Εισαγωγή στον δομημένο προγραμματισμό

Ενότητα 2η: Συντακτικό της γλώσσας C,
Μεταβλητές – Σταθερές –Τύποι Δεδομένων
Τελεστές – Εκφράσεις – Προτάσεις

Αν. καθηγητής Στεργίου Κώστας
e-mail: kstergiou@uowm.gr

Τμήμα Μηχανικών Πληροφορικής και Τηλεπικοινωνιών

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ψηφιακά Μαθήματα στο Πανεπιστήμιο Δυτικής Μακεδονίας**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

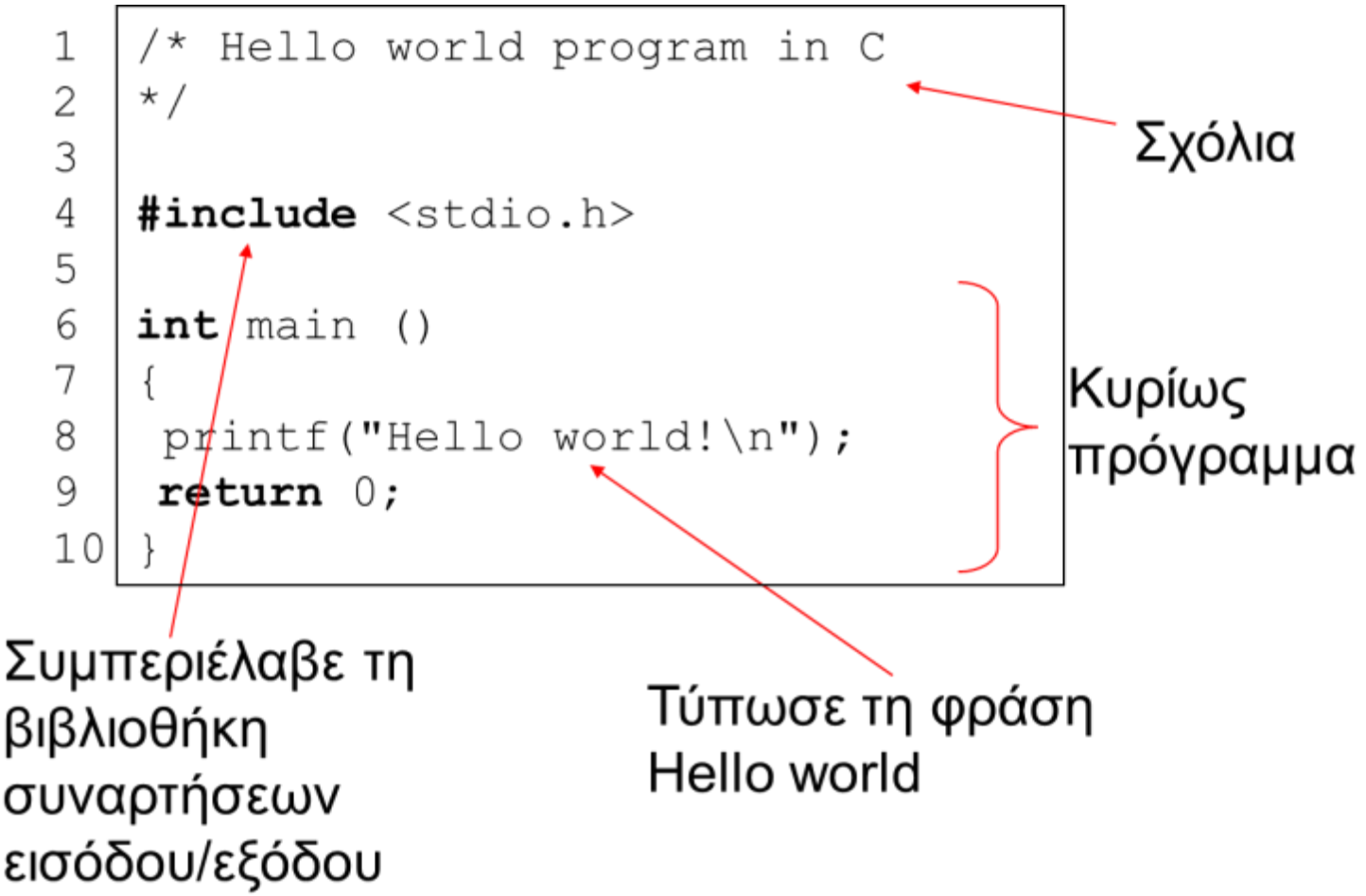
Στόχοι της διάλεξης

- Περιγραφή και ανάλυση ενός απλού προγράμματος στη C.
- Εισαγωγή στις βασικές έννοιες μια δομημένης γλώσσας προγραμματισμού.
- Εξοικείωση με τις βασικές μεθόδους εξόδου και εισόδου δεδομένων στη C.
- Προετοιμασία για τη συγγραφή των πρώτων μας προγραμμάτων στη C.

Περιεχόμενα

- Ανάλυση ενός απλού προγράμματος.
- Λεξιλόγιο και Συντακτικό της γλώσσας C.
- Δεδομένα:
  - Μεταβλητές – Σταθερές – Τύποι Δεδομένων.
- Έξοδος και Είσοδος δεδομένων:
  - printf() και scanf()
- Τελεστές – Εκφράσεις – Προτάσεις:
  - Μετατροπές τύπων.

Ένα Απλό Πρόγραμμα



Ένα Απλό Πρόγραμμα (ανάλυση) (1/5)

```
1  /*
2   Hello  world  program  in  C
3   */
```

Οτιδήποτε είναι ανάμεσα σε

```
/*
*/
```

είναι σχόλια τα οποία αφαιρούνται τελείως από τον προεπεξεργαστή.

Επίσης σχόλιο είναι κάθε γραμμή που αρχίζει με //

Ένα Απλό Πρόγραμμα (ανάλυση) (2/5)

```
4  #include <stdio.h>
```

Δίνει οδηγίες στον μεταγλωττιστή (πιο συγκεκριμένα στον προ-επεξεργαστή) να διαβάσει το αρχείο *stdio.h* το οποίο περιέχει κάποιες βοηθητικές συναρτήσεις για να εκτυπώνουμε στην οθόνη.

Στην προκειμένη περίπτωση την συνάρτηση **`printf`**() .

Ένα Απλό Πρόγραμμα (ανάλυση) (3/5)

```
6  int main()
7  {
```

και

```
11 }
```

είναι ο ορισμός της συνάρτησης **main**() η οποία υπάρχει σε κάθε πρόγραμμα C και λέει στον υπολογιστή από που να ξεκινήσει την εκτέλεση του προγράμματος.

Η συνάρτηση **main**() μόλις τελειώσει την εκτέλεση της επιστρέφει μία τιμή ακεραίου **int** στον χρήστη.

Οι αγκύλες υποδηλώνουν την αρχή και το τέλος την συνάρτησης.

Ένα Απλό Πρόγραμμα (ανάλυση) (4/5)

```
8       printf( "hello, world\n" );
```

εκτελεί μία συνάρτηση (ορισμένη στο αρχείο που διαβάστηκε με την εντολή **#include** <stdio.h>) η οποία τυπώνει στην οθόνη του υπολογιστή την ακολουθία χαρακτήρων

hello, world

το **\n** υποδηλώνει έναν ειδικό χαρακτήρα στην C που σημαίνει newline (νέα γραμμή) και υποδηλώνει στον υπολογιστή πως πρέπει να αλλάξει γραμμή στην οθόνη.

Ολόκληρη η γραμμή είναι μία **πρόταση** και κάθε πρόταση πρέπει να τερματίζεται πάντα με ένα ελληνικό ερωτηματικό.

Ένα Απλό Πρόγραμμα (ανάλυση) (5/5)

```
10      return 0;
```

εκτελεί μία πρόταση ακόμη που λέει στο πρόγραμμα να σταματήσει την εκτέλεση την συνάρτησης **`main`**() και να επιστρέψει στον χρήστη την τιμή 0.

Συμβουλές

- να χρησιμοποιείτε σχόλια ώστε ο κώδικας να είναι ευανάγνωστος και από 3ους
- μεταξύ των αγκίστρων { } ο κώδικας που εμφανίζεται πρέπει να είναι στοιχισμένος πιο δεξιά (πχ με 1 tab ή με 8 κενά) ώστε να είναι πιο ευδιάκριτος
- τα προγράμματα σας πρέπει να είναι απλά, δηλαδή να αποφεύγεται η χρήση "προγραμματιστικών κόλπων"

Παράδειγμα προς αποφυγή!

```
#include <stdio.h>

main(t,_,a)
char *a;
{return!0<t?t<3?main(-79,-13,a+main(-87,1-_,
main(-86, 0, a+1 )+a)):1 ,t<_?main(t+1, _, a ):3 ,main ( -94, -27+t, a
)&&t == 2 ?_<13 ?main ( 2, _+1, "%s %d %d\n" ):9:16:t<0?t<-72?main(_,
t,"@n'+,#'/*{}w+/w#cdnr/+,{}r/*de}+,/*{*+,/w{%+,/w#q#n+,/#{l,+,/n{n+\
,/+#n+,/#;#q#n+,/+k#;*+,/'r :'d*'3,}{w+K w'K:'+}e#';dq#'l q#'+d'K#!/\
+k#;q#'r}eKK#}w'r}eKK{nl]'/#;#q#n')()#}w'){){nl]'/+#n';d}rw' i;# ){n\
l]!/n{n#'; r{#w'r nc{nl]'/#{l,+'K {rw' iK{;[{nl]'/w#q#\
n'wk nw' iwk{KK{nl]!/w{%'l##w#' i; :{nl]'/*{q#'ld;r'}{nlwb!/*de}'c \
;;{nl'-{}rw]'/+,}##'*}#nc,',#nw]'/+kd'+e}+;\
#'rdq#w! nr'/ ') }+}{rl#'{n' ')# }'+}##(!!/")
:t<-50?_==*a ?putchar(a[31]):main(-65,_,a+1):main((*a == '/')+t,_,a\
+1 ):0<t?main ( 2, 2 , "%s"):*a=='/'||main(0,main(-61,*a, "!ek;dc \
i@bK'(q)-[w]*%n+r3#l,{}:\nuwloca-O;m .vpbks,fxntdCeghiry"),a+1);}
```

Αυτός ο κώδικας μπορεί να μεταγλωττιστεί με επιτυχία!!

Στοιχεία της Γλώσσας

- **Αλφάβητο:**
  - Οι χαρακτήρες με τους οποίους σχηματίζονται οι λέξεις της γλώσσας.
- **Λεξιλόγιο:**
  - Οι λέξεις που χρησιμοποιεί η γλώσσα.
- **Συντακτικό:**
  - Οι κανόνες σύνταξης των προτάσεων της γλώσσας.
- **Σημασιολογία:**
  - Οι κανόνες ερμηνείας των προτάσεων της γλώσσας.

```
1  /* Hello world program
2  */
3
4  #include <stdio.h>
5
6  int main ()
7  {
8  printf("Hello world!\n");
9  return 0;
10 }
```

Λεξιλόγιο της Γλώσσας

- **Δεσμευμένες λέξεις** (reserved words):
  - Λέξεις-κλειδιά (π.χ. int, if, for).
  - Ονόματα συναρτήσεων βιβλιοθήκης (π.χ. printf, isdigit).
  - Ονόματα εντολών προεπεξεργαστή (π.χ. include, define).
- **Τελεστές** (operators):
  - Ειδικά σύμβολα ή λέξεις που αναπαριστούν συγκεκριμένες διεργασίες (π.χ. +, &&).
- **Αναγνωριστές** (identifiers):
  - Αυθαίρετα ονόματα μεταβλητών, σταθερών, συναρτήσεων που καθορίζονται από τον προγραμματιστή.

```
mkd = 0;
while (mkd == 0) {
    z = x % y;
    if (z == 0) {
          mkd = y;
          return mkd;
       }
       x = y;
       y = z;
}
```

Λέξεις-Κλειδιά

- **int**: αναπαριστά τον ακέραιο τύπο δεδομένων (integer).
- **if – else**: έλεγχος ροής προγράμματος.
- Συνήθως οι πιο βασικές λέξεις-κλειδιά είναι κοινές σε πολλές γλώσσες προγραμματισμού.
- Αν και είναι περιορισμός της γλώσσας, αυξάνουν την αναγνωσιμότητα και αξιοπιστία των προγραμμάτων.

Ονόματα Αναγνωριστών

- Δεν πρέπει να είναι ίδια με τις δεσμευμένες λέξεις.
- Μπορούν να περιλαμβάνουν γράμματα, αριθμούς και το χαρακτήρα “_” .
- Πρέπει να αρχίζουν από γράμμα ή το χαρακτήρα “_”.
- Διάκριση μεταξύ μικρών και κεφαλαίων.
- Όριο μήκους (π.χ. 31 για ANSI C).

Παραδείγματα

name1 ~~1name~~ ~~get-word~~ get_word ~~int~~ mAxVel

Αναγνωσιμότητα Προγραμμάτων

```
i = 120;
if (i > j)
  function1();
else
  function2();

velocity = 120;
if (velocity >
  max_velocity)
  decrease_velocity();
else
  increase_velocity();
```

Πρόγραμμα 1

Πρόγραμμα 2

Γενικοί Κανόνες για Ευανάγνωστα Προγράμματα

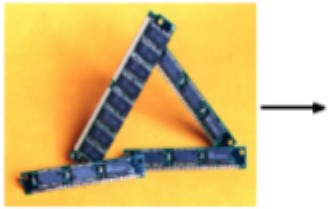
- Αποφύγετε ονόματα ενός χαρακτήρα (i, j) εκτός από συγκεκριμένες περιπτώσεις.
- Χρησιμοποιείτε εκφραστικά ονόματα:
  - Αναπαράσταση μέγιστης ταχύτητας: max_velocity ή maxVelocity.
  - Συνάρτηση εμφάνισης λάθους στην οθόνη: display_error ή displayError.
- Χρησιμοποιείστε μικρά γράμματα για τις μεταβλητές και κεφαλαία για τις μακροεντολές.

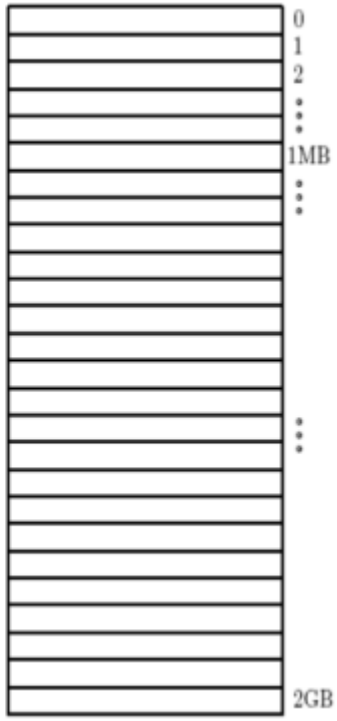
Συντακτικά-Σημασιολογικά Λάθη

- Συντακτικά λάθη:
  - Οφείλονται σε παραβίαση των συντακτικών κανόνων της γλώσσας.
  - Ανιχνεύονται στη διάρκεια της μεταγλώττισης.
- Σημασιολογικά λάθη:
  - Οφείλονται στην κακή απόδοση της λύσης του προβλήματος.
  - Ανιχνεύονται στο χρόνο εκτέλεσης (λάθος αποτελέσματα).

Δεδομένα

- Ένα υπολογιστικό πρόγραμμα διαχειρίζεται πληροφορία που αποθηκεύεται στη **μνήμη.**
- Η πληροφορία μπορεί να έχει τη μορφή αριθμητικών δεδομένων, γραμμάτων ή συμβόλων που αποτελούν τα δεδομένα εισόδου.
- Κατά τη διάρκεια διαχείρισης των δεδομένων προκύπτουν:
  - Τα τελικά αποτελέσματα που περιμένει ο χρήστης.
  - Ενδιάμεσα αποτελέσματα που χρησιμοποιούνται για την παραγωγή των τελικών.





Η μνήμη είναι σαν πίνακας όπου κάθε στοιχείο αντιστοιχεί σε έναν Αριθμό.

Μεταβλητές (1/5)

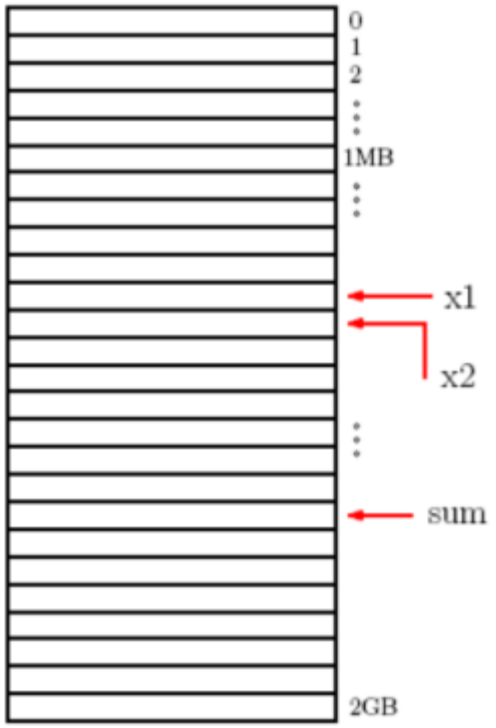
- Οι γλώσσες προγραμματισμού υποστηρίζουν την πρόσβαση στα δεδομένα μέσω συμβολικών ονομάτων που καλούνται μεταβλητές:
  - Έτσι μπορούμε να προσπελάσουμε την μνήμη του Η/Υ χωρίς να θυμόμαστε αριθμούς και διευθύνσεις.
- Χαρακτηριστικά μεταβλητής:
  - Το **όνομά** της.
  - Ο **τύπος** της.
  - Η **τιμή** της.
- Το περιεχόμενο (τιμή) μιας μεταβλητής μπορεί να αλλάζει στη διάρκεια εκτέλεσης ενός προγράμματος.

Μεταβλητές (2/5)

```
1  /* Variable Example */
2  #include <stdio.h>
3
4  int main()
5  {
6      int x1, x2, sum;
7  }
```

Στο παραπάνω πρόγραμμα λέμε στον υπολογιστή να δεσμεύσει 3 θέσεις μνήμης για να αποθηκεύσουμε ακέραιους αριθμούς. Αυτές οι 3 θέσεις μνήμης πρέπει να ονομάζονται **x1**, **x2** και **sum** αντίστοιχα.

Ουσιαστικά ο compiler θυμάται τις διευθύνσεις με τα παραπάνω ονόματα.



Ουσιαστικά τα ονόματα είναι συντομεύσεις για τις διευθύνσεις.

Μεταβλητές (3/5)

- Οι μεταβλητές αποτελούν αναφορά:
  - Σε μια θέση μνήμης.
  - Στο περιεχόμενο της θέσης μνήμης.
  - `count = count + 1;`
- Οι μεταβλητές παριστάνουν οντότητες του προβλήματος:
  - Πρόβλημα: υπολογισμός επιφάνειας ορθογωνίου.
  - Οντότητες: πλάτος, μήκος, εμβαδόν.
  - Μεταβλητές: platos, mikos, embadon.

Μεταβλητές (4/5)

```
1  /* Variable Example */
2  #include <stdio.h>
3
4  int main()
5  {
6      int x1, x2, sum;
7  }
```

Οι θέσεις μνήμης που δεσμεύτηκαν δεν έχουν αρχικοποιηθεί, είναι καθήκον του προγραμματιστή να δώσει αρχικές τιμές.

Μεταβλητές (5/5)

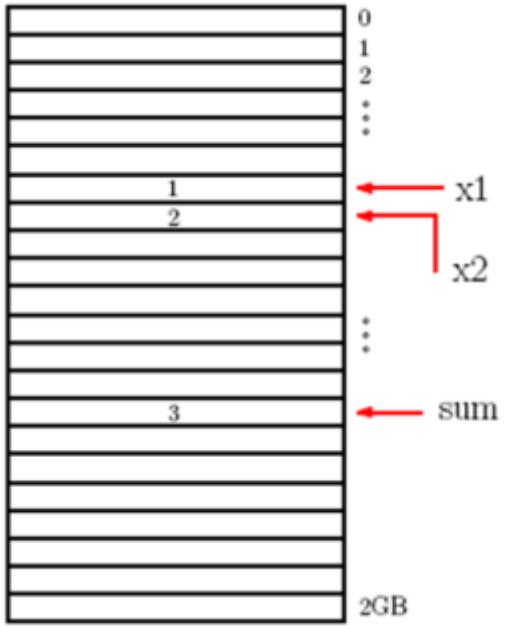
```
1  /* Variable Example */
2  #include <stdio.h>
3
4  int main()
5  {
6      int x1, x2, sum;
7
8      x1 = 1;
9      x2 = 2;
10     sum = x1 + x2;
11 }
```

τύπος

όνομα

τιμή

Το παραπάνω πρόγραμμα αφού δεσμεύσει την μνήμη για τις μεταβλητές, αποθηκεύει την τιμή 1 στην θέση μνήμης x1, την τιμή 2 στην θέση μνήμης x2 και την τιμή 3 στην θέση μνήμης sum.



Τα ονόματα είναι συντομεύσεις για τις διευθύνσεις.

Ονόματα Μεταβλητών

Για να είναι αποδεκτό από τον μεταγλωττιστή ένα όνομα στη C πρέπει να ξεκινάει με κάποιο χαρακτήρα (και όχι αριθμό), να μην περιέχει κενά και να μην έχει το ίδιο όνομα με κάποια συγκεκριμένα αλφαριθμητικά που χρησιμοποιεί η C όπως **main** (δεσμευμένες λέξεις).

λάθος ορισμός μεταβλητής

```
int main;
int 3x;
int hello world;
```

σωστός ορισμός μεταβλητής

```
int Main;
int x123456;
int hello_world;
```

Τύποι Δεδομένων (1/2)

- Όταν ορίζουμε μια μεταβλητή πρέπει να πούμε στον μεταγλωττιστή τι είδους (τι τύπο) πληροφορίας θα αποθηκευτεί στη θέση μνήμης που θα κρατηθεί.
  - Ένας τύπος δεδομένων προσδιορίζει το πεδίο τιμών μιας μεταβλητής και τις πράξεις που μπορούν να γίνουν σε αυτή.
- Η C απαιτεί την δήλωση του τύπου των μεταβλητών από τον προγραμματιστή.
- **Ενσωματωμένοι τύποι** (ακέραιοι, πραγματικοί).
- **Παραγόμενοι τύποι** (δομές).

Τύποι Δεδομένων (2/2)

- Δηλώνοντας το σωστό τύπο για τις μεταβλητές:
  - Επιτυγχάνουμε καλύτερη εκμετάλλευση της μνήμης.
  - Επιτυγχάνουμε καλύτερο έλεγχο κατά τη μεταγλώττιση του προγράμματος.

- **Πρωτογενείς τύποι**:
  - Ακέραιος (int).
  - Πραγματικός απλής ακρίβειας (float).
  - Πραγματικός διπλής ακρίβειας (double).
  - Χαρακτήρας (char).

Δήλωση Μεταβλητών στη C (1/2)

- <τύπος δεδομένων> <λίστα μεταβλητών>;

```
int count;
int count, num;
```

- <τύπος δεδομένων> <μεταβλητή>=<τιμή>;

```
int count = 20;
float num = 0.2;
```

Δήλωση σταθερών (δεν αλλάζουν τιμή)

- const <τύπος δεδομένων> <μεταβλητή>=<τιμή>;

```
const float pi = 3.14;
```

Δήλωση Μεταβλητών στη C (2/2)

Στη C μεταβλητές και οι τύποι τους πρέπει να ορίζονται στην αρχή του κομματιού κώδικα όπου θα χρησιμοποιηθούν.

```
1.  int main ()
2.  {
3.   int x,y,z;
4.    x = 2;
5.    y = 3;
6.    z = y-x;
7.   return 0;
8.  }
```

ΣΩΣΤΟ

```
1.  int main ()
2.  {
3.   int x,y;
4.    x = 2;
5.    y = 3;
6.    int z;
7.    z = y-x;
8.   return 0;
9.  }
```

ΛΑΘΟΣ!

Αρχικοποίηση Μεταβλητών

- Με τη δήλωση μιας μεταβλητής γίνεται δέσμευση θέσης μνήμης από τον compiler.
  - Η αρχικοποίηση (ανάθεση αρχικής τιμής) στην μεταβλητή πρέπει να γίνει από τον προγραμματιστή.
    - Αυτό μπορεί να γίνει μαζί με την δήλωση ή αργότερα στο πρόγραμμα.
    - **Προσοχή:** Αν προσπαθήσετε να χρησιμοποιήσετε μεταβλητή που δεν έχει πάρει τιμή θα λάβετε αναπάντεχα αποτελέσματα.

```
1.  int main ()
2.  {
3.   int x,y,z;
4.    x = 2;
5.    y = 3;
6.    z = y-x;
7.   return 0;
8.  }
```

ΣΩΣΤΟ

```
1.  int main ()
2.  {
3.   int x=2,y=3,z;
4.    z = y-x;
5.   return 0;
6.  }
```

ΣΩΣΤΟ

```
1.  int main ()
2.  {
3.   int x,y,z;
4.    y = 3;
5.    z = y-x;
6.   return 0;
7.  }
```

ΛΑΘΟΣ

Τύπος Χαρακτήρα (char)

- Αναπαριστά απλούς χαρακτήρες.
- Η τιμή του δηλώνεται σε απλά εισαγωγικά:
  - `'C', '2', '*', ','`
- Λέξη-κλειδί: `char`
  - `char choice='A';`
  - `char x, y;`
- Απαιτεί 1 byte μνήμης.
- Στη C κάθε χαρακτήρας χειρίζεται ως ακέραιος με τιμή τον αντίστοιχο κωδικό ASCII.

```
1.  int main ()
2.  {
3.   char c1,c2;
4.    c1 = 'A';
5.    c2 = 'B';
6.   return 1;
7.  }
```

```
 !"#$%&'()*+,-./
0123456789:;<=>?
@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_
`abcdefghijklmno
pqrstuvwxyz{|}~
```

Οι 95 εκτυπώσιμοι χαρακτήρες από 32 έως 126 (δεκαδικό). Οι χαρακτήρες από 0 έως 31 είναι ειδικοί χαρακτήρες για τον έλεγχο των συσκευών εξόδου, πχ το 8 είναι το BACKSPACE, το 13 είναι το ENTER και το 27 είναι το ESC.

Τύπος Ακεραίου (integer)

- Αναπαριστά ακεραίους (προσημασμένους ή μη).
- Λέξη-κλειδί: `int`.
- Πεδίο τιμών ανάλογα με το μήκος λέξης του υπολογιστή:
  - Λέξη 16 bit: -32767 έως +32768 ή
    0 έως 65535 για απρόσημους.
  - Σε πολλά σύγχρονα συστήματα μπορεί να πάρει όλες τις τιμές στο διάστημα [-2147483647, 2147483647].
- Προσδιοριστές:
  - Μικρότερης/Μεγαλύτερης ακρίβειας: `short/long`,
  - Απρόσημοι: `unsigned`,
  - `unsigned int count;`

Τύπος Πραγματικού (real)

- Αναπαριστά (προσεγγιστικά) πραγματικούς αριθμούς:
  - Σταθερής υποδιαστολής (0.012).
  - Κινητής υποδιαστολής (6.3e-05).
- Λέξεις-κλειδιά:
  - `float`: Απλής ακρίβειας (6-7 δεκαδικά ψηφία).
  - `double`: Διπλής ακρίβειας (14-15 δεκαδικά ψηφία).
- Προσδιοριστές: `long double`
- Αποθήκευση (δεν ορίζεται σαφώς στη C):
  - float: 32 bits,
  - double: 64 bits.

```
1. int main ()
2. {
3.  float pi;
4.  pi = 3.141592
5.  return 0;
6. }
```

Τύποι και όρια

τύπος	ελάχιστη τιμή	μέγιστη τιμή
`char`	≤ -127	$\geq +127$
`unsigned char`	≤ 0	$\geq +255$
`short int`	≤ -32767	$\geq +32767$
`unsigned short int`	≤ 0	$\geq +65535$
`int`	≤ -32767	$\geq +32767$
`unsigned int`	≤ 0	$\geq +65535$
`long`	≤ -2147483647	$\geq +2147483647$
`unsigned long`	≤ 0	$\geq +4294967295$

τύπος	ελάχιστη τιμή	μέγιστη τιμή
`float`	$\leq -1E-37$	$\geq 1E+37$
`double`	$\leq -1E-37$	$\geq 1E+37$

Οι σύγχρονοι compilers έχουν μεγαλύτερα όρια από αυτά (πρότυπο ANSI C).

Σταθερές (constants) (1/3)

- Οι σταθερές είναι ποσότητες που δεν αλλάζουν κατά τη διάρκεια εκτέλεσης του προγράμματος.
- Υπάρχουν διάφορα είδη σταθερών:
  - **Κυριολεκτικές σταθερές:**
    - 12, ‘A’, -3478.34
  - **Συμβολικές σταθερές:**
    - Ορίζονται με την εντολή #define και τις επεξεργάζεται ο προεπεξεργαστής.
  - **Δηλωμένες σταθερές:**
    - Ορίζονται στο κυρίως πρόγραμμα με χρήση του const.

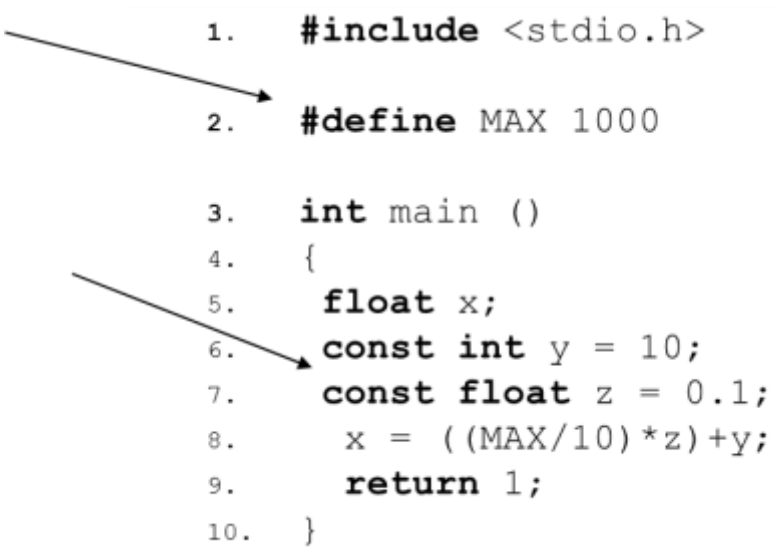
Σταθερές (constants) (2/3)

- Ο τύπος μιας κυριολεκτικής ή μιας συμβολικής σταθεράς αναγνωρίζεται άμεσα από την εμφάνισή της (δεν χρειάζεται δήλωση).
- Πραγματικές σταθερές: θεωρούνται double εκτός και αν δηλώνεται κάτι άλλο.
- **Παραδείγματα:**
  - `123` → `int`
  - `4536L` → `long int`
  - `'A'` → `char`
  - `123.5f` → `float`
  - `123.5` → `double`
  - `9e-05` → `double`

```
1.  int main ()
2.  {
3.   float pi;
4.   pi = 3.141592
5.   return 0;
6.  }
```

Σταθερές (constants) (3/3)

- Με την εντολή **#define** του προεπεξεργαστή ορίζουμε συμβολοσειρές οι οποίες αντικαθίστανται με τις αντίστοιχες τιμές κατά την εκτέλεση.
- Το **const** λέει στον μεταγλωττιστή ότι η συγκεκριμένη μεταβλητή δεν θα αλλάξει ποτέ τιμή.
- Αν προσπαθήσουμε να αλλάξουμε τιμή σε μεταβλητή δηλωμένη ως const, θα προκύψει λάθος μεταγλώττισης.



```
1.  #include <stdio.h>

2.  #define MAX 1000

3.  int main ()
4.  {
5.    float x;
6.    const int y = 10;
7.    const float z = 0.1;
8.     x = ((MAX/10)*z)+y;
9.    return 1;
10. }
```

Εκτύπωση Μεταβλητών

- Γίνεται με τη συνάρτηση **printf** της βασικής βιβλιοθήκης <stdio.h> χρησιμοποιώντας κατάλληλο προσδιοριστή ανάλογα με τον τύπο της μεταβλητής.
- `printf("περιγραφή", <μεταβλητές>);`
  - `%c`: χαρακτήρας.
  - `%d`: ακέραιος.
  - `%f`: πραγματικός κινητής υποδιαστολής (συμβολισμός με σταθερή υποδιαστολή).
  - `%e`: πραγματικός κινητής υποδιαστολής (επιστημονικός συμβολισμός).
- Παραδείγματα:

```
printf("ο χαρακτήρας είναι %c", choice);
printf("η ASCII τιμή είναι %d", choice);
printf("dec=%d, octal=%o, hex=%x", num, num, num);
printf("μπορεί να γραφεί ως %f και ως %e", val, val);
```

Προσδιοριστές της printf

- Οι περισσότεροι προσδιοριστές (specifiers) δίνονται στον πίνακα παρακάτω.

specifier	Έξοδος	Παράδειγμα
c	χαρακτήρας	a
d ή i	δεκαδικός αριθμός με πρόσημο	392
e	επιστημονικός συμβολισμός με e	3.9265e+2
E	επιστημονικός συμβολισμός με E	3.9265E+2
f	δεκαδικός αριθμός κινητής υποδιαστολής	392.65
o	οκταδικός με πρόσημο	610
s	αλφαριθμητικό	sample
u	δεκαδικός ακέραιος χωρίς πρόσημο	7235
x	δεκαεξαδικός αριθμός χωρίς πρόσημο	7fa
X	δεκαεξαδικός αριθμός χωρίς πρόσημο με κεφαλαία	7FA

Η γραμμή

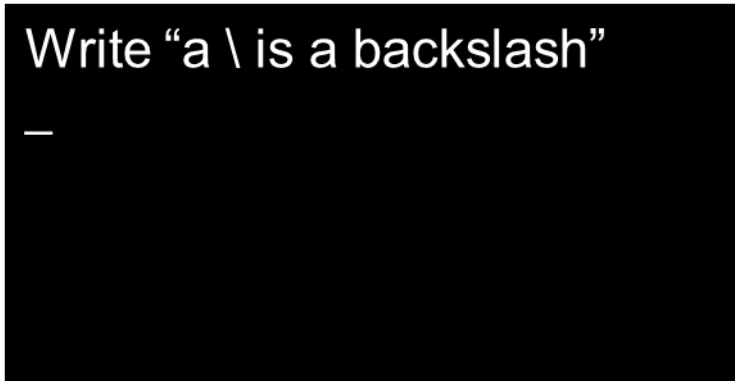
```
printf("%d, %o and %x\n", 27, 27, 27);
```

εκτυπώνει 27, 33 and 1b.

Ειδικοί χαρακτήρες της C

- Σταθερές τύπου χαρακτήρα που δεν τυπώνονται:
  - Νέα γραμμή: `'\n'`
  - Στηλοθέτης: `'\t'`

- Άλλοι χαρακτήρες:
  - backslash `'\\'`
  - single quotation mark `'\''`
  - double quotation mark `'\"'`



- Χρήση του χαρακτήρα διαφυγής:
  - `printf("Write \"a \\ is a backslash\"\n");`

Είσοδος Δεδομένων (1/3)

Παράδειγμα: πρόσθεση δύο ακέραιων αριθμών.

```
1.  #include <stdio.h>

2.  int main ()
3.  {
4.   int num1,num2,sum;

5.   printf("Enter first number\n");
6.   scanf("%d",&num1);
7.   printf("Enter second number\n");
8.   scanf("%d",&num2);
9.   sum = num1 + num2;
10.  printf("The sum of the numbers is %d\n",sum);
11.  return 1;
12. }
```

Είσοδος Δεδομένων (2/3)

- Γίνεται με τη scanf συνάρτηση της <stdio.h> βιβλιοθήκης **χρησιμοποιώντας κατάλληλο προσδιοριστή** όπως και στην printf.
- Η εκτέλεση της εντολής:

```
scanf("%d",&num1);
```

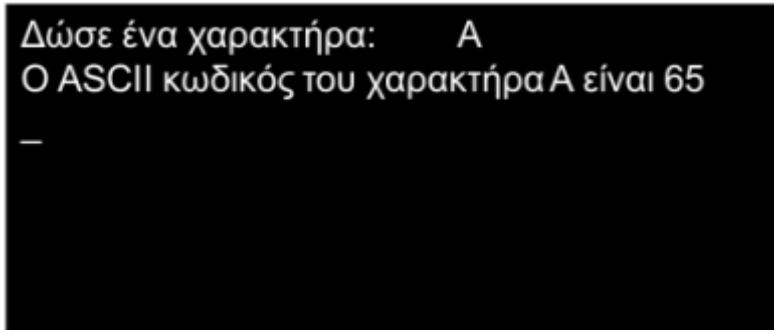
σταματάει την εκτέλεση του προγράμματος μέχρι ο χρήστης να πληκτρολογήσει έναν ακέραιο και να πατήσει ENTER.

- `scanf("<προσδιοριστής>", &<μεταβλητή>);`
  - `scanf("%c", &var);`
  - `scanf("%d", &var);`
  - `scanf("%f", &var);`
  - `scanf("%e", &var);`

Τελεστής διεύθυνσης

Παράδειγμα

```
#include <stdio.h>
void main ()
{
char ch;
printf("Δώσε ένα χαρακτήρα:\t");
scanf("%c", &ch);
printf("\nO ASCII κωδικός του χαρακτήρα %c
  είναι %d\n", ch, ch);
}
```



Είσοδος Δεδομένων (3/3)

Προσδιοριστές (specifiers) της συνάρτησης `scanf()`

specifier	Έισοδος	Παράδειγμα
d	διάβασε ένα ακέραιο ως προσημασμένο δεκαδικό	392
c	χαρακτήρας	a
f	δεκαδικός αριθμός κινητής υποδιαστολής	392.65
x	δεκαεξαδικός αριθμός χωρίς πρόσημο	7fa
o	οκταδικός αριθμός	610
s	αλφαριθμητικό	sample
(space)	διαβάζει κενά	
lf	διαβάζει έναν **`double`**	1.333
Lf	διαβάζει έναν **`long double`**	1.333

Τελεστές (operators)

- Σύμβολα ή λέξεις που αναπαριστούν μία συγκεκριμένη διεργασία που εκτελείται σε ένα ή περισσότερα δεδομένα.
- Τα δεδομένα καλούνται τελεστέοι:
  - Μεταβλητές.
  - Σταθερές.
  - Κλήσεις συναρτήσεων.
- Οι τελεστές χρησιμοποιούνται για το σχηματισμό εκφράσεων (π.χ. μαθηματικών εκφράσεων):
  - num + 12.
  - num1 > num2.

Κατηγορίες Τελεστών

- **Αριθμητικοί** (+, -, *, /, %).
- **Ανάθεσης** (=, +=).
- **Λογικοί** (&&, ||, !).
- **Συσχετιστικοί** (>, >=, ==, !=).
- **Διαχείρισης bits** (>>, &, |, ^, ~).
- **Διαχείρισης μνήμης** (&, *, ., ->).
- **Ειδικοί.**

Αριθμητικοί Τελεστές

Τα περισσότερα προγράμματα στη C εκτελούν μαθηματικούς υπολογισμούς .

```
1. #include <stdio.h>

2. int main ()
3. {
4.  int x,y,z;
5.  z = 10;
6.  x = z - 5;
7.  y = x + z;
8.  return 1;
9. }
```

Δυαδικοί αριθμητικοί τελεστές

	τελεστής	αλγεβρική έκφραση	C
πρόσθεση	+	$x + 7$	`x + 7`
αφαίρεση	-	$p - c$	`p - c`
πολλαπλασιασμός	*	bm	`b * m`
διαίρεση	/	x/y ή $x \div y$	`x / y`
υπόλοιπο διαίρεσης	%	$r \bmod s$	`r % s`

Στην C οι παρενθέσεις χρησιμοποιούνται περίπου όπως στις αλγεβρικές εκφράσεις

x = (y+z)*(w+f);

Τελεστής Ανάθεσης (1/2)

- Στο αριστερό μέρος καταχωρείται η τιμή του δεξιού μέρους.
- <μεταβλητή> = <έκφραση>;

```
mikos = 5;
embadon = mikos * platos;
x = x + 1;
```

- Ο τύπος της μεταβλητής πρέπει να είναι συμβατός με το αποτέλεσμα της έκφρασης.

  int x = 0.1*14.45 **ΛΑΘΟΣ!**

- Ο τελεστής '==' ελέγχει την ισότητα.

Προτεραιότητα τελεστών (1/2)

- Στην C υπάρχει συγκεκριμένη προτεραιότητα των τελεστών που καθορίζει τον τρόπο υπολογισμού εκφράσεων:
  1. **παρενθέσεις**: ( )
     - Υπολογίζονται πρώτα, από τα αριστερά προς τα δεξιά. Αν υπάρχουν ένθετες υπολογίζονται πρώτα αυτές.
  2. **πολλαπλασιασμός, διαίρεση, υπόλοιπο**: * , / , %
     - Υπολογίζονται μετά, από αριστερά προς δεξιά.
  3. **πρόσθεση, αφαίρεση**: + , −
     - Υπολογίζονται μετά, από αριστερά προς δεξιά.
  4. **ανάθεση**: =
     - Από δεξιά προς αριστερά.

```
x = ((y+z)/5)*((w+f)%4)-20;
```

Εκφράσεις και προτεραιότητα τελεστών

$$x = \frac{y+z+w+f}{k}$$

Σε C

```
x = (y+z+w+f)/k;
```

Τι θα γίνει αν παραλείψουμε τις παρενθέσεις;

$x = ay^2 + bz + c$

Σε C

```
x = a*y*y+b*z+c;
```

Εδώ γιατί δεν χρειάζονται παρενθέσεις;

Σε C

```
x = a*y/z+b%w-f;
```

Ποια είναι η προτεραιότητα των τελεστών;

```
x = a * y / z + b % w - f;
```

6 1 2 4 3 5

Ειδικοί Μοναδιαίοι Αριθμητικοί Τελεστές (1/2)

- **Μοναδιαίας αύξησης/μείωσης (++, --)**

  ++x; ή x++; → x = x + 1;

  --x; ή x--; → x = x – 1;

- Οι μοναδιαίοι αριθμητικοί τελεστές έχουν μεγαλύτερη προτεραιότητα από τους άλλους αριθμητικούς (εκτός από τις παρενθέσεις) και υπολογίζονται από τα δεξιά προς τα αριστερά.

```
int x = 1;
                    x
++x;                2
x++;                3
x--;                2
--x;                1
```

Ειδικοί Μοναδιαίοι Αριθμητικοί Τελεστές (2/2)

- Χρειάζεται προσοχή στην χρήση του μαζί με ανάθεση:
  - y = x++; Αυξάνει το x μετά την ανάθεση.
  - y = ++x; Αυξάνει το x πριν την ανάθεση.

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

x	y
2	1
3	3
2	3
1	1
0	2
1	3

Προτεραιότητα τελεστών (2/2)

- Στην C υπάρχει συγκεκριμένη προτεραιότητα των τελεστών που καθορίζει τον τρόπο υπολογισμού εκφράσεων:
  1. **παρενθέσεις**: ( )
     - Υπολογίζονται πρώτα, από τα αριστερά προς τα δεξιά. Αν υπάρχουν ένθετες υπολογίζονται πρώτα αυτές.
  2. **μοναδιαίοι αριθμητικοί τελεστές**: −−, ++
     - Υπολογίζονται από δεξιά προς τα αριστερά.
  3. **πολλαπλασιασμός, διαίρεση, υπόλοιπο**: *, /, %
     - Υπολογίζονται μετά, από αριστερά προς δεξιά.
  4. **πρόσθεση, αφαίρεση**: +, −
     - Υπολογίζονται μετά, από αριστερά προς δεξιά.
  5. **ανάθεση**: =
     - Από δεξιά προς αριστερά.

Τελεστής Ανάθεσης (2/2)

- Ο τελεστής ανάθεσης εκτελείται από δεξιά προς τα αριστερά.
- Χρησιμοποιείται συχνά και μέσα σε εκφράσεις.
  - μια έκφραση όπως `x = 1` επιστρέφει την τιμή που ανατέθηκε στην μεταβλητή `x`.

```
1.  #include <stdio.h>

2.  int main ()
3.  {
4.   int x,y,z;
5.   z = 10;
6.   x = y = z;
7.   printf("%d %d %d",x,y,z);
8.   return 1;
9.  }
```

Η έκφραση αυτή υπολογίζεται μετά την ανάθεση της τιμής του z στην y

Τελεστές σύνθετης ανάθεσης

- Η C παρέχει διάφορους τελεστές ανάθεσης πέρα του βασικού, για συντόμευση των λειτουργιών ανάθεσης.
- **Τελεστές σύνθετης ανάθεσης** `(+=, -=, *=, /=, %/).`

```
x += 10;  → x = x + 10;
x -= 10;  → x = x - 10;
x *= 10;  → x = x * 10;
x /= 10;  → x = x / 10;
x %= 10;  → x = x % 10;
```

```
int x = 10;
int y = 20;          x    y
++x;                 11   20
y = --x;             10   10
y += x--;            9    20
y -= x++;            10   11
```

  - Οι τελεστές σύνθετης ανάθεσης έχουν ίδια προτεραιότητα με τον τελεστή ανάθεσης και υπολογίζονται από τα δεξιά προς τα αριστερά.

Εκφράσεις

- Συνδυασμός ενός ή περισσοτέρων τελεστέων και ενός ή περισσοτέρων τελεστών.

- Αριθμητικές (αποτέλεσμα αριθμητικού τύπου).

```
(5*x+y/4)*8
```

- Σύγκρισης (αποτέλεσμα λογικού τύπου).

```
x==3      a!=b     (x+y) >= 4
```

- Λογικές (αποτέλεσμα λογικού τύπου).

```
(x<5) && (x>=1)          (x==0) || (y==0)
```

Υπολογισμός Εκφράσεων

- **Προτεραιότητα.**
  - Χωρισμός των τελεστών σε ομάδες διαφορετικού επιπέδου προτεραιότητας:

    (+, -): χαμηλότερο επίπεδο.

    (*, /): υψηλότερο επίπεδο.

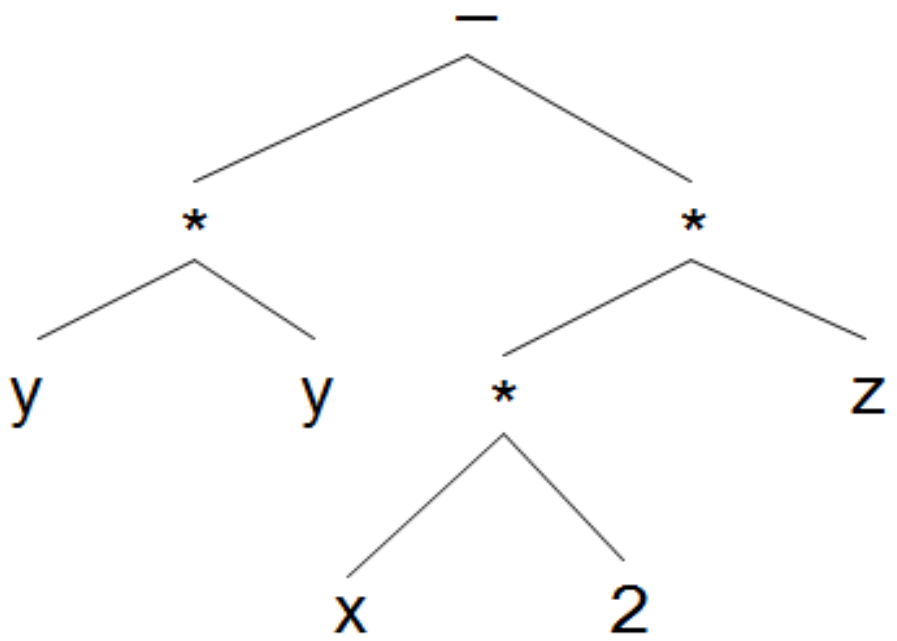
    `x-y*z` → `x-(y*z)`
- **Προσεταιριστικότητα.**
  - Καθορισμός κατεύθυνσης εφαρμογής τελεστών ίδιας προτεραιότητας.
  - Αριστερή προσεταιριστικότητα:

    10+8-2 → (10+8)-2
  - Δεξιά προσεταιριστικότητα:

    num1=num2=10 → num1=(num2=10)

Δέντρο Αφηρημένης Σύνταξης

- y*y-x*2*z → (y*y)-((x*2)*z)



Μετατροπές Τύπων (1/2)

Εκφράσεις που περιέχουν μεταβλητές και σταθερές διαφορετικών τύπων δίνουν ως αποτέλεσμα τιμή του “ευρύτερου τύπου”:

- (char < int < long < float < double).
- Μια αριθμητική πράξη μεταξύ int και double δίνει αποτέλεσμα double.

```
int main (void)
{
 int x = 3;
 float y = 3.5, z;
 z = x + y;
 return 1;
}
```

6.5

```
#include <stdio.h>

int main (void)
{
 int x = 3, y = 7;
 printf("%d %d\n",x/y,y/x);
 return 1;
}
```

Τι τυπώνεται;

Μετατροπές Τύπων (2/2)

- Ανάθεση μιας τιμής ενός τύπου σε μεταβλητή άλλου τύπου έχει ως αποτέλεσμα την μετατροπή της τιμής στον τύπο της μεταβλητής, είτε:
  - **χωρίς απώλεια πληροφορίας**, αν η μεταβλητή έχει ευρύτερο τύπο από την τιμή
  - **με απώλεια πληροφορίας**, αν η τιμή έχει ευρύτερο τύπο από την μεταβλητή.
    - Αυτό συχνά είναι επικίνδυνο!
- **Υπονοούμενη μετατροπή τύπου**
  - Ο στενότερος τύπος μετατρέπεται στον ευρύτερο
  - (char < int < long < float < double)

    float f;

    int i;

    f = i = 3.3; → το i παίρνει τιμή 3 και το f παίρνει τιμή 3.0

```
int x = 5;
float y = x;
```

```
float x = 5.25;
int y = x;
```

- **Ρητή μετατροπή τύπου (cast)**

  (<τύπος δεδομένων>)έκφραση

  (float)2 → 2.0

  (float)3/2 → 1.5

  (float)(3/2) → 1.0

```
float x = 5.25;
int y;
y = (int) x;
```

Προτάσεις

- Πρόταση είναι μια πλήρης εντολή που προσδιορίζει την εκτέλεση ενός συγκεκριμένου έργου.
- Το τέλος κάθε πρότασης προσδιορίζεται με το ελληνικό ερωτηματικό πρόταση1;
- Σύνθετη πρόταση: ένα σύνολο προτάσεων που περικλείονται σε αγκύλες ({}),

  {πρόταση1; πρόταση2; πρόταση3;}.

Κατηγορίες Προτάσεων

- Δήλωσης.

```
int num;
```

- Κλήσης συνάρτησης.

```
printf("Hello world");
```

- Ελέγχου ροής.

```
if (a>b) then a else b;
```

- Ανάθεσης.

```
num=20;
```

- Σύνθετη.

```
{num=max(12,13);
 printf("num = %d\n", num);}
```

Προτάσεις Προεπεξεργαστή

- Ειδικές προτάσεις (δεν τελειώνουν σε ;).
- Εκτελούν κάποια επεξεργασία πριν τη μεταγλώττιση.
- Πρόταση συμπερίληψης.

```
#include <stdio.h>
#include “mylib.h”
```

- Πρόταση μακρο-αντικατάστασης.

```
#define PI 3.141592654
```

Παράδειγμα 1ο

- Να γραφεί πρόγραμμα που να διαβάζει δύο ακέραιους από το πληκτρολόγιο και στη συνέχεια τυπώνει το άθροισμα, τη διαφορά, το γινόμενο, πηλίκο και το υπόλοιπο της διαίρεσης.

```
Dwse 2 akeraious (xwrismenous me keno):
```

```
#include <stdio.h>
void main()
{
  int a1, a2;
  printf("Dwse 2 akeraious (xwrismenous me keno): ");
  scanf("%d %d",&a1,&a2);
  printf("To athroisma einai %d\nH diafora einai %d\n To
        ginomeno einai %d\nTo piliko einai %d\nTo
        ypoloipo einai %d\n", a1+a2, a1-a2, a1*a2,
        a1/a2,  a1%a2);
}
```

Παράδειγμα 2ο

- Να γραφεί πρόγραμμα που να διαβάζει από το πληκτρολόγιο ένα πραγματικό αριθμό που αντιστοιχεί στη θερμοκρασία σε βαθμούς Φαρενάιτ και να τη μετατρέπει σε βαθμούς Κελσίου με βάση τον τύπο C=(5/9)*(F-32) .

```
#include <stdio.h>
void main()
{
    float F;
    printf("Dwse thermokrasia se Farenheit: ");
    scanf("%f",&F);
    printf("H thermokrasia se bathmous Kelsiou einai %f",
        (5.0/9.0)*(F-32));
}
```

Τέλος Ενότητας

Σημείωμα Αναφοράς

- Copyright Πανεπιστήμιο Δυτικής Μακεδονίας, Τμήμα Μηχανικών Πληροφορικής και Τηλεπικοινωνιών, Στεργίου Κωνσταντίνος. «Εισαγωγή στον Δομημένο Προγραμματισμό». Έκδοση: 1.0. Κοζάνη 2015. Διαθέσιμο από τη δικτυακή διεύθυνση:

  https: //eclass.uowm.gr/courses/ICTE258/

Σημείωμα Αδειοδότησης

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.