Διάλεξη 7η: Συναρτήσεις και Μεταβλητές

Τμήμα Επιστήμης Υπολογιστών, Πανεπιστήμιο Κρήτης

Εισαγωγή στην Επιστήμη Υπολογιστών

Βασίζεται σε διαφάνειες του Κ. Παναγιωτάκη

Συναρτήσεις

- Επικοινωνία με το υπόλοιπο πρόγραμμα
  - Δέχονται ορίσματα
  - Επιστρέφουν τιμή
  - Απομονώνουν την επίλυση κάποιου υποπροβλήματος
- Ιδανικά, καμία άλλη επίδραση στο υπόλοιπο πρόγραμμα
  - Συναρτησιακός προγραμματισμός
- Μπορεί όμως να επηρεάζουν το υπόλοιπο πρόγραμμα όταν:
  - γράφουν καθολικές μεταβλητές
  - γράφουν μνήμη που δεν τους “ανήκει”
- Παρενέργειες (side-effects)

Μεταβλητές

- Ιδιότητες μεταβλητών
  - Εμβέλεια (scope)
    - Ποιά μέρη του προγράμματος αναγνωρίζουν την ίδια μεταβλητή (ίδια θέση μνήμης) με το ίδιο όνομα
  - Διάρκεια ύπαρξης στη μνήμη (duration)
    - Πότε μια συγκεκριμένη διεύθυνση μνήμης λαμβάνει ένα συγκεκριμένο όνομα και πότε αποδεσμεύεται αυτή η αντιστοίχηση
- Προς το παρόν αγνοούμε προγράμματα που χωρίζονται σε διαφορετικά αρχεία

Εμβέλεια

- Μια μεταβλητή μπορεί να έχει εμβέλεια
  - Καθολική (global): σε όλο το πρόγραμμα ενός αρχείου
  - Τοπική (local): μέσα στον ορισμό μιας συνάρτησης
  - Τοπική: μέσα σε ένα block κώδικα { }

Καθολικές μεταβλητές

- Δηλώνονται συνήθως στην αρχή του προγράμματος πριν και έξω από κάθε συνάρτηση

Παράδειγμα: καθολικές μεταβλητές

```
int global1, global2;
int main(void) {
  global1 = 5;
}
```

- Εμβέλεια:
  - Μπορούμε να τις προσπελάσουμε από κάθε συνάρτηση μετά τη δήλωσή τους
- Διάρκεια ύπαρξης:
  - Από την αρχή της εκτέλεσης μέχρι το τέλος του προγράμματος

Καθολικές μεταβλητές (2)

- Χρησιμοποιούνται όταν πολλές συναρτήσεις μοιράζονται δεδομένα
  - Αντί να περνάμε τα ίδια ορίσματα σε όλες τις συναρτήσεις
- Προβλήματα:
  - Μια συνάρτηση που γράφει καθολικές μεταβλητές επηρεάζει και τις υπόλοιπες συναρτήσεις που χρησιμοποιούν τις μεταβλητές
  - Μια συνάρτηση που χρησιμοποιεί καθολικές μεταβλητές εξαρτάται απο όποια άλλη συνάρτηση γράφει στις μεταβλητές
  - Είναι δύσκολο να βεβαιωθούμε ότι οι αλλαγές στις καθολικές μεταβλητές γίνονται με τη σωστή σειρά
  - Γίνεται δύσκολη η επαναχρησιμοποίηση αυτών των συναρτήσεων και σε άλλα προγράμματα
- Αποφεύγετε τις πολλές καθολικές μεταβλητές
  - Είναι απαραίτητες;
  - Πού χρειάζεται να είναι ορατή ή να υπάρχει η κάθε μεταβλητή;

Τοπικές μεταβλητές

- Δηλώνονται μέσα σε μια συνάρτηση ή block κώδικα

locals.c

```
int foo(int x)
{
  int local1, local2;
  local1 = 42;
  {
    int local3;
    local3 = 5;
  }
  return 0;
}
```

Τοπικές μεταβλητές

- Δηλώνονται μέσα σε μια συνάρτηση ή block κώδικα

Οι παράμετροι μιας συνάρτησης λειτουργούν σαν τοπικές μεταβλητές με αρχική τιμή την τιμή του ορίσματος

locals.c

```
int foo(int x)
{
  int local1, local2;
  local1 = 42;
  {
    int local3;
    local3 = 5;
  }
  return 0;
}
```

Τοπικές μεταβλητές

- Δηλώνονται μέσα σε μια συνάρτηση ή block κώδικα

locals.c

```
int foo(int x)
{
  int local1, local2;
  local1 = 42;
  {
    int local3;
    local3 = 5;
  }
  return 0;
}
```

Τοπικές μεταβλητές της συνάρτησης–διαρκούν μέχρι να επιστρέψει η `foo`

Τοπικές μεταβλητές

- Δηλώνονται μέσα σε μια συνάρτηση ή block κώδικα

locals.c

```
int foo(int x)
{
  int local1, local2;
  local1 = 42;
  {
    int local3;
    local3 = 5;
  }
  return 0;
}
```

Τοπικές μεταβλητές του block κώδικα–διαρκούν μέχρι το τέλος του block

Τοπικές μεταβλητές (2)

- Εμβέλεια:
  - Προσπελάσιμες μόνο μέσα στην ίδια συνάρτηση ή block κώδικα
- Διάρκεια ζωής:
  - Δημιουργούνται κάθε φορά που καλείται η συνάρτηση ή ξεκινά το block
  - Διαρκούν μέχρι το τέλος της συνάρτησης ή του block
- Τις περισσότερες φορές αρκούν οι τοπικές μεταβλητές
- Είναι προτιμότερη η χρήση τους στις περισσότερες περιπτώσεις
  - Συνήθως χρειάζονται μεταβλητές για να αποθηκεύσουν το “ενδιάμεσο” αποτέλεσμα ενός υπολογισμού
  - Το “τελικό” αποτέλεσμα ενός υπολογισμού είναι η τιμή που επιστρέφει η συνάρτηση
- Προσπαθείτε να χρησιμοποιείτε *μόνο* τοπικές μεταβλητές
- Προσπαθείτε να σκέφτεστε λύσεις που δεν χρησιμοποιούν καθολικές μεταβλητές

Παράδειγμα

example.c

```
double sum;

int next_of(int x)
{
  int y = x;
  y = y + 1;
  sum = sum + 1;
  return y;
}

int main(void)
{
  int x = 0;
  sum = 0;
  x = next_of(0);
  x = next_of(x);
  return 0;
}
```

Παράδειγμα

Καθολική μεταβλητή: δημιουργείται πριν αρχίσει η `main`, προσβάσιμη από όλες τις συναρτήσεις του αρχείου

example.c

```
double sum;

int next_of(int x)
{
  int y = x;
  y = y + 1;
  sum = sum + 1;
  return y;
}

int main(void)
{
  int x = 0;
  sum = 0;
  x = next_of(0);
  x = next_of(x);
  return 0;
}
```

Παράδειγμα

Παράμετρος: δημιουργείται καινούρια κάθε φορά που καλείται η συνάρτηση `next_of` και αρχικοποιείται με το όρισμα, είναι προσβάσιμη μέσα στη συνάρτηση, χάνεται στο τέλος της συνάρτησης

example.c

```
double sum;

int next_of(int x)
{
  int y = x;
  y = y + 1;
  sum = sum + 1;
  return y;
}

int main(void)
{
  int x = 0;
  sum = 0;
  x = next_of(0);
  x = next_of(x);
  return 0;
}
```

Παράδειγμα

example.c

```
double sum;

int next_of(int x)
{
  int y = x;
  y = y + 1;
  sum = sum + 1;
  return y;
}

int main(void)
{
  int x = 0;
  sum = 0;
  x = next_of(0);
  x = next_of(x);
  return 0;
}
```

Τοπική μεταβλητή: δημιουργείται καινούρια κάθε φορά που αρχίζει η συνάρτηση `next_of`, προσβάσιμη μέσα από τη συνάρτηση, χάνεται στο τέλος της συνάρτησης

Παράδειγμα

example.c

```
double sum;

int next_of(int x)
{
  int y = x;
  y = y + 1;
  sum = sum + 1;
  return y;
}

int main(void)
{
  int x = 0;
  sum = 0;
  x = next_of(0);
  x = next_of(x);
  return 0;
}
```

Η συνάρτηση επιστρέφει με το return: Οι τοπικές μεταβλητές και παράμετροι χάνονται και ελευθερώνεται η μνήμη που τις περιέχει

Παράδειγμα

example.c

```
double sum;

int next_of(int x)
{
  int y = x;
  y = y + 1;
  sum = sum + 1;
  return y;
}

int main(void)
{
  int x = 0;
  sum = 0;
  x = next_of(0);
  x = next_of(x);
  return 0;
}
```

Τοπική μεταβλητή της `main`: δημιουργείται στην αρχή της `main` και διαρκεί μέχρι το τέλος της, δεν είναι προσβάσιμη από άλλες συναρτήσεις

Στατικές μεταβλητές

- Δηλώνονται με το πρόθεμα `static`

Παράδειγμα

```
static int x = 42;
```

- Διάρκεια ύπαρξης, όπως οι καθολικές μεταβλητές:
  - Δημιουργούνται στην αρχή του προγράμματος
  - Διαρκούν μέχρι το τέλος του προγράμματος
  - Ισχύει *και για τις τοπικές*
    - Οι τοπικές `static` μεταβλητές διατηρούν την τιμή τους μεταξύ των διαφόρων κλήσεων της συνάρτησης
- Εμβέλεια:
  - Οι τοπικές `static` μεταβλητές φαίνονται μόνο μέσα στην συνάρτηση που τις δηλώνει
  - Οι καθολικές `static` μεταβλητές φαίνονται μόνο μέσα στο αρχείο που τις δηλώνει

Στατικές μεταβλητές (2)

- Πότε χρησιμοποιούνται:
  - Τοπικές: Μόνο μια συνάρτηση χρειάζεται τη μεταβλητή, αλλά πρέπει να θυμάται την τιμή της μεταξύ διαφορετικών κλήσεων
  - Καθολικές: Ένα σύνολο από συναρτήσεις χρησιμοποιούν τη μεταβλητή
    - Δηλώνουμε τη μεταβλητή ως `static` καθολική και όλες αυτές οι συναρτήσεις μπαίνουν στο ίδιο αρχείο
- Οι `static` μεταβλητές αρχικοποιούνται στην αρχή της εκτέλεσης του προγράμματος
  - Ακόμη και οι τοπικές `static` μεταβλητές

Παράδειγμα

staticlocal.c

```
void bar(void)
{
  static int static_variable = 0;
  static_variabe++;
  printf("You have called function bar() %d times\n", static_variable);
}

int main(void)
{
  bar();
  bar();
  bar();
}
```

Έξοδος

```
You have called function bar() 1 times
You have called function bar() 2 times
You have called function bar() 3 times
```

Παράδειγμα

staticlocal.c

```
void bar(void)
{
  static int static_variable = 0;
  static_variabe++;
  printf("You have called function bar() %d times\n", static_variable);
}

int main(void)
{
  bar();
  bar();
  bar();
}
```

Η μεταβλητή δημιουργείται και αρχικοποιείται όπως μια καθολική, πριν την αρχή της `main`

Έξοδος

```
You have called function bar() 1 times
You have called function bar() 2 times
You have called function bar() 3 times
```

Παράδειγμα

staticlocal.c

```
void bar(void)
{
  static int static_variable = 0;
  static_variabe++;
  printf("You have called function bar() %d times\n", static_variable);
}

int main(void)
{
  bar();
  bar();
  bar();
}
```

Είναι ορατή μόνο μέσα στη συνάρτηση `bar()` που τη δηλώνει

Έξοδος

```
You have called function bar() 1 times
You have called function bar() 2 times
You have called function bar() 3 times
```

Παράδειγμα

staticlocal.c

```
void bar(void)
{
  static int static_variable = 0;
  static_variabe++;
  printf("You have called function bar() %d times\n", static_variable);
}

int main(void)
{
  bar();
  bar();
  bar();
}
```

Την πρώτη φορά τυπώνει 1, τη δεύτερη 2, κλπ

Έξοδος

```
You have called function bar() 1 times
You have called function bar() 2 times
You have called function bar() 3 times
```

Call by value

- Στη C, κάθε κλήση συνάρτησης περνάει τα ορίσματα αντιγράφοντάς τα στις παραμέτρους
- Αλλαγές στο αντίγραφο δεν επηρεάζουν το πρωτότυπο
- Χρησιμοποιείται όταν η συνάρτηση δεν χρειάζεται να αλλάξει τα ορίσματα

Call by value

```
void say_something(int x)
{
  x++;
  printf("The answer is %d\n", x);
}

int main(void)
{
  int i = 41;
  say_something(i);
}
```

Call by value

- Στη C, κάθε κλήση συνάρτησης περνάει τα ορίσματα αντιγράφοντάς τα στις παραμέτρους
- Αλλαγές στο αντίγραφο δεν επηρεάζουν το πρωτότυπο
- Χρησιμοποιείται όταν η συνάρτηση δεν χρειάζεται να αλλάξει τα ορίσματα

Call by value

```
void say_something(int x)
{
  x++;
  printf("The answer is %d\n", x);
}

int main(void)
{
  int i = 41;
  say_something(i);
}
```

Ξεκινά το πρόγραμμα εκτελώντας τη `main()`

Call by value

- Στη C, κάθε κλήση συνάρτησης περνάει τα ορίσματα αντιγράφοντάς τα στις παραμέτρους
- Αλλαγές στο αντίγραφο δεν επηρεάζουν το πρωτότυπο
- Χρησιμοποιείται όταν η συνάρτηση δεν χρειάζεται να αλλάξει τα ορίσματα

Call by value

```
void say_something(int x)
{
  x++;
  printf("The answer is %d\n", x);
}

int main(void)
{
  int i = 41;
  say_something(i);
}
```

Δημιουργείται μια θέση μνήμης `i`

Call by value

- Στη C, κάθε κλήση συνάρτησης περνάει τα ορίσματα αντιγράφοντάς τα στις παραμέτρους
- Αλλαγές στο αντίγραφο δεν επηρεάζουν το πρωτότυπο
- Χρησιμοποιείται όταν η συνάρτηση δεν χρειάζεται να αλλάξει τα ορίσματα

Call by value

```
void say_something(int x)
{
  x++;
  printf("The answer is %d\n", x);
}

int main(void)
{
  int i = 41;
  say_something(i);
}
```

Γράφεται η τιμή 41 στη θέση μνήμης `i`

Call by value

- Στη C, κάθε κλήση συνάρτησης περνάει τα ορίσματα αντιγράφοντάς τα στις παραμέτρους
- Αλλαγές στο αντίγραφο δεν επηρεάζουν το πρωτότυπο
- Χρησιμοποιείται όταν η συνάρτηση δεν χρειάζεται να αλλάξει τα ορίσματα

Call by value

```
void say_something(int x)
{
  x++;
  printf("The answer is %d\n", x);
}

int main(void)
{
  int i = 41;
  say_something(i);
}
```

Καλείται η συνάρτηση `say_something()`

Call by value

- Στη C, κάθε κλήση συνάρτησης περνάει τα ορίσματα αντιγράφοντάς τα στις παραμέτρους
- Αλλαγές στο αντίγραφο δεν επηρεάζουν το πρωτότυπο
- Χρησιμοποιείται όταν η συνάρτηση δεν χρειάζεται να αλλάξει τα ορίσματα

Call by value

```
void say_something(int x)
{
  x++;
  printf("The answer is %d\n", x);
}

int main(void)
{
  int i = 41;
  say_something(i);
}
```

Το όρισμα της κλήσης είναι η τιμή που περιέχει το `i`, δηλαδή 41

Call by value

- Στη C, κάθε κλήση συνάρτησης περνάει τα ορίσματα αντιγράφοντάς τα στις παραμέτρους
- Αλλαγές στο αντίγραφο δεν επηρεάζουν το πρωτότυπο
- Χρησιμοποιείται όταν η συνάρτηση δεν χρειάζεται να αλλάξει τα ορίσματα

Η συνάρτηση `say_something()` έχει μια παράμετρο `x`

Call by value

```
void say_something(int x)
{
  x++;
  printf("The answer is %d\n", x);
}

int main(void)
{
  int i = 41;
  say_something(i);
}
```

Call by value

- Στη C, κάθε κλήση συνάρτησης περνάει τα ορίσματα αντιγράφοντάς τα στις παραμέτρους
- Αλλαγές στο αντίγραφο δεν επηρεάζουν το πρωτότυπο
- Χρησιμοποιείται όταν η συνάρτηση δεν χρειάζεται να αλλάξει τα ορίσματα

Δημιουργείται μια νέα θέση μνήμης x

Call by value

```
void say_something(int x)
{
  x++;
  printf("The answer is %d\n", x);
}

int main(void)
{
  int i = 41;
  say_something(i);
}
```

Call by value

- Στη C, κάθε κλήση συνάρτησης περνάει τα ορίσματα αντιγράφοντάς τα στις παραμέτρους
- Αλλαγές στο αντίγραφο δεν επηρεάζουν το πρωτότυπο
- Χρησιμοποιείται όταν η συνάρτηση δεν χρειάζεται να αλλάξει τα ορίσματα

Αντιγράφεται η τιμή του ορίσματος της κλήσης (δηλαδή 41) στη θέση μνήμης x

Call by value

```
void say_something(int x)
{
  x++;
  printf("The answer is %d\n", x);
}

int main(void)
{
  int i = 41;
  say_something(i);
}
```

Call by value

- Στη C, κάθε κλήση συνάρτησης περνάει τα ορίσματα αντιγράφοντάς τα στις παραμέτρους
- Αλλαγές στο αντίγραφο δεν επηρεάζουν το πρωτότυπο
- Χρησιμοποιείται όταν η συνάρτηση δεν χρειάζεται να αλλάξει τα ορίσματα

Call by value

```
void say_something(int x)
{
  x++;
  printf("The answer is %d\n", x);
}

int main(void)
{
  int i = 41;
  say_something(i);
}
```

Αυξάνεται ο αριθμός που περιέχει η θέση μνήμης x κατά 1

Call by value

- Στη C, κάθε κλήση συνάρτησης περνάει τα ορίσματα αντιγράφοντάς τα στις παραμέτρους
- Αλλαγές στο αντίγραφο δεν επηρεάζουν το πρωτότυπο
- Χρησιμοποιείται όταν η συνάρτηση δεν χρειάζεται να αλλάξει τα ορίσματα

Call by value

```
void say_something(int x)
{
  x++;
  printf("The answer is %d\n", x);
}

int main(void)
{
  int i = 41;
  say_something(i);
}
```

Η θέση μνήμης `i` είναι διαφορετική, δεν επηρεάζονται τα περιεχόμενά της

Call by value

- Στη C, κάθε κλήση συνάρτησης περνάει τα ορίσματα αντιγράφοντάς τα στις παραμέτρους
- Αλλαγές στο αντίγραφο δεν επηρεάζουν το πρωτότυπο
- Χρησιμοποιείται όταν η συνάρτηση δεν χρειάζεται να αλλάξει τα ορίσματα

Call by value

```
void say_something(int x)
{
  x++;
  printf("The answer is %d\n", x);
}

int main(void)
{
  int i = 41;
  say_something(i);
}
```

Στην κλήση της `printf` το όρισμα είναι η τιμή του `x` δηλαδή 42

Call by value

- Στη C, κάθε κλήση συνάρτησης περνάει τα ορίσματα αντιγράφοντάς τα στις παραμέτρους
- Αλλαγές στο αντίγραφο δεν επηρεάζουν το πρωτότυπο
- Χρησιμοποιείται όταν η συνάρτηση δεν χρειάζεται να αλλάξει τα ορίσματα

Call by value

```
void say_something(int x)
{
  x++;
  printf("The answer is %d\n", x);
}

int main(void)
{
  int i = 41;
  say_something(i);
}
```

Στο τέλος της `say_something()` απελευθερώνεται η θέση μνήμης `x` και χάνονται τα περιεχόμενά της

Call by value

- Στη C, κάθε κλήση συνάρτησης περνάει τα ορίσματα αντιγράφοντάς τα στις παραμέτρους
- Αλλαγές στο αντίγραφο δεν επηρεάζουν το πρωτότυπο
- Χρησιμοποιείται όταν η συνάρτηση δεν χρειάζεται να αλλάξει τα ορίσματα

Call by value

```
void say_something(int x)
{
  x++;
  printf("The answer is %d\n", x);
}

int main(void)
{
  int i = 41;
  say_something(i);
}
```

Ο έλεγχος ροής επιστρέφει στη `main()` για την επόμενη εντολή

Call by value

- Στη C, κάθε κλήση συνάρτησης περνάει τα ορίσματα αντιγράφοντάς τα στις παραμέτρους
- Αλλαγές στο αντίγραφο δεν επηρεάζουν το πρωτότυπο
- Χρησιμοποιείται όταν η συνάρτηση δεν χρειάζεται να αλλάξει τα ορίσματα

Call by value

```
void say_something(int x)
{
  x++;
  printf("The answer is %d\n", x);
}

int main(void)
{
  int i = 41;
  say_something(i);
}
```

Η θέση μνήμης `i` δεν επηρεάστηκε από την `say_something`

Call by reference

- Μερικές φορές χρειάζεται μια συνάρτηση να αλλάζει τιμή στα ορίσματά της
  - Όπως στις παραμέτρους *εξόδου* μιας διεργασίας στο Λύκειο
- Στη C δεν υπάρχει παράμετρος εξόδου
- Χρησιμοποιούμε ένα δείκτη στη θέση μνήμης που είναι το όρισμα

Δείκτες

```
int *x;
int i;
x = &i;
```

- Η συνάρτηση μπορεί να γράψει ή να διαβάσει “εκεί που δείχνει” ο δείκτης
  - Έμμεσα, λειτουργεί όπως οι παράμετροι εξόδου
- Η χρήση δεικτών χρειάζεται *πολλή* προσοχή
  - Συχνά λάθη

Call by reference

- Μερικές φορές χρειάζεται μια συνάρτηση να αλλάζει τιμή στα ορίσματά της
  - Όπως στις παραμέτρους *εξόδου* μιας διεργασίας στο Λύκειο
- Στη C δεν υπάρχει παράμετρος εξόδου
- Χρησιμοποιούμε ένα δείκτη στη θέση μνήμης που είναι το όρισμα

Η μεταβλητή x περιέχει μια διεύθυνση στη μνήμη, όπου βρίσκεται αποθηκευμένος ένας ακέραιος

Δείκτες

```
int *x;
int i;
x = &i;
```

- Η συνάρτηση μπορεί να γράψει ή να διαβάσει “εκεί που δείχνει” ο δείκτης
  - Έμμεσα, λειτουργεί όπως οι παράμετροι εξόδου
- Η χρήση δεικτών χρειάζεται *πολλή* προσοχή
  - Συχνά λάθη

Call by reference

- Μερικές φορές χρειάζεται μια συνάρτηση να αλλάζει τιμή στα ορίσματά της
  - Όπως στις παραμέτρους *εξόδου* μιας διεργασίας στο Λύκειο
- Στη C δεν υπάρχει παράμετρος εξόδου
- Χρησιμοποιούμε ένα δείκτη στη θέση μνήμης που είναι το όρισμα

Δείκτες

```
int *x;
int i;
x = &i;
```

Αποθηκεύει στη μεταβλητή `x` τη διεύθυνση του `i`

- Η συνάρτηση μπορεί να γράψει ή να διαβάσει “εκεί που δείχνει” ο δείκτης
  - Έμμεσα, λειτουργεί όπως οι παράμετροι εξόδου
- Η χρήση δεικτών χρειάζεται *πολλή* προσοχή
  - Συχνά λάθη

Παράδειγμα

swap.c

```
void swap(int *x, int *y)
{
  int temp;

  temp = *x;
  *x = *y;
  *y = temp;
}

int main(void)
{
  int i = 1;
  int j = 31337;

  swap(&i, &j);
  return 0;
}
```

Παράδειγμα

Η συνάρτηση swap δέχεται δύο ορίσματα που δείχνουν σε θέσεις μνήμεις που περιέχουν ακέραιους

swap.c

```
void swap(int *x, int *y)
{
  int temp;

  temp = *x;
  *x = *y;
  *y = temp;
}

int main(void)
{
  int i = 1;
  int j = 31337;

  swap(&i, &j);
  return 0;
}
```

Παράδειγμα

swap.c

```
void swap(int *x, int *y)
{
  int temp;

  temp = *x;
  *x = *y;
  *y = temp;
}

int main(void)
{
  int i = 1;
  int j = 31337;

  swap(&i, &j);
  return 0;
}
```

Θα χρειαστεί μια προσωρινή θέση μνήμης για να αποθηκεύσουμε τη μια τιμή

Παράδειγμα

swap.c

```
void swap(int *x, int *y)
{
  int temp;

  temp = *x;
  *x = *y;
  *y = temp;
}

int main(void)
{
  int i = 1;
  int j = 31337;

  swap(&i, &j);
  return 0;
}
```

Αντιγράφουμε τα περιεχόμενα της θέσης μνήμης που δείχνει ο δείκτης `x` στη θέση μνήμης `temp`

Παράδειγμα

swap.c

```
void swap(int *x, int *y)
{
  int temp;

  temp = *x;
  *x = *y;
  *y = temp;
}

int main(void)
{
  int i = 1;
  int j = 31337;

  swap(&i, &j);
  return 0;
}
```

Αντιγράφουμε τα περιεχόμενα της θέσης μνήμης που δείχνει ο δείκτης y στη θέση μνήμης που δείχνει ο δείκτης x

Παράδειγμα

swap.c

```
void swap(int *x, int *y)
{
  int temp;

  temp = *x;
  *x = *y;
  *y = temp;
}

int main(void)
{
  int i = 1;
  int j = 31337;

  swap(&i, &j);
  return 0;
}
```

Αντιγράφουμε τα περιεχόμενα της θέσης μνήμης `temp` στη θέση μνήμης που δείχνει ο δείκτης `y`

Παράδειγμα

swap.c

```
void swap(int *x, int *y)
{
  int temp;

  temp = *x;
  *x = *y;
  *y = temp;
}

int main(void)
{
  int i = 1;
  int j = 31337;

  swap(&i, &j);
  return 0;
}
```