Προγραμματισμός Η/Υ Ι (Χρήση της C)
1η Θεωρία
Εισαγωγή στη Γλώσσα Προγραμματισμού C

1. Γενικά χαρακτηριστικά και κανόνες σύνταξης της γλώσσας C

Στον πίνακα 1-1 αναγράφονται οι 32 δεσμευμένες λέξεις που χρησιμοποιεί το πρότυπο ANSI της C. Θα πρέπει όμως να τονιστεί ότι ο προγραμματιστής μπορεί να συναντήσει και άλλες δεσμευμένες λέξεις ανάλογα με την έκδοση του μεταγλωττιστή και την εταιρία που τον συνέταξε. Ο πλήρης κατάλογος των δεσμευμένων λέξεων για την έκδοση της C που χρησιμοποιείται θα πρέπει να αναζητηθεί στο αντίστοιχο εγχειρίδιο.

auto	double	int	struct
break	else	long	switch
case	enum	register	while
char	extern	static	typedef
const	float	return	volatile
continue	for	short	union
default	goto	signed	unsigned
do	if	sizeof	void

Πίνακας 1-1. Δεσμευμένες λέξεις από το πρότυπο ANSI C

Στη γλώσσα C υπάρχει διάκριση μεταξύ πεζών και κεφαλαίων χαρακτήρων. Ετσι, η λέξη for είναι μια **δεσμευμένη** λέξη, ενώ η λέξη FOR ή η λέξη For δεν είναι.

Εκτός από τις συναρτήσεις που μπορεί να γράψει ο προγραμματιστής η C διαθέτει και μία πλούσια βιβλιοθήκη έτοιμων συναρτήσεων. Οι συναρτήσεις αυτές βρίσκονται καταχωρημένες κατά κατηγορίες σε διάφορα αρχεία τα οποία συνοδεύουν το μεταγλωττιστή. Η επίκληση μιας τέτοιας συνάρτησης απαιτεί ειδική αναφορά στο αρχείο στο οποίο βρίσκεται κατά τη συγγραφή του πηγαίου κώδικα.

Η C απαιτεί να γίνεται προσδιορισμός του τύπου κάθε στοιχείου που χρησιμοποιείται στον κώδικα (π.χ. μεταβλητές, συναρτήσεις, ετικέτες κ.τ.λ.). Οι προσδιορισμοί τύπου μπορεί να είναι γενικοί ή τοπικοί. Οι γενικοί γράφονται έξω από το σώμα των συναρτήσεων και έχουν ισχύ για όλες τις συναρτήσεις που βρίσκονται στο ίδιο αρχείο και ο κώδικάς τους είναι γραμμένος κάτω από το σημείο όπου γράφτηκαν οι προσδιορισμοί. Οι τοπικοί προσδιορισμοί γράφονται στο εσωτερικό των αγκίστρων που ορίζουν το σώμα της συνάρτησης και έχουν ισχύ μόνο για τα στοιχεία της συγκεκριμένης συνάρτησης.

Στους γενικούς προσδιορισμούς περιλαμβάνονται και οι αναφορές των αρχείων που περιέχουν τις συναρτήσεις βιβλιοθήκης τις οποίες χρησιμοποιεί το πρόγραμμα.

Τέλος, κάθε εντολή στον κώδικα του προγράμματος πρέπει να τελειώνει με το χαρακτήρα **;**.

Στον κώδικα ενός προγράμματος γραμμένου σε γλώσσα C μπορούμε να εισάγουμε σχόλια ή τίτλους που θα το κάνουν πιο κατανοητό. Η εισαγωγή σχολίων γίνεται με την αναγραφή τους μεταξύ των χαρακτήρων /* και */. Ο μεταγλωττιστής θα αγνοήσει οτιδήποτε υπάρχει ανάμεσα στα ζεύγη αυτών των χαρακτήρων. Π.χ. η εγγραφή

```
/* Πρόγραμμα: test.c
   Γλώσσα: ANSI C */
```

αποτελεί σχόλιο που αγνοείται από το μεταγλωττιστή.

Η συνάρτηση main ( )

Ο κώδικας ενός προγράμματος γραμμένου σε γλώσσα C αποτελείται από μία ή περισσότερες συναρτήσεις. Κάθε πρόγραμμα πρέπει απαραίτητα να περιλαμβάνει στον κώδικά τού τη συνάρτηση main( ) η οποία αποτελεί για τους μεταγλωττιστές της γλώσσας το ρόλο που παίζει το κυρίως πρόγραμμα σε άλλες γλώσσες προγραμματισμού. Ο κώδικας κάθε συνάρτησης ξεκινά πάντα με ένα αριστερό άγκιστρο και τελειώνει με ένα δεξιό άγκιστρο. Στο εσωτερικό των αγκίστρων γράφονται οι τοπικοί προσδιορισμοί και οι εντολές του κώδικα.

```
main()
{

}
```

Το παραπάνω πρόγραμμα που βρίσκεται είναι το απλούστερο δυνατό πρόγραμμα σε C. Δεν υπάρχει τρόπος να απλοποιήσουμε άλλο αυτό το πρόγραμμα ή να παραλείψουμε κάτι. Δυστυχώς το πρόγραμμα αυτό δεν κάνει τίποτε.

Η λέξη κλειδί main είναι πολύ σημαντική και πρέπει να εμφανιστεί μία φορά σε κάθε πρόγραμμα C. Πρόκειται για το σημείο που αρχίζει η εκτέλεση του προγράμματος. Δεν είναι απαραίτητο να είναι η πρώτη πρόταση του κειμένου του προγράμματος αλλά πρέπει να υπάρχει κάπου ως σημείο εισόδου του μεταφραστή. Τη λέξη κλειδί main ακολουθεί ένα ζευγάρι παρενθέσεων που υποδεικνύει στον μεταφραστή οτι η main είναι μία συνάρτηση. Οι συναρτήσεις συζητούνται παρακάτω, για την ώρα απλά ξέρουμε ότι οι παρενθέσεις είναι απαραίτητες.

Το ζευγάρι των αγκυλών χρησιμοποιείται για να ορίσει τα όρια του κειμένου του προγράμματος. Οι προτάσεις του προγράμματος βρίσκονται ανάμεσα στις αγκύλες. Στη περίπτωση του παραπάνω προγράμματος διαπιστώνουμε ότι το πρόγραμμα δεν περιλαμβάνει προτάσεις και γι΄ αυτό δεν κάνει τίποτε.

Η συνάρτηση #define

Μπορούμε να μαζέψουμε όλες τις προτάσεις #define σ' ένα αρχείο και να το ονομάσουμε, π.χ. *const.h*. Μετά, στην αρχή κάθε προγράμματος μπορούμε να εισάγουμε την πρόταση #include "const.h" και όταν τρέξει το πρόγραμμα, ο προεπεξεργαστής θα διαβάσει το αρχείο const.h και θα χρησιμοποιήσει όλες τις προτάσεις #define που θα βρει εκεί.

Χρησιμοποιούμε την ονομασία "const.h" για να ψάξει πρώτα στον τρέχοντα κατάλογο και μετά στους καταλόγους του συστήματος, ενώ η ονομασία <const.h> σημαίνει να ψάξει στον κανονικό κατάλογο πρώτα. Μια άλλη σπουδαία δυνατότητατης #define είναι ότι μπορούμε να κάνουμε τέτοιες αντικαταστάσεις ως εξής :

```
#define program main()

#define begin {

#define end }

#define times *
```

Έτσι, όπου ο προεπεξεργαστής βρει τους αριστερούς όρους, τους αντικαθιστά μ' ό,τι είναι δεξιά

Η συνάρτηση #define για σταθερές

Εδώ εξηγούμε τι συμβαίνει στις γραμμές που αρχίζουν με τη λέξη #define. Αυτές αποτελούν εντολές για τον cpp και όχι για τον compiler. Το νόημα μιας εντολής

```
#define name text
```

είναι ότι ζητείται από τον cpp οποτεδήποτε βλέπει το string *name* στο κείμενο του προγράμματος να αντικαθιστά αυτό με το κείμενο *text*. Το string *name* πρέπει να είναι του ίδιου τύπου με αυτά που χρησιμοποιούνται για μεταβλητές, δηλαδή να αρχίζει με γράμμα ή underscore (_) και να περιέχει μόνο γράμματα, αριθμούς και underscores, αλλά το κείμενο *text* είναι τελείως ελεύθερο και αποτελείται από ολόκληρο το υπόλοιπο κομμάτι της γραμμής του #define. Ο cpp λοιπόν, θα αντικαταστήσει μέσα στο πρόγραμμα οποιαδήποτε εμφάνιση του identifier GRADE_A_MIN με το κείμενο 0.9. Γιατί το κάνουμε αυτό και δε γράφουμε κατ' ευθείαν το 0.9 εκεί που το χρειαζόμαστε; Η απάντηση είναι τόσο απλή όσο και σημαντική: γιατί ο identifier GRADE_A_MIN παριστάνει μια έννοια που μπορεί να χρησιμοποιείται σε πάρα πολλά σημεία μέσα στο πρόγραμμα. Στο συγκεκριμένο πρόγραμμα δε συμβαίνει αυτό αλλά θα μπορούσε κάλλιστα το πρόγραμμα να περιλαμβάνει μερικές δεκάδες αναφορές στο GRADE_A_MIN. Αν αντί για GRADE_A_MIN εμείς είχαμε γράψει 0.9 και ξαφνικά αποφάσιζε ο καθηγητής που κάνει το μάθημα να ελαττώσει το κατώφλι του βαθμού Α σε 0.85 εμείς θα έπρεπε να ψάξουμε να δούμε που μέσα στο πρόγραμμα έχουμε κάνει μια αναφορά στην έννοια αυτή και να αντικαταστήσουμε το 0.9 με 0.85. Ενώ τώρα το μόνο που χρειζεται να κάνουμε είναι να αλλάξουμε το αντίστοιχο #define σε

```
#define GRADE_A_MIN 0.85
```

Γίνεται έτσι φανερό πόσο το να τηρεί κανείς κάποιους κανόνες σωστού σχεδιασμού για το πρόγραμμά του μπορεί να βοηθήσει στο γράψιμο και τη συντήρηση αυτού (τυχόν αλλαγές που ζητούνται στη διάρκεια του χρόνου, λάθη που ανακαλύπτονται και πρέπει να διορθωθούν, κλπ).

Ακολουθεί ένα πρόγραμμα **χωρίς** και **με** τη χρήση **σταθερών** τιμών.

```
#include <stdio.h>
main()
{
    float tel;
    printf ("\nΤιμή τηλεόρασης\n");
    scanf ("%f", &tel);
    tel = tel + 0.18 * tel;
    printf ("\n\n");
    printf ("Τιμή με ΦΠΑ \n");
    printf ("%f ", tel);
}
```

Χρησιμοποιώντας μια σταθερά:

```
#include <stdio.h>
#define FPA 0.18
main()
{
    float tel;
    printf ("\nΤιμή τηλεόρασης\n");
    scanf ("%f", &tel);
    tel = tel + FPA * tel;
    printf ("\n\n");
    printf ("Τιμή με ΦΠΑ \n");
    printf ("%f \n", tel);
}
```

Η συνάρτηση printf( )

Παράδειγμα

Τι θα εμφανιστεί στην οθόνη;

```
main()
{
   printf("αυτή είναι μια γραμμή κειμένου");
}
```

Απάντηση

Η πρόταση είναι μια κλήση σε μια συνάρτηση που περιλαμβάνεται στη standard βιβλιοθήκη εισόδου/εξόδου της C. Η συγκεκριμένη συνάρτηση εμφανίζει μηνύματα στην οθόνη. Το μήνυμα τοποθετείται ανάμεσα σε διπλά εισαγωγικά και στη συνέχεια ανάμεσα στις παρενθέσεις της printf.

Προσοχή στο ελληνικό ερωτηματικό στο τέλος της γραμμής. Είναι το οριοθετικό της πρότασης στη C και είναι αναγκαίο για τον μεταφραστή για να θεωρήσει τη πρόταση πλήρη.
Μετά την εκτέλεση η οθόνη πρέπει να περιέχει τα παρακάτω:

αυτή είναι μια γραμμή κειμένου

Η **printf εμφανίζει** στην οθόνη ό,τι βρίσκεται ανάμεσα στα " και ".
Π.χ.

```
printf ("Άθροισμα = %d\n", num1+num2);
```

\n : **χαρακτήρας ελέγχου (μεταφορά στην επόμενη γραμμή)**
(θεωρείται ότι είναι ένας μόνο χαρακτήρας)
%d : **προσδιοριστής μορφής (προδιαγραφή)**.
Ορίζει πού και πώς θα εμφανιστεί ένας ακέραιος

Παράδειγμα

Τι θα εμφανιστεί στην οθόνη;

```
main()
{
   printf("This is a line of text to output.\n");
   printf("And this is another ");
   printf("line of text.\n\n");
   printf("This is a third line.\n");
```

Απάντηση

Το πρόγραμμα περιέχει τέσσερις προτάσεις, όπου η
κάθε μία είναι μία κλήση της συνάρτησης printf.

Η σειρά εκτέλεσης των προτάσεων καθορίζεται από την σειρά που είναι γραμμένες στο κείμενο του προγράμματος, έτσι ώστε η εκτέλεση ξεκινά από την πρώτη γραμμή και ακολουθούν η δεύτερη, η τρίτη και τελικά η τέταρτη.

Σημειώστε τον χαρακτήρα '\' (backslash) στο τέλος των μηνυμάτων. Το backslash χρησιμοποιείται για να ειδοποιήσει τον μεταφραστή ότι οι χαρακτήρες που ακολουθούν δεν είναι εκτυπώσιμοι αλλά αποτελούν χαρακτήρες ελέγχου. Έτσι ο χαρακτήρας 'n' σημαίνει νέα γραμμή (new line, carriage return/line feed) έτσι ώστε ο δρομέας της οθόνης μετακινείται μία γραμμή προς τα κάτω και τοποθετείται στην πρώτη θέση της νέας γραμμής. Το '\n' θα μπορούσε να είχε τοποθετηθεί οπουδήποτε μέσα στο μήνυμα και να το κόψει σε δύο γραμμές.
Τώρα το πρόγραμμα μπορεί να περιγραφεί πλήρως. Η πρώτη πρόταση εμφανίζει ένα μήνυμα και πηγαίνει σε νέα γραμμή. Η δεύτερη πρόταση εμφανίζει μήνυμα αλλά δεν αλλάζει γραμμή έτσι ώστε η τρίτη πρόταση συνεχίζει την εμφάνιση του δικού της μηνύματος στην ίδια γραμμή. Τελειώνει όμως με δύο αλλαγές γραμμής έτσι ώστε έχουμε μια κενή γραμμή στην οθόνη. Η τελευταία γραμμή τυπώνει το μήνυμά της, αλλάζει γραμμή και το πρόγραμμα τερματίζει. Μετά την εκτέλεση η οθόνη πρέπει να περιέχει τα παρακάτω

This is a line of text to output.

And this is another line of text.

This is a third line.

1η άσκηση

Να γραφεί πρόγραμμα που θα εμφανίζει στην οθόνη τα εξής:

Όνομα: το όνομά σας

Εξάμηνο: το εξάμηνό σας

2η άσκηση

Να γραφεί πρόγραμμα που θα εμφανίζει στην οθόνη τα εξής:

Όνομα: το όνομά σας

Ηλικία: την ηλικία σας

3η Άσκηση

Να γραφεί πρόγραμμα που θα εμφανίζει στην οθόνη τα εξής:

Καλημέρα, (3 σειρές κενό), Καλημέρα

Στις θέσεις που θέλουμε να τυπωθεί μια έκφραση ή μια μεταβλητή τοποθετούμε το χαρακτήρα `%` ακολουθούμενο από κάποιους κωδικούς χαρακτήρες ή και αριθμούς οι οποίοι δηλώνουν τον τύπο της προς τύπωση έκφρασης και διάφορες άλλες παραμέτρους. Μπορεί κανείς π.χ. να προσδιορίσει πόσα δεκαδικά ψηφία θα χρησιμοποιηθούν όταν τυπώνεται ένας `double`. Μερικές από τις δυνατές επιλογές για το κείμενο που ακολουθεί το σύμβολο `%` είναι οι ακόλουθες:

Κείμενο στο format string	**Τι τυπώνεται**
`%d`	`int`, δεκαδική αναπαράσταση

%u	unsigned int, δεκαδική αναπαράσταση
%x	unsigned int, δεκαεξαδική αναπαράσταση
%o	unsigned int, οκταδική αναπαράσταση
%f	float/double, συνήθης αναπαράσταση με υποδιαστολή
%e	float/double, εκθετική μορφή
%g	float/double, επιλέγεται αυτόματα %f ή %e ανάλογα με τον αριθμό
%c	int ή char, τυπώνεται το ASCII σύμβολο που αντιστοιχεί στον αριθμό, π.χ. αν η έκφραση είναι ίση με 65 θα τυπωθεί το A κεφαλαίο λατινικό
%s	διεύθυνση χαρακτήρα (char *), τυπώνεται η λέξη που έχει αποθηκευτεί να ξεκινά από εκείνη τη διεύθυνση χαρακτήρα
%p	διεύθυνση μνήμης

Παράδειγμα

Τι θα εμφανίσει το ακόλουθο πρόγραμμα;

```
 1 #include        <stdio.h>
 2
 3    main()
 4        {
 5         printf("[%d]\n", 106);
 6         printf("[%6d]\n", 106);
 9         printf("[%-6d]\n", 106);
10         printf("[%06d]\n", 106);
11         printf("[%c]\n", 106);
 }
```

Απάντηση

```
% a.out
[106]
[   106]
[106   ]
[000106]
[j]
```

Σε κάθε μια από τις γραμμές του output όπου τυπώνεται ένας αριθμός (ακέραιος ή δεκαδικός) τυπώνεται μέσα σε ένα ζεύγος αγκυλών, ώστε να είναι φανερό ακριβώς πόσες θέσεις πιάνει. Αλλιώς οι λευκοί χαρακτήρες (blanks) δε θα φαίνονταν. Στην πρώτη γραμμή τυπώνεται ο ακέραιος 106 σε μορφή δεκαδική. Στις επόμενες δύό γραμμές τυπώνεται ο ίδιος ακέραιος έτσι ώστε να καταλαμβάνει 6 θέσεις και στοιχισμένος αριστερά (που είναι η default στοίχιση) και στοιχισμένος δεξιά. Στην επόμενη γραμμή καταλαμβάνει πάλι 6 θέσεις αλλά συμπληρώνεται με μηδενικά προς τα αριστερά. Η τελευταία γραμμή του πρώτου block γραμμών είναι ίσως η πιο ενδιαφέρουσα και μη ασυνήθιστη σε άλλες γλώσσες προγραμματισμού. Σε αυτήν ο ακέραιος 106 τυπώνεται ως χαρακτήρας: δηλ. τυπώνεται το σύμβολο που του αντιστοιχεί στον πίνακα ASCII που είναι το λατινικό γράμμα j.

4η άσκηση

Τι θα εμφανίσει το ακόλουθο πρόγραμμα;

```
 1    #include  <stdio.h>
 3    main()
 4        {
 5         printf("[%d]\n", 212);
 6         printf("[%6d]\n", 212);
 9         printf("[%-6d]\n", 212);
10         printf("[%06d]\n", 212);
          }
```

5η άσκηση

Τι θα εμφανίσει το ακόλουθο πρόγραμμα;

```
 1 #include  <stdio.h>
 2
 3    main()
 4        {
 5         printf("[%d]\n", 768);
 6         printf("[%6d]\n", 768);
 9         printf("[%-6d]\n", 768);
10         printf("[%06d]\n", 768);
          }
```

Η συνάρτηση scanf( )

Διαβάζει την τιμή μιας μεταβλητής από το πληκτρολόγιο.

Η γραμμή scanf("%d", &a); διαβάζει ένα ακέραιο από το πληκτρολόγιο και τον αποθηκεύει στη μεταβλητή `a`. Το format string είναι το "Το νόημα του "διαβάσει από το πληκτρολόγιο ένα προσημασμένο δεκαδικό (εξ ού και το `d`, από το decimal) ακέραιο, και να τον τοποθετήσει στη θέση που λέει η επόμενη διεύθυνση μνήμης που έχει περαστεί ως όρισμα στη `scanf`.

Παράδειγμα

Πώς θα διαβάσουμε τους αριθμούς `a`, `b` και `x` που έχουν δηλωθεί ως εξής;

```
int    a, b;
double  x;
```

Απάντηση

Η απάντηση είναι:

```
scanf("%d %d %f", &a, &b, &x);
```

6η Άσκηση

Να γραφεί πρόγραμμα που να διαβάζει και να τυπώνει 1 ακέραιο αριθμό(int), 1 δεκαδικό(float) και 1 χαρακτήρα(char).

ΒΑΣΙΚΑ ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΤΗΣ ΓΛΩΣΣΑΣ

- Γλώσσα ελεύθερης έκφρασης, **ελεύθερου format**.
- **Κάνει διάκριση** μικρών/κεφαλαίων.
- Τα σχόλια τοποθετούνται ανάμεσα σε /* και */
- Εντολή #include : **οδηγία** προς τον **προεπεξεργαστή**
- Αποτελείται από Συναρτήσεις

- main() : η κύρια συνάρτηση, υπάρχει πάντα μόνο μία
- Αρχή και τέλος συναρτήσεων τα άγκιστρα { }
- Η συνάρτηση printf **εμφανίζει στην οθόνη** ό,τι είναι ανάμεσα στα "….".

Ονόματα μεταβλητών

- Αποτελούνται από γράμματα του αγγλικού αλφαβήτου και ψηφία, με τον πρώτο χαρακτήρα να είναι γράμμα.
- Ο χαρακτήρας υπογράμμιση "_" θεωρείται γράμμα (χρήσιμος για να βελτιώσει την αναγνωσιμότητα μεγάλων ονομάτων μεταβλητών). Συμβουλή: Μην αρχίζετε ονόματα μεταβλητών με χαρακτήρα υπογράμμισης (χρησιμοποιείται συνήθως από συναρτήσεις βιβλιοθήκης).
- Γίνεται διάκριση σε μικρά (πεζά) και κεφαλαία γράμματα. Παραδοσιακή πρακτική: χρησιμοποιούμε μικρά γράμματα για ονόματα μεταβλητών.
- Τουλάχιστον οι 31 χαρακτήρες ενός ονόματος είναι σημαντικοί.
- Λέξεις κλειδιά όπως **int, if, else** είναι δεσμευμένες και δεν μπορούμε να τις χρησιμοποιήσουμε στα ονόματα μεταβλητών.
- Καλό είναι να επιλέγουμε ονόματα μεταβλητών σχετικά με τον σκοπό που εξυπηρετεί η κάθε μεταβλητή.

Μεταβλητές, σταθερές, τελεστές και εκφράσεις στη C

Ο τύπος ενός αντικειμένου (συνάρτησης ή μεταβλητής) προσδιορίζει το πεδίο τιμών που αυτό επιστρέφει όταν αποτιμάται, αλλά και την μνήμη που χρειάζεται για να αποθηκευτεί η αντίστοιχη πληροφορία (μόνο για μεταβλητές)

- υπάρχουν οι εξής βασικοί τύποι:

`char`	χαρακτήρας
`int`	ακέραιος αριθμός
`float`	δεκαδικός αριθμός
`double`	δεκαδικός αριθμός διπλής ακρίβειας
`void`	μη ύπαρξη τιμής

- για τον τύπο `int` υπάρχει προσδιορισμός `long/short` που υποδεικνύει τον αριθμό των bits που χρησιμοποιούνται για την αποθήκευση των τιμών - το μέγεθος **διαφέρει** από σύστημα σε σύστημα!
- για τους τύπους `char` και `int` υπάρχει προσδιορισμός `signed/unsigned` (με/χωρίς πρόσημο) - δεν αλλάζει τον αριθμό των bits που χρησιμοποιούνται για αποθήκευση αλλά προσδιορίζει την ερμηνεία των δεδομένων
- σύμβαση: όταν χρησιμοποιούνται οι προσδιορισμοί `long/short` και `signed/unsigned` σε `int` μπορεί να παραληφθεί η ένδειξη `int` (άσχημο στυλ!)

- εκτός από τους βασικούς τύπους υπάρχουν και οι παραγόμενοι τύποι που δημιουργούνται ως πίνακες `"[]"`, δείκτες `"*"` και συναρτήσεις `"()"` βασικών τύπων
- τα strings είναι μια ειδική περίπτωση πινάκων, με στοιχεία τύπου `char`, που τερματίζονται (κατά σύμβαση) με `'\0'`

Παράδειγμα

"Hello", Το πρώτο πρόγραμμα C

```
#include <stdio.h>

int main()

{

printf("Hello there\n");

return 0;

}
```

Παρατηρήσεις

1. Το σώμα του προγράμματος περικλείεται στις αγκύλες "{", "}".
2. Κάθε εντολή της C τελειώνει με ";" (semicolon).
3. Σχόλια μιας γραμμής εισάγονται "//". Σχόλια περισσοτέρων γραμμών περικλείονται με /*.....*/
4. Η πρώτη γραμμή #include <stdio.h> ενσωματώνει το αρχείο standard I/O στο πρόγραμμά μας. Το αρχείο αυτό περιέχει συναρτήσεις I/O.
5. Το κύριο πρόγραμμα αρχίζει με "int main()" και επιστρέφει (return) 0.
6. Η εντολή printf εμφανίζει στην οθόνη την συμβολοσειρά "Hello there" και μετά την χαρακτήρα αλλαγής γραμμής (\n).

Παράδειγμα

Τι θα εμφανίσει το ακόλουθο πρόγραμμα;

```
main()
{
int index;
   index = 13;
   printf("The value of the index is
%d\n",index);
   index = 27;
   printf("The value of the index is
%d\n",index);
   index = 10;
   printf("The value of the index is
%d\n",index);
}
```

Απάντηση

Η αρχή του προγράμματος πρέπει να είναι κατανοητή. Το πρώτο καινούργιο στοιχείο είναι η δήλωση μεταβλητής

int index;

Η δήλωση αυτή σημαίνει οτι το πρόγραμμα χρησιμοποιεί μια ακέραια (integer) μεταβλητή με το όνομα index. Οι ακέραιες μεταβλητές μπορούν να πάρουν τιμές μεταξύ των ορίων -32768 και 32767, στους περισσότερους μεταφραστές προσωπικών υπολογιστών. Το όνομα μπορεί να είναι οποιοδήποτε συμφωνεί με τους κανόνες ονοματοδοσίας της C και η δήλωση τερματίζεται με το οριοθετικό πρότασης.

Η δήλωση μεταβλητής δεν σημαίνει αυτόματη εκχώρηση τιμής. Θα δούμε αργότερα ότι η δήλωση μπορεί να συνοδεύεται από εκχώρηση, καθώς επίσης και ότι μπορούμε να έχουμε πολλαπλές δηλώσεις στην ίδια πρόταση.

Μετά από τη δήλωση της μεταβλητής ακολουθούν έξι προτάσεις που μπορούν να ομαδοποιηθούν ανά δύο. Η πρώτη πρόταση κάθε ομάδας εκχωρεί μια ακέραια τιμή

στη μεταβλητή ενώ η δεύτερη πρόταση εμφανίζει στην οθόνη την τρέχουσα τιμή της μεταβλητής. Για παράδειγμα η πρώτη πρόταση εκχώρησης δίνει στην μεταβλητή την τιμή 13

index = 13;

Η αμέσως επόμενη πρόταση εμφανίζει την τιμή αυτή με τη βοήθεια της συνάρτησης `printf` (το πως γίνεται αυτό ακριβώς εξηγείται παρακάτω). Στη συνέχεια η μεταβλητή λαμβάνει την τιμή 27 και τελικά την τιμή 10. Πρέπει να είναι πια καθαρό ότι η μεταβλητή index αντιστοιχεί σε μιά θέση μνήμης ικανή να αποθηκεύσει έναν ακέραιο αριθμό στη μορφή του πρόσημου του 2 και μεγέθους 2 bytes. Είναι επίσης προφανές ότι κάθε νέα τιμή που εκχωρείται στη μεταβλητή έχει ως αποτέλεσμα την υπεργραφή (overwite) και άρα την απώλεια της προηγούμενης τιμής έτσι ώστε η θέση μνήμης κρατά πάντοτε την τελευταία, ή τρέχουσα, τιμή της μεταβλητής. Ας γυρίσουμε στην εμφάνιση των αριθμών στην οθόνη. Διαπιστώνουμε ότι όλες οι προτάσεις `printf` είναι ίδιες και ξεκινούν ακριβώς όπως και οι προτάσεις εμφάνισης μηνυμάτων που είδαμε μέχρι τώρα. Η πρώτη διαφορά εμφανίζεται στη χρήση του χαρακτήρα '%'. Είναι ένας χαρακτήρας ελέγχου, αντίστοιχος με τον '\', που ειδοποιεί τον μεταφραστή ότι θα πρέπει να εμφανίσει την τιμή μιας μεταβλητής και όχι ένα μήνυμα. Ο χαρακτήρας που ακολουθεί αμέσως μετά το '%', δηλαδή στη περίπτωσή μας το 'd', καθορίζει τον τύπο της μεταβλητής που η τιμή της θα εμφανιστεί. Συγκεκριμένα το 'd' ορίζει ότι πρόκειται να εμφανιστεί μια δεκαδική ακέραια μεταβλητή. Μετά το 'd' βρίσκουμε το ήδη γνωστό '\n' και το διπλό εισαγωγικό που συμβολίζει το τέλος του μηνύματος. Μετά το μήνυμα υπάρχει κόμμα και στη συνέχεια το όνομα της μεταβλητής της οποίας η τιμή θα εμφανιστεί. Εδώ βρίσκει η `printf` τις τιμές που θα εμφανίσει την οθόνη. Θα μπορούσαμε μέσα στο ίδιο μήνυμα να είχαμε περισσότερα '%d' καθώς επίσης και αντίστοιχα ονόματα μεταβλητών χωρισμένα με κόμματα στο τέλος. Το μήνυμα επομένως καθορίζει τη μορφή της εκτύπωσης και οι μεταβλητές καθορίζουν τις τιμές που θα εμφανιστούν. Πρέπει φυσικά να είμαστε προσεκτικοί έτσι ώστε ο αριθμός των '%' σε ένα μήνυμα να είναι ίδιος με τον αριθμό των μεταβλητών που ακολουθούν.

7η Άσκηση

Να γραφεί πρόγραμμα σε C που να διαβάζει 3 φορές μια μεταβλητή x και να τυπώνει «η τιμή του x είναι 7, η τιμή του x είναι 12, η τιμή του x είναι 234.

8η άσκηση

Τι θα εμφανίσει το ακόλουθο πρόγραμμα;

```
main()
{
int a;              /* simple integer type                  */
long int b;         /* long integer type                    */
short int c;        /* short integer type                   */
unsigned int d;     /* unsigned integer type                */
char e;             /* character type                       */
float f;            /* floating point type                  */
double g;           /* double precision floating point      */

   a = 1023;
   b = 2222;
   c = 123;
   d = 1234;
   e = 'X';
   f = 3.14159;
   g = 3.1415926535898;

   printf("a = %d\n",a);       /* decimal output             */
   printf("a = %o\n",a);       /* octal output               */
   printf("a = %x\n",a);       /* hexadecimal output         */
   printf("b = %ld\n",b);      /* decimal long output        */
   printf("c = %d\n",c);       /* decimal short output       */
   printf("d = %u\n",d);       /* unsigned output            */
   printf("e = %c\n",e);       /* character output           */
   printf("f = %f\n",f);       /* floating output            */
   printf("g = %f\n",g);       /* double float output        */
   printf("\n");

}
```

9η άσκηση

Πέρα από την ορθότητα ενός κώδικα C, σημαντικό ρόλο στη χρησιμότητά του παίζει και ο τρόπος γραφής του. Ξαναγράψτε τον παρακάτω κώδικα ώστε να είναι πιο ευανάγνωστος.

```
#include <stdio.h>
int x,y;main(){printf(
"\nΔώστε δύο αριθμούς: ");scanf(
"%d %d",&x,&y);printf(
"\n\nO %d είναι μεγαλύτερος.\n",(x>y)?x:y);r
return(0);}
```

Σταθερές

Σταθερά είναι μια έκφραση το αποτέλεσμα της οποίας μπορεί να υπολογισθεί κατά την μετάφραση του προγράμματος

- Διατηρούν **την ίδια τιμή** σε όλη τη διάρκεια εκτέλεσης του προγράμματος.
- Ορίζονται με **εντολές προς τον προεπεξεργαστή**.
  Π.χ.
  #define CNT 99.3
- Γράφονται συνήθως με **κεφαλαία γράμματα**.
- Μετά την τιμή **δεν υπάρχει το σύμβολο ;**

Όταν ένα σταθερό μέγεθος χρησιμοποιείται σε πολλά σημεία του προγράμματος, είναι καλό να δηλώνεται με ένα συμβολικό όνομα το οποίο να χρησιμοποιείται αντί της τιμής - έτσι το πρόγραμμα γίνεται ευανάγνωστο και διευκολύνονται οι αλλαγές

```
#define TABLE_SIZE 512
…
i=0; s=0;
while (i < TABLE_SIZE){s=s+a[i]; i=i+1;}
```

Δηλώσεις και Ορισμοί

Πριν χρησιμοποιηθεί ένα οποιοδήποτε αντικείμενο (μεταβλητή ή συνάρτηση) "πρέπει" να δηλωθεί - η δήλωση γίνεται δίνοντας πρώτα τον τύπο και μετά το όνομα του αντικειμένου

```
char c;                          /* char */
int i=1;                         /* int */
float f[];                  /* array of float */
```

```
int *p;                    /* pointer to int */
const double pi=3.1415;    /* const double */
int sqr(int);    /* function returning int */
```

- η δήλωση απλά ανακοινώνει την χρήση ενός αντικειμένου που ορίζεται κάπου αλλού (μέσω άλλης δήλωσης) ώστε να γίνει έλεγχος της σωστής χρήσης του από τον compiler
- μια δήλωση μπορεί να είναι ταυτόχρονα και ο ορισμός του αντικειμένου, όπως π.χ. για τα `i` και `pi` - μια δήλωση δεν μπορεί να είναι ορισμός αν το αντικείμενο δεν προσδιορίζεται επαρκώς, όπως π.χ. για τα `f` και `sqr`
- όταν ορίζεται ένα αντικείμενο μπορεί να αρχικοποιηθεί δίνοντας του μια συγκεκριμένη τιμή - με το πρόθεμα `const` δηλώνεται πως η τιμή του αντικειμένου δεν αλλάζει μετά την πρώτη ανάθεση τιμής
- για κάθε αντικείμενο/όνομα μπορεί να υπάρχουν πολλές δηλώσεις αλλά μόνο ένας ορισμός - οι δηλώσεις πρέπει να συμφωνούν με τον αντίστοιχο ορισμό του αντικειμένου

```
int count;
int Count;      /* ok */
float count;    /* error! */

double sqr(double d) {return (d*d)};

int sqr(int);  /* error! */
```

➢ υπάρχουν περιπτώσεις όπου ένα αντικείμενο μπορεί να χρησιμοποιηθεί χωρίς να έχει προηγηθεί δήλωση, οπότε ο compiler εξαγάγει τον τύπο από την χρήση του αντικειμένου!

Αριθμητικοί Τελεστές

+	πρόσθεση (και ατομικός τελεστής)
−	αφαίρεση (και ατομικός τελεστής)
*	πολλαπλασιασμός
/	διαίρεση
%	υπόλοιπο (μόνο για ακέραιους)

α) Καταχώρησης: =

Π.χ.

val = 589;
k = k + 1;

ΟΜΩΣ Όχι (ΠΟΤΕ!) 693 = val;

Δικαιούμαι να ΓΡΑΨΩ ΚΑΙ :

val = temp = met = 153;

β) Πρόσθεσης: + printf ("%d", 89+6);

γ) Αφαίρεσης: - printf ("%d", 89-6);

δ) Πολλαπλασιασμού: * printf ("%d", 89*6);

ε) Διαίρεσης: / printf ("%d", 89/6);

- Η **διαίρεση με float** δίνει **float**.
- Η **ακέραια διαίρεση** (int δια int) δίνει **ακέραιο αποτέλεσμα**.
- Στη διαίρεση int με float, ο int **μετατρέπεται αυτόματα** σε float.

στ) Τελεστής προσήμου: + και -
Π.χ. met = -32;

ζ) Τελεστής ακεραίου υπολοίπου: **%**

- Δίνει το **υπόλοιπο της ακέραιας διαίρεσης** δύο ακεραίων. Π.χ. 6%4 δίνει 2
- Λέγεται και **modulo.**
- **Όχι** με float.

Παράδειγμα:

```
#include <stdio.h>
#define SPM 60
    main( )
      {
       int sec, min, left;

       printf ("Δώστε αριθμό δευτερολέπτων");
       scanf("%d", &sec);
       min = sec / SPM;          /* Λεπτά */
       left = sec % SPM;
       printf ("%d δευτερόλεπτα είναι %d λεπτά και "
      "%d δευτερόλεπτα", sec, min, left);
      }
```

η) Τελεστές αύξησης και μείωσης: ++ και - -

- **++a** ή **a++** σημαίνει **a+1**
- **--b** ή **b--** σημαίνει **b-1**

Όμως:

Το a++ **δεν είναι το ίδιο** με το ++a.

Έτσι:

α) p = 2 * ++a; σημαίνει:

- **Πρώτα** αύξησε το a κατά 1
- **Μετά** πολλαπλασίασε το α επί 2 και καταχώρησε το αποτέλεσμα στο p.

β) p = 2 * a++; σημαίνει:

- **Πρώτα** πολλαπλασίασε το α επί 2 και καταχώρησε το αποτέλεσμα στο p.
- **Μετά** αύξησε το α κατά 1.

Παράδειγμα:

```
#include <stdio.h>
  main()
   {
      int a=1, b=1, aplus, plusb;

      aplus = a++;
      plusb = ++b;
      printf ("a   aplus   b   plusb   \n");
      printf ("%1d   %5d   %5d   %5d\n", a, aplus, b, plusb);
   }
```

θ) Άλλοι τελεστές καταχώρησης:

+= -= *= /= %=

x += 5;	σημαίνει	x = x + 5;
k *= 7;	σημαίνει	k = k * 7;

sum %= 15; σημαίνει sum = sum % 15;

Παραδείγματα

α)2 + 3*5
2 + 15
17

β)(2+3)*5

5* 5

25

10η Άσκηση

Να υπολογιστεί το αποτέλεσμα

α) (12+6)/3*2
β)10%3%2

9η Άσκηση
Να υπολογιστεί το αποτέλεσμα

α)8/3/3
β) -3+8/-2

11η Άσκηση

Συμπληρώστε το αποτέλεσμα για τις παρακάτω εντολές C.

a. 3.5*6=
b. (int)85.76*2=
c. 243*-83=
d. (int)(20.35*24)=
e. 23/9=

Παράδειγμα

Γράψτε τι *ακριβώς* θα εμφανιστεί στην οθόνη για καθεμιά από τις παρακάτω περιπτώσεις. Χρησιμοποιείται ο χαρακτήρας υπογράμμισης (_) για να δηλώσει τον κενό χαρακτήρα. Κάντε το ίδιο στις απαντήσεις σας.

```
i. int a, b, c = 3;
        a = b = 2; a = c > b;
        b = b == 1;
        printf("a=%d_b=%2d\nc=%d", a, b, c);
ii. int x = 5, y = 10, z = 15;
         printf("%d_%d_%d\n", x == z - y, x > y || y >
    z,
         x == 5 && !(z == 15));
```

Απάντηση

a) i. a=1_b=_0
c=3
ii. 1_0_0

12η Άσκηση

Γράψτε τι *ακριβώς* θα εμφανιστεί στην οθόνη για καθεμιά από τις παρακάτω περιπτώσεις. Χρησιμοποιείται ο χαρακτήρας υπογράμμισης (_) για να δηλώσει τον κενό χαρακτήρα. Κάντε το ίδιο στις απαντήσεις σας.

```
i. int x = 2, y = 3, z = 5;
        printf("%2d\n", z % y + x);
        printf("%2d\n", z % y - x);

ii. float x = 2.549;
    float y = 3.14159,z = 5.72;
         printf("Τιμές_μεταβλητών:_x=%3.1f_y=%3.1f_z=%3.2f\n",

x, y, z);
```

13η Άσκηση

Γράψτε τι <u>ακριβώς</u> θα εμφανιστεί στην οθόνη όταν εκτελεστεί το παρακάτω απόσπασμα κώδικα:

```
int a = 1, b = 3;
float d = 5.3, f, g;
f = 2 * a + b % 2 – b;
g = (a + b) / d;
printf("f_=_%4.2f__g_=_%5.1f\n", f, g);
```

Ο χαρακτήρας _ χρησιμοποιείται για να δηλώσει τον κενό χαρακτήρα. Κάντε κι εσείς το ίδιο στην απάντησή σας.

Σχεσιακοί και Λογικοί Τελεστές

Οι σχεσιακοί και λογικοί τελεστές χρησιμοποιούνται για την κατασκευή λογικών εκφράσεων (συνθηκών) με τις οποίες μπορεί να κατευθυνθεί η εκτέλεση του προγράμματος

>, >=	**μεγαλύτερο (ή ίσο)**
<, <=	**μικρότερο (ή ίσο)**
==, !=	**ισότητα, ανισότητα**
!	**άρνηση (ατομικός τελεστής)**

- δεν υπάρχει λογικός τύπος boolean αλλά χρησιμοποιείται το 0 ως FALSE και οποιαδήποτε διαφορετική τιμή ως TRUE

```
if (0) …    /* if FALSE */
if (!5) …   /* if FALSE */
i=(k==j);   /* i is 1 if k==j, else i is 0 */
```

- το αποτέλεσμα μιας λογικής έκφρασης είναι 0 ή 1 και μπορεί να χρησιμοποιηθεί ως ακέραιος αριθμός.
- οι σύνδεσμοι && (logical and) και || (logical or) χρησιμοποιούνται για την κατασκευή λογικών εκφράσεων

```
if ((a==b)&&(c==d)) ⇔ if!((a!=b)||(c!=d))
```

➢ η αποτίμηση των εκφράσεων που κατασκευάζονται με αυτούς τους συνδέσμους είναι **conditional** (γίνεται υπό συνθήκη) - τα ορίσματα αποτιμώνται από αριστερά προς τα δεξιά και μόνο όσο χρειάζεται για να διαπιστωθεί η τιμή της έκφρασης

```
if (x && y) …
/* does not evaluate y if x is 0 */

if (x || y) …
/* does not evaluate y if x is 1 */
```

Παράδειγμα

Έστω ότι οι μεταβλητές x, y, z έχουν τιμές 1, 2, και 2, αντίστοιχα. Σε ποια τιμή υπολογίζεται καθεμιά από τις ακόλουθες εκφράσεις;

α. `(1 + 2 * 3)`
β. `((1 + 2) * 3)`
γ. `10 % 3 * 3 - (1 + 2)`
δ. `10 % (3 * 3) - (1 + 2)`
ε. `(5 == 5)`
στ. `x = y`
ζ. `x == y`
η. `x == y == z`
θ. `x != z`
ι. `x = y = z`
ια. `x = y != z`

Απάντηση

α. 7 β. 9 γ. 0 δ. –2 ε. 1 στ. 2 ζ. 0 η. 0 θ. 1 ι. 2 ια. 0

14η Άσκηση

Έστω ότι οι μεταβλητές x, y, z έχουν τιμές 1, 3, και 4, αντίστοιχα. Για καθεμιά από τις παρακάτω εκφράσεις βρείτε αν είναι αληθής (Α) ή ψευδής (Ψ):

α. `x`
β. `x + y`
γ. `x + y == z`
δ. `y % 2 != z % 2`
ε. `!(x + y == z) || (y == z)`
στ. `(z / x == z / y) || (y > z && z < x) || (x + y != z)`

(Μια έκφραση στη C είναι αληθής αν έχει τιμή μη-μηδενική. Είναι ψευδής μόνο αν είναι μηδέν.)

15η Άσκηση

Τι θα εμφανίσει το ακόλουθο πρόγραμμα;

```
#include <stdio.h>

int main()

{

int a,b;

int c=20;

a=30;

printf("Type the value of b: ");

scanf("\n%d", &b);

printf("value of b is %d ", b);

b=a-c;

printf("value of a-c is %d", b);

return 0;
```

16η Άσκηση Τι θα εμφανίσει το ακόλουθο πρόγραμμα;

```
#include <stdio.h>

main()

{

double r1, r2;

long r3;

unsigned int r4;

r1=4.0/3; //float ---> double

r2=4/3; //int ---> double

r3=r1; //double ---> long

r4=-r1; //double --->unsigned int

printf("double: 4.0/3= ", r1);

printf("double: 4/3= ", r2 );

printf("long: 4.0/3= ", r3 );

printf("unsigned int: -(4.0/3)=", r4 );}
```

Τελεστές αύξησης και μείωσης: ++ και - -

- **++a** ή **a++** σημαίνει **a+1**
- **--b** ή **b--** σημαίνει **b-1**

Όμως:

Το a++ **δεν είναι το ίδιο** με το ++a.

Έτσι:

α) p = 2 * ++a; σημαίνει:

- **Πρώτα** αύξησε το a κατά 1
- **Μετά** πολλαπλασίασε το α επί 2 και καταχώρησε το αποτέλεσμα στο p.

β) p = 2 * a++; σημαίνει:

- **Πρώτα** πολλαπλασίασε το α επί 2 και καταχώρησε το αποτέλεσμα στο p.
- **Μετά** αύξησε το α κατά 1.
- π.χ. x *= y + 1; σημαίνει x = x * (y+1);

Άλλοι τελεστές καταχώρησης:

+= -= *= /= %=

x += 5;	σημαίνει	x = x + 5;
k *= 7;	σημαίνει	k = k * 7;
sum %= 15;	σημαίνει	sum = sum % 15;

17η Άσκηση

Ποια είναι η τιμή του x μετά την εκτέλεση καθεμιάς από τις παρακάτω εντολές;

a. x=2;
b. x=3+(3>x);
c. x+=x-=2;
d. x=(++x-6)*3;
e. x*=(5>x)*(3+23);

18η Άσκηση

Τι θα εμφανιστεί στην οθόνη;

α) int a=1; printf("%d\n", 2 * ++a);

β) int a=1; printf("%d\n", 2 * a++);

Ο Τελεστής sizeof

Η C έχει έναν εσωτερικό τελεστή, τον *sizeof*, που δίνει το μέγεθος κάποιων πραγμάτων σε bytes. Αυτό θα το δούμε καλύτερα μ' ένα παράδειγμα :

```
#include <stdio.h>

main()

{

        printf("Ο τύπος int έχει μέγεθος %d bytes. \n", sizeof(int));

        printf("Ο τύπος char έχει μέγεθος %d bytes. \n", sizeof(char));

        printf("Ο τύπος long έχει μέγεθος %d bytes. \n", sizeof(long));

        printf("Ο τύπος double έχει μέγεθος %d bytes. \n", sizeof(double));

}
```

Το αποτέλεσμα θα είναι :

Ο τύπος int έχει μέγεθος 2 bytes.

Ο τύπος char έχει μέγεθος 1 bytes.

Ο τύπος long έχει μέγεθος 4 bytes.

Ο τύπος double έχει μέγεθος 8 bytes.

Τελεστές Ανάθεσης

Η ανάθεση δεν είναι εντολή αλλά τελεστής (!) που επιστρέφει την τιμή που ανατίθεται - προσοχή στις προτεραιότητες

```
i=j=n;        /* i,j are n */
i=(j=n)+1;    /* i is n+1, j is n */
if ((i=f())==j)… ⇔ i=f(); if (i==j)…
if (i=f()==j)…   ⇔ i=(f()==j); if (i)…
```

Εκτός από την συμβατική ανάθεση, υπάρχουν και άλλες εξειδικευμένες μορφές ανάθεσης που χρησιμοποιούνται συχνά

- η προσθετική ανάθεση "`x op= y`" συνδυάζει έναν δυαδικό τελεστή με τον τελεστή της ανάθεσης - είναι ισοδύναμη με την έκφραση "`x = x op y`", όμως το `x` αποτιμάται μια φορά

```
i+=2;          /* i = i+2 */
x*=y+1;        /* x = (x*y)+1   */
```

19η Άσκηση

Τι σημαίνουν οι παρακάτω εντολές;

int x =5, y = 1;

α) **x *= 2;**

β) printf("%d\n", x)

γ) x %= 2 + y

δ) printf("%d\n", x)

ε) x /= 2;

ε) printf("%d\n", x)

Προτεραιότητες

Οι προτεραιότητες και προσεταιριστικότητες των τελεστών έχουν ως εξής:

`() [] . ->`	left->right
`! ~ ++ -- + - * & (type)` --unary	right->left
`* / %`	left->right
`+ -`	left->right
`<< >>`	left->right
`> >= < <=`	left->right
`==, !=`	left->right
`&`	left->right
`^`	left->right
`|`	left->right
`&&`	left->right
`||`	left->right
`? :`	right->left
`= += -= *= /= %= &= |= <<= >>=`	right->left
`,`	left->right

Η προσεταιριστικότητα είναι από δεξιά προς τα αριστερά στους ατομικούς τελεστές, τελεστές ανάθεσης και `"? :"` και από αριστερά προς τα δεξιά στις υπόλοιπες περιπτώσεις, π.χ.

$y=x=e_1?e_2:e_3\ ,e_4 \quad \Leftrightarrow \quad (y=(x\ =(e_1?e_2:e_3))),e_4$

Μετατροπή Τύπων

Μετατροπή τύπου γίνεται όταν μια τιμή τύπου x πρέπει να γίνει συμβατή με μεγέθη τύπου y

- μετατροπές τύπων συμβαίνουν αυτόματα (!) όταν αποτιμώνται εκφράσεις (και αναθέσεις):
  - τιμές `char` και `short` μετατρέπονται σε `int`

- τιμές `float` μετατρέπονται σε `double` (παραδοσιακή C)
- αν ένα από τα ορίσματα μιας πράξης που θέλει ορίσματα ίδιου τύπου, είναι `long double`, `double`, `float`, `long`, `unsigned`, `int` (με αυτή τη σειρά) τότε και το άλλο όρισμα προάγεται στον ίδιο τύπο

➢ προσοχή στις μετατροπές από `char->int` καθώς το `char` μπορεί να είναι `signed` και να έχουμε sign extension

```
char c='\xff';
int i=c;        /* i is 255 or -1 ? */
```

- μετατροπές γίνονται και όταν μια τιμή ανατίθεται σε μεταβλητή μικρότερου μεγέθους οπότε χάνεται μέρος των δεδομένων (τα most significant bits)

```
int i=256;
char c;
c=i; i=c;  /* i is 0 */
```

➢ είναι ευθύνη του προγραμματιστή να εντοπίσει τέτοιες "περικοπές" γιατί ο μεταφραστής δεν προειδοποιεί πάντα!

- μετατροπές τύπων μπορεί να γίνουν και από τον προγραμματιστή με casting (με `void` μπορεί να αγνοηθεί η τιμή που επιστρέφει μια συνάρτηση)

```
int i=(unsigned char)'\xff';/* i is 255 */
int i=(unsigned int) '\xff';/* i is -1 */
double d=(double)10/3;      /* d is 3.33… */
double d=10/3;              /* d is 3.0 */
(void) foo();    /* ignore return value */
```

Παράδειγμα

```
/* The purpose of this program is to show type conversion */
main()
{
int a = 2;
float x = 17.1, y = 8.95, z;
char c;

   c = (char)a + (char)x;
   c = (char)(a + (int)x);
   c = (char)(a + x);
   c = a + x;

   z = (float)((int)x * (int)y);
   z = (float)((int)(x * y));
   z = x * y;

}
```

Το πρόγραμμα ξεκινά με έναν ενδιαφέροντα νεωτερισμό στις δηλώσεις των μεταβλητών. Διαπιστώνουμε οτι οι εκχωρήσεις των αρχικών τιμών, δηλαδή η αρχικοποίηση των μεταβλητών μπορεί να γίνει ταυτόχρονα με τη δήλωση των μεταβλητών, κάτι αρκετά συνηθισμένο σε προγράμματα C.

Η πρώτη ομάδα των τεσσάρων εκχωρήσεων εκτελεί την ίδια πράξη με διαφορετικούς τρόπους. Προσθέτει μιάν ακέραια μεταβλητή με μία πραγματική και τοποθετεί το αποτέλεσμα σε έναν χαρακτήρα. Η ρητή μετατροπή της τιμής μιάς μεταβλητής ή και μιάς έκφρασης γίνεται με την τοποθέτηση πριν από την μεταβλητή ή την έκφραση του νέου τύπου μέσα σε παρένθεση. Σε περίπτωση που πρόκειται γιά έκφραση τότε και η ίδια η έκφραση εγκλείεται σε ξεχωριστές παρενθέσεις. Η τελευταία από τις εκχωρήσεις χρησιμοποιεί εννοούμενες μετατροπές γιά να επιτύχει το ίδιο εξαγόμενο. Το αποτέλεσμα της κάθε εκχώρησης μπορεί να ελεγχθεί με μία κατάλληλη συνάρτηση printf. Θα διαπιστώσουμε οτι το αποτέλεσμα είναι ίδιο σε όλες τις περιπτώσεις.

Η δεύτερη ομάδα εκχωρήσεων παρουσιάζει τον συνδυσαμό ακέραιων και πραγματικών αριθμών. Είναι προφανές οτι οι τρείς εκφράσεις δεν παράγουν το ίδιο ακριβώς αποτέλεσμα. Το γινόμενο των ακεραίων μερών δύο πραγματικών διαφέρει από το ακέραιο μέρος του γινομένου των πραγματικών. Ειδικά στην περίπτωση της διαίρεσης η ύπαρξη ενός πραγματικού τελουμένου επιβάλλει πραγματική διαίρεση εκτός και αν ρητά δηλωθεί το αντίθετο

Παράδειγμα

Ποιο είναι το αποτέλεσμα της πράξης;

(int) 5.7 + 4

Απάντηση

5 + 4
9

20η Άσκηση

Ποιο είναι το αποτέλεσμα της πράξης;

α)(float) 10 /3
β)(float) (10/3)