while (E2)`
`{`
`......`
`}`
	`do`
	`{`
	`......`
	`} while (E2);`

E_2: Συνθήκη Εισόδου και Συνέχισης Επανάληψης	E_2: Συνθήκη Συνέχισης Επανάληψης

Η δομή επανάληψης `for`:

FOR	Όπου

`for(E1; E2; E3)`
`{`
`............`
`}`
	E_1: Εντολή Αρχικοποίησης
	E_2: Συνθήκη Εισόδου και Συνέχισης Επανάληψης (εφόσον ισχύει, εισερχόμαστε στον βρόχο ή επαναλαμβάνεται ο βρόχος)
	E_3: Εντολή Επανάληψης (εκτελείται στο τέλος της κάθε επανάληψης και πριν αρχίσει η επόμενη)

Ισοδύναμη δομή `while` με την δομή `for`:

FOR	WHILE

`for(E1; E2; E3)`
`{`
`............`
`}`
	`E1;`
	`while (E2)`
	`{`
	`......`
	`E3;`
	`}`

Τύποι Δεδομένων

`int` (ακέραιος, συνήθως με 4 bytes)
`long` (ακέραιος, τουλάχιστον με 4 bytes)
`long long` (ακέραιος, τουλάχιστον με 8 bytes)

(βλέπε και σημειώσεις μαθήματος, σελ. 31)

Η συνάρτηση `printf()` μπορεί να εκτυπώσει την τιμή μεταβλητών που δέχεται σαν όρισμα.

Σύνταξη: `printf("%y", var);`
όπου `y` είναι σύμβολο που αντιστοιχεί στον τύπο δεδομένων της μεταβλητής `var`.

`d:`	int	`f:`	float ή double
`ld:`	long	`Lf:`	long double
`lld:`	long long		
`u:`	unsigned int		
`lu:`	unsigned long		
`llu:`	unsigned long long		

1.2. Τροποποιήστε το πρόγραμμά σας, ώστε να υπολογίζει το άθροισμα:

$$\sum_{i=1}^{100} 1/i^2$$

και να το εκτυπώνει στην οθόνη.

Υπόδειξη: Χρησιμοποιείστε μία ενδιάμεση μεταβλητή για να αποθηκεύετε προσωρινά τον τρέχοντα όρο της σειράς.

Τύποι Δεδομένων

`float` (πραγματικός αριθμός, συνήθως με 4 bytes)
`double` (πραγματικός αριθμός, συνήθως με 8 bytes)

1.3. Αν γνωρίζετε ότι

$$\pi = \sqrt{6\sum_{i=1}^{+\infty} 1/i^2}$$

τροποποιήστε το πρόγραμμα σας, ώστε να υπολογίσετε το π μέσω των πρώτων 100 όρων της σειράς.

`double sqrt(double)`: Επιστρέφει την τετραγωνική ρίζα του αριθμού που δέχεται σαν όρισμα.

Απαιτείται η ενσωμάτωση του αρχείου επικεφαλίδας `math.h`.
Στην μεταγλώττιση με τον `gcc` απαιτείται και η επιλογή `-lm` για να συμπεριληφθεί η μαθηματική βιβλιοθήκη.

1.4. Παρατηρήστε ότι οι όροι που προστίθενται στην σειρά ολοένα και γίνονται πιο μικροί, με αποτέλεσμα από ένα σημείο και μετά να είναι αρκούντως μικροί για να μην επηρεάζουν το αποτέλεσμα, αφού χρησιμοποιούμε αριθμητική πεπερασμένης ακρίβειας. Αλλάξτε το κριτήριο τερματισμού του υπολογισμού, ώστε ο υπολογισμός να σταματάει όταν ο τρέχων όρος που προστίθεται στην σειρά είναι μικρότερος από 10^{-15}.

1.5. Τροποποιήστε το πρόγραμμά σας, ώστε να εκτυπώνεται στην οθόνη το αποτέλεσμα με ακρίβεια 8 δεκαδικών ψηφίων.

Η συνάρτηση `printf()` μπορεί να εκτυπώσει την τιμή πραγματικών μεταβλητών που δέχεται σαν όρισμα με επιθυμητή μορφοποίηση.

Σύνταξη: `printf("%a.bf", var);`

`a`: Ορίζουμε το πλήθος των θέσεων που θα γίνει η εκτύπωση.
`b`: Ορίζουμε το πλήθος των ψηφίων μετά την υποδιαστολή που θα εκτυπωθούν.

1.6 Τροποποιήστε το πρόγραμμα σας, ώστε να υπολογίζει την σειρά:

$$S_1 = \frac{1}{1^2} - \frac{1}{2^2} + \frac{1}{3^2} - \frac{1}{4^2} + \frac{1}{5^2} - \frac{1}{6^2} + \ldots$$

1.7 Επαναλάβετε και για τις σειρές που εμφανίζονται στις σημειώσεις του μαθήματος, σελ. 23.

Άσκηση 2: Το πρόβλημα collatz

Ορισμός Συνάρτησης (δείτε και σημειώσεις μαθήματος, σελ. 59-61):

```
<Τύπος Επιστροφής> <Όνομα Συνάρτησης>(<Τυπικές Παράμετροι>)
{
    <Εντολές και Δηλώσεις>
}
```

`<Τύπος Επιστροφής>`
 	
Τύπος δεδομένων της τιμής που επιστρέφεται από τη συνάρτηση μέσω της εντολής:
 	
`return <παράσταση>;`
 	
Σε περίπτωση μη επιστροφής τιμής, ο `<Τύπος Επιστροφής>` ορίζεται σαν `void`.
	`<Τυπικές Παράμετροι>`
	
	Μεταβλητές, μαζί με τους τύπους τους, που χρησιμοποιούνται στη συνάρτηση, χωρισμένες με κόμμα, στις οποίες δίνονται τιμές από την καλούσα συνάρτηση.
		Προαναγγελία πρωτότυπου συνάρτησης, όταν καλείται πριν ορισθεί
		
		`<Τύπος Επιστροφής> <Όνομα Συνάρτησης>(<Τυπικές Παράμετροι>);`
		
		`int main(void)`
		`{`
		`  ....`
		`}`
		
		`<Τύπος Επιστροφής> <Όνομα Συνάρτησης>(<Τυπικές Παράμετροι>)`
		`{`
		`  ....`
		`}`

2.1 Κατασκευάστε τη συνάρτηση `unsigned long long collatz(unsigned long long n)` που δέχεται σαν όρισμα έναν φυσικό αριθμό και επιστρέφει:

- Αν ο n είναι περιττός το 3n+1
- Αν ο n είναι άρτιος το n/2

Η συνάρτηση αυτή ουσιαστικά επιστρέφει τον επόμενο όρο της ακολουθίας collatz.

Η δομή ελέγχου if ... else

```
if (C)
   E1
else
   E2
```

Εάν η συνθήκη C είναι αληθής, τότε εκτελείται η εντολή (ή μπλοκ εντολών) E_1, αλλιώς η εντολή (ή block εντολών) E_2

2.2 Κατασκευάστε τη συνάρτηση `unsigned int collatz_len(unsigned long long n)` που χρησιμοποιώντας την συνάρτηση `collatz` να βρίσκει και να εκτυπώνει όλους τους όρους της ακολουθίας collatz[1] μέχρι να συναντήσει τον όρο με τιμή 1.
Όταν αυτό συμβεί να επιστρέφει το πλήθος των όρων της ακολουθίας από το n μέχρι και το 1.
Το παραπάνω να το πετυχαίνει **χωρίς να χρησιμοποιεί** κάποια δομή επανάληψης.

2.3 Ολοκληρώστε το πρόγραμμά σας υλοποιώντας τη συνάρτηση `main` που θα καλεί την `collatz_len` για συγκεκριμένο σταθερό αριθμό (πχ το 22) και θα εκτυπώνει την επιστρεφόμενη τιμή της.

[1] Εικάζεται, αλλά δεν έχει αποδειχθεί μαθηματικά, ότι αν ξεκινήσουμε από ένα θετικό ακέραιο αριθμό n και πάρουμε σαν επόμενό του τον n/2, αν ο n είναι άρτιος, ή τον 3n+1, αν ο n είναι περιττός, συνεχίζοντας με αυτόν τον τρόπο, κάποια στιγμή θα καταλήξουμε στο 1. Για παράδειγμα, αν ξεκινήσουμε από το n=22, η ακολουθία αριθμών που θα πάρουμε με αυτή τη διαδικασία θα είναι η 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1.

Άσκηση 3: Υπολογισμός ημέρας μίας δεδομένης ημερομηνίας

Ο ακόλουθος αλγόριθμος υπολογίζει την ημέρα που αντιστοιχεί σε μία δεδομένη ημερομηνία:

Έστω η ημερομηνία $DD / MM / YYYY$

Αν $MM \le 2$ τότε
$NYYYY = YYYY - 1$
$NMM = 0$
Αν $MM > 2$ τότε
$NYYYY = YYYY$
$NMM = \left\lfloor \frac{4 * MM + 23}{10} \right\rfloor$

$$IDAY = 365 * YYYY + DD + 31 * (MM - 1) - NMM + \left\lfloor \frac{NYYYY}{4} \right\rfloor - \left\lfloor \frac{3}{4} * \left(\left\lfloor \frac{NYYYY}{100} \right\rfloor + 1 \right) \right\rfloor$$

Αν $IDAY \bmod 7 = 0$ τότε $DAY = Saturday$
Αν $IDAY \bmod 7 = 1$ τότε $DAY = Sunday$
.....
Αν $IDAY \bmod 7 = 6$ τότε $DAY = Friday$

Με $\lfloor x \rfloor$ συμβολίζουμε τον μέγιστο ακέραιο αριθμό που δεν είναι μεγαλύτερος του αριθμού x.

Γράψτε πρόγραμμα C που να βρίσκει τι ημέρα γεννηθήκατε (ενσωματώστε την ημερομηνία γέννησής σας μέσα στο πρόγραμμα).

ΠΑΡΑΡΤΗΜΑ: Αποσφαλμάτωση προγραμμάτων (Πράξη 2η)

Στο προηγούμενο εργαστήριο είδαμε την μέθοδο για να ανιχνεύουμε και να διορθώνουμε συντακτικά λάθη. Στο παρόν εργαστήριο θα συνεχίσουμε τη συζήτηση για την αποσφαλμάτωση προγραμμάτων, εστιάζοντας αυτή τη φορά στα λάθη λογικής.

Λογικά λάθη

Ένα λογικό λάθος είναι ένα λάθος που, ενώ επιτρέπει την επιτυχή μεταγλώττιση και εκτέλεση του προγράμματος, τελικά κάνει τα αποτελέσματά του να μην είναι σωστά. Αυτό μπορεί να οφείλεται είτε σε κάποιο λάθος στην αρχική μας σκέψη για το πως θα λύσουμε το πρόβλημα, είτε ακόμα και στη λανθασμένη υλοποίηση μιας σωστής σκέψης.

Για παράδειγμα ας θεωρήσουμε το πρόβλημα να υπολογίσουμε το άθροισμα 1+3+5+... για τους πρώτους 20 όρους. Ένα πρόγραμμα που ευελπιστεί να κάνει αυτή τη δουλειά είναι το εξής

```
#include <stdio.h>
#define N 20

main() {
    int S=0, a=1;
    while (a<=N) {
        S = S+a;
        a = a+2;
    }
    printf("%d\n", S);
}
```

Το προηγούμενο πρόγραμμα μεταγλωττίζεται με απόλυτη επιτυχία και εμφανίζει αποτελέσματα. Το πρόβλημα είναι ότι το αποτέλεσμα που εμφανίζει, το 100, δεν είναι σωστό. Εμείς θα αναμέναμε το 400. Άρα στο πρόγραμμά μας υπάρχει κάποιο λογικό λάθος το οποίο πρέπει να αναζητήσουμε και να διορθώσουμε.

Τεχνικές εύρεσης και διόρθωσης λογικών λαθών

Τα λογικά λάθη είναι, κατά γενική ομολογία, πολύ δύσκολο να εντοπιστούν. Ενώ με τα συντακτικά λάθη μπορούμε να αντλήσουμε πολύτιμες πληροφορίες από τα μηνύματα του μεταγλωττιστή, με τα λογικά λάθη όλα θα λειτουργούν σωστά, με εξαίρεση ότι τελικά θα παίρνουμε λανθασμένα αποτελέσματα (ή και καθόλου αποτελέσματα).

Όταν αντιμετωπίζουμε ένα λογικό λάθος το πρώτο που πρέπει να κάνουμε είναι να προσπαθήσουμε να εντοπίσουμε τις πιθανές πηγές του. Για να το πετύχουμε αυτό πρέπει να δούμε με προσοχή τα αποτελέσματα που εκτυπώνει το πρόγραμμά μας και να σκεφτούμε πιθανές αιτίες που μπορεί να οδήγησαν σε τέτοιου είδους αποτελέσματα. Για παράδειγμα, αν θέλουμε να εκτυπώνουμε κάποια στοιχεία με βάση ένα κριτήριο και καταλήγουμε να εκτυπώνουμε όλα τα στοιχεία, τότε πιθανότατα το πρόβλημα βρίσκεται στον τρόπο που έχουμε υλοποιήσει το κριτήριο επιλογής.

Έχοντας σκεφτεί περίπου τι μπορεί να φταίει εντοπίζουμε το επίμαχο κομμάτι του κώδικα και το ξανακοιτάμε με προσοχή. Μήπως κάτι που έχουμε γράψει δε συνάδει με την αρχική μας σκέψη; Μήπως έχουμε παραλείψει κάτι;

Αν είναι δύσκολο να δούμε με το μάτι τι μπορεί να φταίει τότε πρέπει να καταφύγουμε σε μια “ιχνηλάτηση” του προγράμματος. Στο επίμαχο κομμάτι του κώδικα και για όσες μεταβλητές περιλαμβάνει εκτυπώνουμε την τιμή τους στα διάφορα στάδια εκτέλεσης με τη χρήση συναρτήσεων

`printf`. Έτσι, μπορούμε να δούμε τις τιμές που παίρνουν και να αποκτήσουμε μια καλύτερη εικόνα για το στάδιο στο οποίο η υλοποίησή μας αρχίζει να χωλαίνει.

Ας επανέλθουμε στο προηγούμενο παράδειγμα και ας δούμε πως θα το αντιμετωπίζαμε με βάση όσα είπαμε. Αρχικά παρατηρούμε ότι το αποτέλεσμα που παίρνουμε δεν είναι σωστό, συγκεκριμένα είναι σημαντικά μικρότερο από αυτό που θα περιμέναμε. Αυτό μπορεί να οφείλεται είτε στο ότι δεν κάνουμε σωστά το άθροισμα, είτε στο ότι δεν παράγουμε σωστά τους όρους που αθροίζουμε, είτε στο ότι αθροίζουμε λιγότερους όρους απ' ότι θα θέλαμε.

Σε κάθε περίπτωση, όλα τα προηγούμενα συγκλίνουν στο ότι δεν κάνουμε κάτι σωστά μέσα στη δομή επανάληψης. Οι επίμαχες μεταβλητές είναι το `S` και το `a`, οπότε ας εμφανίζουμε τις τιμές τους με σωστή τοποθέτηση συναρτήσεων `printf` ως εξής

```
main() {
    int S=0, a=1;
    while (a<=N) {
        S = S+a;
        printf("%d %d\n", S, a);
        a = a+2;
    }
    printf("%d\n",S);
}
```

Εκτελώντας αυτόν τον κώδικα παίρνουμε σαν αποτέλεσμα

```
1 1
4 3
9 5
16 7
25 9
36 11
49 13
64 15
81 17
100 19
100
```

Από το αποτέλεσμα καταλαβαίνουμε ότι ο υπολογισμός του αθροίσματος, αλλά και των όρων, γίνεται σωστά. Άρα το πρόβλημα πρέπει να βρίσκεται στο πλήθος των όρων που χρησιμοποιούμε, που όπως βλέπουμε δεν είναι 20 όπως θα θέλαμε, αλλά 10. Οπότε αυτό που φταίει είναι το πλήθος των φορών που εκτελείται η δομή επανάληψης. Όμως αυτό εξαρτάται από τη συνθήκη της, άρα το λάθος πρέπει να βρίσκεται σε αυτή. Αν τη δούμε καλύτερα, λέει "`while (a<=N)`", το οποίο σημαίνει ο τρέχων όρος να είναι μικρότερος του πλήθους των όρων που θέλουμε να αθροίσουμε, ενώ εμείς θέλουμε απλά να αθροίσουμε 20 όρους. Αυτό μπορούμε να το πετύχουμε αν αντικαταστήσουμε τη `while` με μια `for` της μορφής "`for (i=1 ; i<=N ; i++)`".

Φυσικά, θα υπάρξουν και φορές όπου ο κώδικάς μας θα είναι απόλυτα σύμφωνος με όσα έχουμε σκεφτεί και φαινομενικά σωστός, όμως σε κάποια παραδείγματα δεν θα λειτουργεί σωστά. Τότε αυτό που έχει συμβεί είναι ότι δεν έχουμε καλύψει όλες τις πιθανές περιπτώσεις, μια υποχρέωση που είναι εκ των ων ουκ άνευ για έναν σωστό προγραμματιστή. Γι' αυτό πρέπει πάντα να δοκιμάζουμε εξονυχιστικά τα προγράμματά μας με διάφορα παραδείγματα, όχι μόνο απλά, αλλά και σύνθετα και ασυνήθιστα, για να βεβαιωθούμε ότι είναι όντως σωστά.

Αν κάποιο από τα παραδείγματα που δοκιμάσαμε δεν δίνει το αναμενόμενο αποτέλεσμα τότε έχουμε παραλείψει στον κώδικά μας να καλύψουμε μια τέτοια περίπτωση. Οπότε πρέπει να σκεφτούμε ποια είναι τα χαρακτηριστικά του συγκεκριμένου παραδείγματος που το κάνουν να μην εμπίπτει σε όσα έχουμε σκεφτεί και υλοποιήσει. Για να το πετύχουμε αυτό, είναι πολύ πιθανό ότι θα χρειαστούμε και πάλι να καταφύγουμε σε όσες τεχνικές περιγράψαμε πριν και να τοποθετήσουμε κατάλληλες `printf` σε κρίσιμα σημεία του προγράμματός μας. Μόλις βρούμε τι φταίει, εμπλουτίζουμε την αρχική μας σκέψη ώστε να καλύψει και αυτό το είδος περιπτώσεων και κάνουμε τις απαραίτητες προσθήκες στον κώδικά μας.

Γενικά, η διόρθωση λογικών λαθών απαιτεί εμπειρία, υπομονή και εξοικείωση με το πρόβλημα και τον κώδικά μας. Αν και περιγράψαμε τεχνικές για την εύρεση και διόρθωση λογικών σφαλμάτων, πρέπει να έχουμε κατά νου ότι κάθε λογικό λάθος είναι διαφορετικό από οποιοδήποτε άλλο. Ο τρόπος με τον οποίο θα φτάσουμε από τα λανθασμένα αποτελέσματα στην αιτία τους μέσα στον κώδικά μας απαιτεί αναλυτική σκέψη και δε γίνεται να υπάρξουν καθολικά εφαρμόσιμες τεχνικές γι' αυτό το σκοπό.

Σημείωση: Αν και αναφέραμε έναν τρόπο αποσφαλμάτωσης μέσω `printf`, αυτός ο τρόπος δεν είναι πάντα αποτελεσματικός ή εύκολα υλοποιήσιμος, ειδικά σε μεγάλα προγράμματα. Γι' αυτό το σκοπό υπάρχουν εξειδικευμένα εργαλεία, γνωστά ως debuggers, που βοηθούν πολύ τη διαδικασία αποσφαλμάτωσης. Είναι σημαντικό με την πρώτη ευκαιρία να εξοικειωθείτε με τη χρήση κάποιου τέτοιου προγράμματος, π.χ. με τον gdb που βρίσκεται εγκατεστημένος στα μηχανήματα Linux του Τμήματος. Παρουσίαση του gdb καθώς και του debugger του Dev-C++ θα γίνει στο Εργαστήριο 8.

Άσκηση

Το παρακάτω πρόγραμμα ευελπιστεί να υπολογίζει την παράσταση 1 - 1/2 + 1/4 - 1/8 + 1/16 - ... για όσους όρους είναι μεγαλύτεροι από το `LIMIT`. Για να το πετύχει αυτό ο προγραμματιστής του έκανε την εξής σκέψη "Σε μια μεταβλητή `c` θα κρατάω τον τρέχοντα όρο που θα προστεθεί, ο οποίος παράγεται από τον προηγούμενο αν πολλαπλασιάσω με ½. Θέλω ανά πάσα στιγμή ο όρος `c` να μην ξεπερνά το `LIMIT`. Το `c` έχει διαφορετικό πρόσημο σε κάθε όρο, οπότε θα χρησιμοποιήσω μια μεταβλητή `sign` που θα την πολλαπλασιάζω με το -1 ώστε ανά δύο όρους να έχει το ίδιο πρόσημο. Τελικά θα εκτυπώσω το άθροισμα `S`". Το πρόγραμμα αυτό όμως, έχει συντακτικά και λογικά λάθη. Μπορείτε να το αποσφαλματώσετε;

```
#include <stdio.h>
#define LIMIT 0.008

main() {
    float S=0, c=1;
    int sign=1;

    while (c<LIMIT) {
        S = S+c;
        sign = (-1)*sign;
        c = sign*1/2*c;
    }
    print("%f\n", s);
}
```