ΕΡΓΑΣΙΑ 3
ΒΡΟΧΟΙ

Χρήστος Μαργιώλης – Εργαστηριακό Τμήμα 9

Θεωρητικές ερωτήσεις

1.1. Τί είναι οι επαναληπτικές δομές (βρόχοι);

Οι επαναληπτικές δομές (loops) χρησιμοποιούνται για την εκτέλεση ενός block εντολών πάνω από μια φορές, ώσπου η συνθήκη η οποία επιτρέπει στις εντολές να εκτελεστούν να γίνει ψευδής.

1.2. Ποιές επαναληπτικές δομές γνωρίζετε στην γλώσσα C;

`for`, `while`, και `do-while`.

1.3. Πώς σχετίζονται μεταξύ τους;

Και οι 3 δομές εκτελούν ακριβώς την ίδια εργασία. Η διαφορά είναι ανάμεσα στην `do-while` και στις άλλες 2 δομές (`for` και `while`). Η `do-while` εκτελεί *πρώτα μια φορά* το block εντολών που περιέχει και στο τέλος ελέγχει αν η συνθήκη επανάληψης είναι αληθής ή όχι, και έτσι, ξαναεκτελείται ή τερματίζεται. Η `for` και η `while` πρώτα ελέγχουν αν η συνθήκη επανάληψης είναι αληθής ή ψευδής και έπειτα εκτελούνται (ή δεν εκτελούνται) τα block εντολών που περιέχουν.

2.1. Περιγράψτε την δομή επανάληψης «for».

Η σύνταξη της `for` είναι η εξής: `for(αρχικοποίηση τιμής επανάληψης (initialization); συνθήκη (condition); αύξηση/μείωση τιμής επανάληψης (increment/decrement)`.

Στην *αρχικοποιήση* δίνουμε μια τιμή στην μεταβλητή που συνήθως λειτουργεί ως μετρητής στον βρόχο. Για παράδειγμα `i = 0`.

Στην *συνθήκη* γράφουμε μια συνθήκη η οποία όσο είναι αληθής επιτρέπει στον βρόχο να εκτελεστεί. Για παράδειγμα `i < 10`.

Στην *αύξηση/μείωση* τροποποιούμε τον μετρητή. Για παράδειγμα `i++`.

Με τα παραπάνω παραδείγματα έχουμε την δομή

```
for(i = 0; i < 10; i++)
{
   εντολές;
}
```

Οι εντολές που βρίσκονται μέσα στην επανάληψη θα εκτελεστούν όσο το `i` είναι μικρότερο του 10, και εφοσον το `i` αυξάνεται σε κάθε επανάληψη κατα 1 (`i++`) και η αρχική του τιμή είναι 0, η επανάληψη θα γίνει 10 φορές. Μετά τις 10 φορές, η συνθήκη θα είναι ψευδής και έτσι το πρόγραμμα θα προχωρήσει την λειτουργία του.

2.2. Τί γνωρίζετε για τις εντολές «break» και «continue»;

`break`: Χρησιμοποείται μόνο σε loops ή σε `switch`. Μόλις αυτή η εντολή εκτελεστεί από το πρόγραμμα, το loop σταματάει αμέσως. Αν η `break` εκτελεστεί μέσα σε `switch`, σταματάει όλο το υπόλοιπο σώμα της `switch`.

`continue`: Χρησιμοποείται μόνο σε loops και όταν εκτελεστεί η εντολή αυτή, το πρόγραμμα παραλείπει τις εντολές που βρίσκονται στο loop κάτω από την `continue`, και προχωράει στην επόμενη επανάληψη.

Προγράμματα

3.1. **int-calcs.c** - Άσκηση 3

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
    int num, averPos, sqrOdd;
    int sumPos = 0, numPos = 0, numEven = 0, prodNeg = 1;

    do
    {
        printf("Number: ");
        scanf("%d", &num);

        if (num > 0)
        {
            sumPos += num;
            numPos += 1;


            if (num % 2 != 0)
            {
                sqrOdd = num * num;
                printf("Square of odd number %d: %d\n", num,
sqrOdd);
            }
            else
                numEven += 1;
        }
        else if (num < 0)
            prodNeg *= num;
    }
    while (num != 0);

    averPos = sumPos / numPos;
```

```
    printf("Average value of positive numbers: %d\n",
averPos);
    printf("Product of negative numbers: %d\n", prodNeg);
    printf("Number of even numbers: %d\n", numEven);

    return 0;
}
```

3.2. Περιγραφή υλοποίησης άσκησης 3

- **Μεταβλητές**

  `num`: Ακέραιος *που δίνεται από τον χρήστη κάθε φορά*
  `averPos`: Μέσος όρος των ακεραίων
  `sqrOdd`: Ρίζα περιττού αριθμού
  `sumPos`: Άθροισμα θετικών αριθμών
  `numPos`: Πλήθος θετικών αριθμών
  `numEven`: Πλήθος αρτίων αριθμών
  `prodNeg`: Γινόμενο αρνητικών αριθμών

- **Λειτουργία του προγράμματος**

  Αρχικά δίνεται από τον χρήστη ένας ακέραιος αριθμός και στην συνέχεια ελέγχει αν είναι θετικός ή αρνητικός. Αν είναι θετικός, αυξάνεται το πλήθος και το άθροισμα των θετικών, και κάνει έναν επιπλέον έλεγχο για να βρεθεί αν ο αριθμός είναι περιττός ή άρτιος. Στην περίπτωση που είναι περιττός, υπολογίζεται και εμφανίζεται το τετράγωνο αυτού του αριθμού, και αν είναι άρτιος, αυξάνεται το πλήθος των αρτίων αριθμών. Αν ο αριθμός είναι αρνητικός, ο αριθμός πολλαπλασιάζεται στην μεταβλητή που υπολογίζει το γινόμενο των αρνητικών αριθμών. Τέλος, η επανάληψη αυτή συνεχίζεται μέχρι ο χρήστης να δώσει το 0 ως αριθμό, και αφού τελειώση η επανάληψη, βρίσκεται ο μέσος όρος των θετικών αριθμών.

4.1. **shapes.c** - Άσκηση 4

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
    int row, rowsTotal, col, colsTotal, spaces;

    printf("How many rows?: ");
    scanf("%d", &rowsTotal);
    colsTotal = rowsTotal;

    printf("\n");

    for (row = 0; row < rowsTotal; row++)
    {
        printf("*\n");
        for (col = row+1; col > 0; col--)
        {
            if (row+1 < colsTotal)
                printf("*");
        }
    }

    printf("\n");

    for (row = 0; row < rowsTotal; row++)
    {
        for (col = row+1; col < colsTotal; col++)
            printf(" ");

        for (col = 0; col <= row; col++)
            printf("*");

        printf("\n");
    }

    printf("\n");

    spaces = rowsTotal;
    for (row = 1; row <= rowsTotal; row++)
    {
        for (col = 1; col < spaces; col++)
            printf(" ");

        for (col = 0; col < 2*row - 1; col++)
            printf("*");

        printf("\n");
```

```
            spaces--;
        }

        printf("\n");

        for (row = 1; row <= rowsTotal; row++)
        {
            for (col = 1; col <= colsTotal; col++)
            {
                if (row == 1 || row == rowsTotal || col == 1 ||
col == colsTotal)
                    printf("*");
                else if (row == col || col == (rowsTotal - row +
1))
                    printf(".");
                else
                    printf(" ");
            }

            printf("\n");
        }

        printf("\n");

        return 0;
}
```

4.2. Περιγραφή υλοποίησης άσκησης 4

- **Μεταβλητές**

  `row`: Γραμμή
  `rowsTotal`: Συνολικό πλήθος γραμμών *που έχουν δωθεί από τον χρήστη*
  `col`: Στήλη
  `colsTotal`: Συνολικό πλήθος στηλών
  `spaces`: Πλήθος κενών

- **Λειτουργία του προγράμματος**

  Για `rowsTotal` = 5

  ```
  *
  **
  ***
  ****
  *****
  ```

  * = Ο αστερίσκος που τυπώνεται κάθε φορά στην *πρώτη* `for`.
  * = Οι αστερίσκοι που τυπώνονται στην *δεύτερη* `for`.

```
____*
___**
__***
_****
*****
```

_ = Τα κενά που τυπώνονται στην *πρώτη* `for`.
* = Οι αστερίσκοι που τυπώνονται στην *δεύτερη* `for`.

```
____*____
___***___
__*****__
_*******_
*********
```

_ = Τα κενά που τυπώνονται στην *πρώτη* `for`.
* = Οι αστερίσκοι που τυπώνονται στην *δεύτερη* `for`.

Το σχήμα αυτό δημιουργείται χρησιμοποιώντας μια επιπλέον μεταβλητή (`spaces`) για να μετράει το πλήθος των κενών, ωστέ να εκτυπώνει όλο και λιγότερα κενά όσο αυξάνονται οι αστερίσκοι σε κάθε νέα σειρά.

```
*****
*._.*
*_._*
*._.*
*****
```

* = Οι αστερίσκοι που τυπώνονται στην *πρώτη* `for`.
. = Οι άνω τελείες που τυπώνονται στην *δεύτερη* `for`.
_ = Τα κενά που τυπώνονται στην *τρίτη* `for`.

Προκειμένου να τοποθετηθούν οι άνω τελείες, πρέπει να βρίσκεται η *κύρια* και η *δευτερεύουσα διαγώνιος* του τετραγώνου. Αυτό γίνεται στην `else if (row == col || col == (rowsTotal – row-1))`.

5.1. **sine-taylor.c** - Άσκηση 5

```c
#include <stdio.h>
#include <math.h>

#define PI 3.141592654
#define ACCURACY 0.000001

double power(double, int);
int factorial(int);


int main(int argc, char *argv[])
{
    double xDegrees, xRads, currentFrac, previousFrac, sine =
0;
    int exponent = 1, sign = 1;

    printf("x (in degrees): ");
    scanf("%lf", &xDegrees);
    xRads = xDegrees * (PI/180.0);

    currentFrac = power(xRads, exponent) /
(double)factorial(exponent);
```

```
    do
    {
        sine += sign * currentFrac;

        exponent += 2;

        previousFrac = currentFrac;
        currentFrac = power(xRads, exponent) /
(double)factorial(exponent);

        sign *= -1;
    }
    while (fabs(previousFrac - currentFrac) > ACCURACY);

    printf("sin(%lf) = sin(%lf) = %lf rads\n", xDegrees,
xRads, sine);

    return 0;
}


double power(double xRads, int exponent)
{
    int i;
    double value;

    for (i = 0, value = 1; i < exponent; i++)
        value *= xRads;

    return value;
}


int factorial(int exponent)
{
    int i, fac;

    for (i = 1, fac = 1; i <= exponent; i++)
        fac *= i;

    return fac;
}
```

5.2. Περιγραφή υλοποίησης άσκησης 5

- **Μεταβλητές**

  `xDegrees`: x σε μοίρες
  `xRads`: x σε rads/ακτίνια
  `currentFrac`: Τρέχον κλάσμα
  `previousFrac`: Προηγούμενο κλάσμα
  `exponent`: Εκθέτης
  `sign`: Πρόσιμο παράστασης
  `sine`: Ημίτονο
  `value`: Δύναμη
  `fac`: Παραγοντικό

- **Λειτουργία του προγράμματος**

  Αρχικά ο χρήστης δίνει το `x` σε μοίρες και το πρόγραμμα το μετατρέπει αμέσως σε rads. Εφόσον η αρχική τιμή του εκθέτη είναι 1, καλούνται οι συναρτήσεις `power()` και `factorial()` ωστέ να υπολογίσουν την δύναμη και το παραγοντικό και να βρεθεί το τρέχον κλάσμα `currentFrac`. Στην συνέχεια το τρέχον κλάσμα προστείθεται στην σειρά του ημιτόνου και ο εκθέτης αυξάνεται κατά 2. Το προηγούμενο κλάσμα (`previousFrac`) τώρα είναι το τρέχον κλάσμα, και το νέο τρέχον κλάσμα είναι το κλάσμα που θα προκύψει από την νέα κλήση των παραπάνω συναρτήσεων εφόσον ο εκθέτης άλλαξε, άρα θα αλλάξει και η τιμή του κλάσματος. Η μεταβλητή `sign` αλλάζει πρόσιμο σε κάθε επανάληψη ώστε να επιτευχθεί η εναλλαγή προσίμων της σειράς $\sin(x)=x-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\frac{x^9}{9!}\ldots$. Αυτή η επανάληψη συνεχίζεται όσο η διαφορά του προηγούμενου και του τρέχοντος κλάσματος σε απόλυτη τιμή να είναι μεγαλύτερη του $10^{-6}=0.000001$.

Εργαλεία

- **Editor**: Visual Studio Code
- **Compiler**: gcc
- **Συγγραφή**: Libre Office Writer
- **Γραμματοσειρές**: Liberation Sans *για τα κείμενα* και Liberation Mono *για τους κώδικες*
- **Λειτουργικό σύστημα**: Linux Mint Cinnamon 19.2