Προγραμματισμός Ι (ECE115)

#10 πέρασμα παραμέτρων συνάρτησης καθ' αναφορά (δείκτες και πίνακες)

Παράμετροι συναρτήσεων

- Οι πραγματικές παράμετροι των συναρτήσεων περνιούνται πάντα **καθ' αποτίμηση**
  - π.χ. αν σαν παράμετρος δοθεί μια μεταβλητή ή έκφραση, σαν πραγματική παράμετρος θα περαστεί **η τιμή** της
- Οι πραγματικές παράμετροι (τιμές) αποθηκεύονται σε τοπική μνήμη / την στοίβα
  - αντιστοιχούν σε προσωρινές τοπικές μεταβλητές με τα ονόματα των αντίστοιχων τυπικών παραμέτρων
- Αν ο κώδικας της συνάρτησης αλλάξει την τιμή μιας τυπικής παραμέτρου, στην πραγματικότητα αλλάζει την τιμή της αντίστοιχης **τοπικής μεταβλητής**
  - όχι της μεταβλητής που περάστηκε ως παράμετρος

```c
#include <stdio.h>

void inc0(int a) {
  a=a+1;
}

int main(int argc, char *argv[]) {
  int b = 5;

  inc0(b);
  printf("b=%d\n", b);

  inc0(b);
  printf("b=%d\n", b);

  return(0);
}
```

η παράμετρος `a` και οι όποιες αλλαγές τυχόν γίνουν σε αυτή, είναι ορατά **μόνο** μέσα στο πλαίσιο της συνάρτησης

σαν **πραγματική** παράμετρος (για την τυπική παράμετρο `a` της συνάρτησης) περνιέται η **τιμή** της μεταβλητής `b`

Δείκτες ως παράμετροι συναρτήσεων

- Μπορεί σαν παράμετρος να δοθεί μια **διεύθυνση**
  - τότε η συνάρτηση μπορεί να αλλάξει τα περιεχόμενα σε αυτή την θέση μνήμης, π.χ., μια εξωτερική μεταβλητή
- Αυτό ονομάζεται πέρασμα **καθ' αναφορά**
  - call/pass by reference (vs. call/pass by value)
- Η τυπική παράμετρος δηλώνεται ως `pointer-to-T`
  - ο κώδικας της συνάρτησης χρησιμοποιεί / αλλάζει την τιμή αντίστοιχα, σύμφωνα με όσα έχουμε πει για δείκτες
- Στην κλήση, σαν πραγματική παράμετρος δίνεται μια **διεύθυνση** ενός αντικειμένου (μεταβλητής) τύπου `T`
  - αυτό ακριβώς κάνουμε όταν καλούμε την `scanf` ώστε η τιμή της μεταβλητής να **αλλάξει** μέσα από την `scanf`

```c
#include <stdio.h>

void inc1(int *ptr) {
  *ptr = *ptr + 1;
}

int main(int argc, char *argv[]) {
  int b = 5;

  inc1(&b);
  printf("b=%d\n", b);

  inc1(&b);
  printf("b=%d\n", b);

  return(0);
}
```

δείκτης!

σαν **πραγματική** παράμετρος (για την τυπική παράμετρο `ptr` της συνάρτησης) περνιέται η **διεύθυνση** της μεταβλητής `b`

```c
#include <stdio.h>

void swap(int *ptr1, int *ptr2) {
  int tmp;

  tmp = *ptr1;
  *ptr1 = *ptr2;
  *ptr2 = tmp;
}

int main(int argc, char *argv[]) {
  int val1,val2;

  printf("enter 2 int values: ");
  scanf("%d %d", &val1, &val2);
  printf("val1=%d, val2=%d\n", val1, val2);

  swap(&val1, &val2);

  printf("val1=%d, val2=%d\n", val1, val2);

  return(0);
}
```

Πίνακες ως παράμετροι συναρτήσεων

- Η τυπική παράμετρος δηλώνεται ως **πίνακας**
  - δεν είναι υποχρεωτικό να προσδιοριστεί το μέγεθος της «πρώτης» διάστασης του πίνακα, όμως πρέπει να προσδιορίζεται το μέγεθος των υπολοίπων διαστάσεων

- Σαν πραγματική παράμετρος περνιέται αυτομάτως η **διεύθυνση** (του πρώτου στοιχείου) του πίνακα

- Αποφεύγεται η αντιγραφή όλων των στοιχείων του πίνακα στην στοίβα
  - μείωση χρόνου εκτέλεσης
  - μείωση μεγέθους στοίβας

```
#include<stdio.h>
#define SIZE 5

double calcAverage(int vals[SIZE]) {
  int i;
  double sum = 0;

  for (i = 0; i < SIZE; ++i) {
    sum = sum + vals[i];
  }
  return(sum / SIZE);
}

int main () {
  int v[SIZE] = {1000, 2, 3, 17, 50};
  double avg;

  avg = calcAverage(v);
  printf("Average value is: %lf\n", avg);

  return(0);
}
```

```
#include<stdio.h>
#define SIZE 5

double calcAverage(int vals[], int size) {
  int i;
  double sum = 0;

  for (i = 0; i < size; ++i) {
    sum = sum + vals[i];
  }
  return(sum / size);
}

int main () {
  int v[SIZE] = {1000, 2, 3, 17, 50};
  double avg;

  avg = calcAverage(v, SIZE) ;
  printf("Average value is: %lf\n", avg);

  return(0);
}
```

```c
#include<stdio.h>

#define N 3
#define M 2

void print(int vals[N][M]) {
  int i, j;

  for (i = 0; i < N; i++) {
    for (j = 0; j < M; j++) {
      printf("%d ", vals[i][j]);
    }
    printf("\n");
  }
}

int main(int argc, char *argv[]) {
  int v[][M] = {{1, 2}, {3, 4}, {5, 6}};

  print(v);

  return(0);
}
```

```c
#include<stdio.h>

#define N 3
#define M 2

void print(int vals[][M], int n) {
  int i, j;

  for (i = 0; i < n; i++) {
    for (j = 0; j < M; j++) {
      printf("%d ", vals[i][j]);
    }
  }
  printf("\n");
}

int main(int argc, char *argv[]) {
  int v[][M] = {{1, 2}, {3, 4}, {5, 6}};

  print(v, N);

  return(0);
}
```

Πίνακες ως παράμετροι συναρτήσεων (2)

- Σαν πραγματική παράμετρος περνιέται αυτομάτως η **διεύθυνση** (του πρώτου στοιχείου) του πίνακα
- Το πέρασμα είναι **καθ' αναφορά**!
- Ο κώδικας της συνάρτησης μπορεί (όχι μόνο να προσπελάσει, αλλά και) να **αλλάξει** τις τιμές των στοιχείων του πίνακα

```c
#include<stdio.h>
#define SIZE 5

void sum(int v1[SIZE], int v2[SIZE], int sums[SIZE]) {
  int i;

  for (i=0; i<SIZE; ++i) {
    sums[i] = v1[i] + v2[i];
  }
}

int main () {
  int vals1[SIZE] = {1000, 2, 3, 17, 50};
  int vals2[SIZE] = {50, 17, 3, 2, 1000};
  int vals3[SIZE], i;

  sum(vals1, vals2, vals3);

  printf("The sums are: ");
  for (i=0; i<SIZE; i++) { printf("%d ", vals3[i]); }
  printf("\n");

  return(0);
}
```

```c
#include <stdio.h>

#define N 16

void smallToCapitals(char str[]) {
  int i;

  for (i=0; str[i] != '\0'; i++) {
    if ( (str[i] >= 'a') && (str[i] <= 'z') ) {
      str[i] = 'A' + str[i] - 'a';
    }
  }
}

int main(int argc, char *argv[]) {
  char str[N];

  scanf("%15s",str);
  smallToCapitals(str);
  printf("%s\n",str);

  return(0);
}
```

```c
#include <stdio.h>
#define NUM_NAMES 3
#define NAME_SIZE 20

void read_input(char buffer[NUM_NAMES][NAME_SIZE]) {
  int i;

  printf("Enter 3 strings up to 19 characters\n");
  for (i = 0; i < NUM_NAMES; i++) {
    scanf("%19s", buffer[i]);
  }
}

int main (int argc, char *argv[]) {
  char input[NUM_NAMES][NAME_SIZE];
  int i;

  read_input(input);
  for (i = 0; i < NUM_NAMES; i++) {
    printf("input[%d]: %s\n", i, input[i]);
  }

  return(0);
}
```

Σχόλιο

- Η αλλαγή της τιμής μιας εξωτερικής μεταβλητής μέσα από συνάρτηση, μέσω δείκτη, είναι μια μορφή παρενέργειας
- Όμως αυτό είναι **ορατό** στον κώδικα
  - υπάρχει αντίστοιχη τυπική παράμετρος της συνάρτησης που έχει **δηλωθεί** ως δείκτης (ή πίνακας)
  - όταν γίνεται η κλήση της συνάρτησης, δίνεται ρητά ως πραγματική παράμετρος η **διεύθυνση** της μεταβλητής (ή το όνομα του πίνακα)
- Συνεπώς δεν αποτελεί «κακή» παρενέργεια

- Πρόθεμα `const` σε μια τυπική παράμετρο δείκτη
  - δηλώνει ότι η συνάρτηση **δεν** αλλάζει τα περιεχόμενα μιας παραμέτρου που περνιέται καθ' αναφορά

```c
#include<stdio.h>
#define SIZE 5

double calcAverage(const int vals[SIZE]) {
  int i;
  double sum = 0;

  for (i = 0; i < SIZE; ++i) {
    sum = sum + vals[i];
  }
  return(sum / SIZE);
}

int main () {
  int v[SIZE] = {1000, 2, 3, 17, 50};
  double avg;

  avg = calcAverage(v);
  printf("Average value is: %lf\n", avg);

  return(0);
}
```

Δείκτες ως τιμή επιστροφής συνάρτησης

- Μια συνάρτηση μπορεί να δηλωθεί έτσι ώστε να **επιστρέφει** σαν αποτέλεσμα ένα `pointer-to-T`
- Η τιμή που επιστρέφεται πρέπει να αντιστοιχεί σε διεύθυνση που θα **έχει ισχύ** ακόμα και **μετά** τον τερματισμό της συνάρτησης

- **Κλασικό λάθος**: επιστροφή της διεύθυνσης μιας (προσωρινής) τοπικής μεταβλητής
  - αυτή **δεν** έχει ισχύ μετά την επιστροφή της συνάρτησης
- Αυτό **δεν** εντοπίζεται (πάντα) από τον μεταφραστή
  - αν είμαστε τυχεροί, οδηγεί (**όχι** πάντα/εγγυημένα) σε τερματισμό της εκτέλεσης του προγράμματος

```c
#include <stdio.h>

int *add(int a, int b) {
  int c;
  c = a+b;
  return(&c);
}

int foo(int a, int b) {
  int c;
  c = a-b;
  return(c);
}

int main(int argc, char *argv[]) {
  int a, b, *ptr;

  printf("Enter 2 int values: ");
  scanf("%d %d", &a, &b);

  ptr = add(a, b);
  printf("%d\n", *ptr);

  foo(a, b);
  printf("%d\n", *ptr);

  return(0);
}
```

λάθος!

ίδιος αριθμός/τύπος παραμέτρων και τοπικών μεταβλητών με add