Εισαγωγή στον Προγραμματισμό

Μέρος 2ο: Συναρτήσεις

Εξάμηνο Σπουδών: 3ο

Κωδικός Μαθήματος: 343

Τμήμα Μαθηματικών
Πανεπιστήμιο Ιωαννίνων

Μιχάλης Α. Μπέκος
bekos@uoi.gr

Εισαγωγή

- Είδη συναρτήσεων:
  - Απλές συναρτήσεις vs συναρτήσεις-μέλη
  - Προκαθορισμένες συναρτήσεις vs συναρτήσεις ορισμένες από τον προγραμματιστή

- Τύποι συναρτήσεων:
  - Αυτές που επιστρέφουν κάποια τιμή
  - Αυτές που δεν επιστρέφουν κάποια τιμή (void)

Μέρος 1°:
Προκαθορισμένες συναρτήσεις

Προκαθορισμένες Συναρτήσεις

- Αφορούν συναρτήσεις που περιεχονται σε βιβλιοθήκες της γλώσσας

- Για την κλήση τους, πρέπει να ανοίξουμε την κατάλληλη βιβλιοθήκη με `#include`
  - `<cmath>`, `<cstdlib>` (απλές “C” βιβλιοθήκες)
  - `<iostream>` (για `cout`, `cin`)

- Η κλήση μιας απλής συνάρτησης δεν απαιτεί την ύπαρξη αντικειμένου

```cpp
#include <cmath>
#include <iostream>
using namespace std;

int main() {
  double num;

  // Obtain input
  cout << "Specify a number: ";
  cin  >> num;

  double result = sqrt(num);
  cout << "Square root of " <<
   num << " is " << result;

  return 0;
}
```

Part02/sqrt.cpp

```cpp
int bonus = sqrt(sales)/10;
```

```cpp
double area = 3.14 * pow(r,2);
```

Μαθηματικές Συναρτήσεις

- Βρίσκονται στις βιβλιοθήκες `<cstdlib>` και `<cmath>`
- Οι περισσότερες επιστρέφουν μια τιμή

- `#include <cstdlib>`: Περιέχει συναρτήσεις όπως
  - `abs()` απόλυτη τιμή ενός int
  - `labs()` απόλυτη τιμή ενός long
  - `rand()` τυχαίος ακέραιος

- `#include <cmath>`: Περιέχει συναρτήσεις όπως
  - `pow( , )` δύναμη a^b (δύο ορίσματα)
  - `floor()` στρογγυλοποίηση προς τα κάτω
  - `sqrt()` τετραγωνική ρίζα

```cpp
#include <cmath>
#include <iostream>
using namespace std;

int main() {
  double x, y;

  // Obtain input
  cout << "Enter base: ";
  cin >> x;

  cout << "Enter exponent";
  cin >> y;

  //store the answer in result.
  double result = pow(x, y);

  //print the result
  cout << x << "^" << y << "="
       << result << endl;

  return 0;
}
```

Part02/pow.cpp

Πίνακας Μαθηματικών Συναρτήσεων*

Όνομα	Περιγραφή	Τύπος ορισμάτων	Τύπος επιστροφής	Παράδειγμα	Τιμή	Βιβλιοθήκη
`sqrt`	Τετραγωνική ρίζα	`double`	`double`	`sqrt(4.0)`	`2.0`	`<cmath>`
`pow`	Δύναμη	`double`	`double`	`pow(2.0,3.0)`	`8.0`	`<cmath>`
`abs`	Απόλυτη τιμή	`int`	`int`	`abs(-7)`	`7`	`<cstdlib>`
`labs`	Απόλυτη τιμή	`long`	`long`	`labs(-70000)`	`70000`	`<cstdlib>`
`fabs`	Απόλυτη τιμή	`double`	`double`	`fabs(-7.5)`	`7.5`	`<cmath>`
`ceil`	ανώφλι	`double`	`double`	`ceil(3.2)`	`4.0`	`<cmath>`
`floor`	κατώφλι	`double`	`double`	`floor(3.2)`	`3.0`	`<cmath>`
`exit`	τερματισμός προγράμματος	`int`	`void`	`exit(1)`	—	`<cstdlib>`
`rand`	τυχαίος αριθμός	—	`int`	`rand()`	ποικίλλει	`<cstdlib>`
`log`	λογάριθμος με βάση το 2	`double`	`double`	`log(4.0)`	`2.0`	`<cstdlib>`
`log10`	λογάριθμος με βάση το 10	`double`	`double`	`log10(100.0)`	`2.0`	`<cstdlib>`

* Υπάρχουν πολλές άλλες συναρτήσεις (τριγωνομετρικές, αλφαριθμητικές, ...)

Γεννήτριες τυχαίων αριθμών

- Επιστρέφουν “τυχαία επιλεγμένους” αριθμούς
- `rand()`:
  - δεν δέχεται ορίσματα
  - επιστρέφει ακέραιο μεταξύ `0` & `RAND_MAX`
  - `RAND_MAX`: σταθερά της βιβλιοθήκης `<cstdlib>` εξαρτάται από το σύστημα ≈ 32767
- Κλιμάκωση:
  - Επιτρέπει επιλογή από μικρότερο διάστημα

```
rand() % 6; // random int in [0,5]
```

- Μετατόπιση:
  - Μετατοπίζει το φάσμα επιλογής

```
rand() % 6 + 1; // random int in [1,6], e.g., dice
```

```
#include <cstdlib>
#include <ctime>
#include <iostream>
using namespace std;

int main () {
  int iSecret, iGuess;

  /* random number in [1,10] */
  srand(time(0));
  iSecret = rand() % 10 + 1;

  do {
    cout<<"Your guess in [1,10]:";
    cin>>iGuess;
    if (iSecret<iGuess)
      cout<<"The number is lower";
    else if (iSecret>iGuess)
      cout<<"The number is higher";
    cout<<endl;
  } while (iSecret!=iGuess);

  cout<<"Congratulations"<<endl;
}
```

Part02/rand.cpp

Γεννήτριες τυχαίων αριθμών

- Πολλές κλήσεις της `rand()` ⇒ παράγουν την ίδια ακολουθία αριθμών
- Χρήση της γεννήτριας συνάρτησης `seed()` για διαφορετικές τιμές
- `srand(seed_value)`:
  - `void` συνάρτηση
  - Δέχεται ένα όρισμα , ακέραιο αριθμό
  - Οποιαδήποτε τιμή, ακόμα και τον χρόνο συστήματος `srand(time(0))`
- `time()`:
  - επιστρέφει τον χρόνο συστήματος ως ακέραιο
  - περιέχεται στη βιλιοθήκη `<ctime>`

```cpp
#include <cstdlib>
#include <ctime>
#include <iostream>
using namespace std;

int main () {
  int iSecret, iGuess;

  /* random number in [1,10] */
  srand(time(0));
  iSecret = rand() % 10 + 1;

  do {
    cout<<"Your guess in [1,10]:";
    cin>>iGuess;
    if (iSecret<iGuess)
      cout<<"The number is lower";
    else if (iSecret>iGuess)
      cout<<"The number is higher";
    cout<<endl;
  } while (iSecret!=iGuess);

  cout<<"Congratulations"<<endl;
}
```

Part02/rand.cpp

Παράδειγματα

```cpp
#include <cstdlib>
#include <iostream>
#include <ctime>
using namespace std;

int main()
{
  const int MIN = 10, MAX = 100;

  srand(time(0));
  for (int i = 0; i < 5; i++) {
    /* random number in [0.0,1.0] */
    double num = rand() / ((double)(RAND_MAX));
    /* shift to [0.0, MAX-MIN] */
    num = num * (MAX-MIN);
    /* further shift to [MIN, MAX] */
    num = num + MIN;

    cout << num << endl;
  }

  return 0;
}
```

Part02/rand_double.cpp

Τυχαίοι `double`

```cpp
#include <cstdlib>
#include <iostream>
#include <ctime>
using namespace std;

int main()
{
  srand(100);
  cout << "First sequence: ";
  for (int i=0; i<10; i++) {
    cout << (rand() % 10 + 1) << " ";
  }
  cout << endl;

  srand(100);
  cout << "Second sequence: ";
  for (int i=0; i<10; i++) {
    cout << (rand() % 10 + 1) << " ";
  }
  cout << endl;

  return 0;
}
```

Part02/rand_sequences.cpp

10 όμοιες ακολουθίες τυχαίων ακεραίων

Μέρος 2°:

Συναρτήσεις οριζόμενες από τον προγραμματιστή

Συναρτήσεις οριζόμενες από τον προγραμματιστή

- Χωρίζουν το πρόγραμμα σε κομμάτια
  - Δομημένος προγραμματισμός
  - Πιο εύκολος στην κατανόηση
  - Επαναχρησιμοποίηση συναρτήσεων

- Όπως και με τις κλάσεις, ο ορισμός των συναρτήσεων μπορεί να γίνει:
  - είτε στο ίδιο αρχείο
  - είτε σε διαφορετικό αρχείο για να τις χρησιμοποιήσουν και άλλοι

Συστατικά για τον ορισμό συναρτήσεων

- 3 βασικά συστατικά
  - Δήλωση Συνάρτησης
    Πληροφορία για τον μεταφραστή
    Τοποθετείται πριν το πρώτο κάλεσμα της συνάρτησης
  - Ορισμός Συνάρτησης
    Ο κώδικας της συνάρτησης για το τί ακριβώς κάνει
    Όπως ακριβώς κάναμε μέσα στη `main()`
  - Κάλεσμα Συνάρτησης
    Μεταφέρει τον έλεγχο στην συνάρτηση
- Προσοχή: Η δήλωση ή ο ορισμός μιας συνάρτησης δεν μπορεί να γίνει “μέσα” σε κάποια άλλη συνάρτηση.
  Σύνηθες σφάλμα: “μέσα” στην `main()`

```cpp
#include <iostream>
using namespace std;

//function declaration
void starline();

//contains 3 calls to starline()
int main() {
  starline();
  cout << "Data type Range" << endl;
  starline();
  cout << "char      [-128,127]" << endl;
  cout << "int       System Dep" << endl;
  starline();
}

//function definition
void starline() {
  for (int i=0; i<45; i++) {
    cout << '*';
  }
  cout << endl;
}
```

Part02/starline.cpp

Συστατικά για τον ορισμό συναρτήσεων

- 3 βασικά συστατικά
  - Δήλωση Συνάρτησης
    Πληροφορία για τον μεταφραστή
    Τοποθετείται πριν το πρώτο κάλεσμα της συνάρτησης
  - Ορισμός Συνάρτησης
    Ο κώδικας της συνάρτησης για το τί ακριβώς κάνει
    Όπως ακριβώς κάναμε μέσα στη `main()`
  - Κάλεσμα Συνάρτησης
    Μεταφέρει τον έλεγχο στην συνάρτηση
- Αντίστοιχα γίνεται ο ορισμός των συναρτήσεων μελών (μεθόδων) των κλάσεων

```cpp
#include<iostream>
using namespace std;

class Fraction
{
private:
  int numerator, denominator;

public:
  ...
  //function declaration
  double getValue() const;
  ...
};

//function definition
double Fraction::getValue() const {
  return numerator / denominator;
}

int main() {
  Fraction f;
  cout << f.getValue() << endl;
}
```

Part02/starline.cpp

Δήλωση συνάρτησης

- Δήλωση για τον μεταφραστή
  - Περιλαμβάνει τον τύπο επιστροφής καθώς και τον τύπο όλων των παραμέτρων

    μην ξεχνάτε το ; στο τέλος
  - Τοποθετείται πριν το πρώτο κάλεσμα της συνάρτησης

    Συνήθως (σχεδόν πάντα) πριν τον ορισμό της `main()`
  - Τα ονόματα των παραμέτρων μπορούν να παραλείπονται

    π.χ. Η δήλωση `void repchar(char, int);` είναι ισοδύναμη της `rechar(char c, int n);`

```
#include <iostream>
using namespace std;

//function declaration
void repchar(char, int);

//contains 3 calls to repchar()
int main() {
  repchar('*', 43);
  cout << "Data type Range" << endl;
  repchar('-', 43);
  cout << "char      [-128,127]" << endl;
  cout << "int       System Dep" << endl;
  repchar('*', 43);
}

//function definition
void repchar(char c, int n) {
  for (int i=0; i<n; i++) {
    cout << c;
  }
  cout << endl;
}
```

Part02/repchar.cpp

Ορισμός συνάρτησης

- Κώδικας της συνάρτησης για το τί θα εκτελεί
  - όπως ακριβώς κάναμε μέσα στη `main()`
    Προσοχή: Μην τοποθετείται ; στο τέλος
  - Αποτελείται από μια ακολουθία εντολών, οι οποίες εκτελούνται σειριακά
  - Τα ονόματα των παραμέτρων στον ορισμό μιας συνάρτησης δεν μπορούν να παραλείπονται
    Χρησιμοποιήστε ονόματα παραμέτρων με νόημα
    Κάνουν τον κώδικα εύκολο στην ανάγνωση/κατανόηση

```
#include <iostream>
using namespace std;

//function declaration
void repchar(char, int);

//contains 3 calls to repchar()
int main() {
  repchar('*', 43);
  cout << "Data type Range" << endl;
  repchar('-', 43);
  cout << "char      [-128,127]" << endl;
  cout << "int       System Dep" << endl;
  repchar('*', 43);
}

//function definition
void repchar(char c, int n) {
  for (int i=0; i<n; i++) {
    cout << c;
  }
  cout << endl;
}
```

Part02/repchar.cpp

Επιστοφής τιμής

- Εντολή `return`:
  - Επιστρέφει την τιμή μιας έκφρασης
    Ο τύπος της τιμής θα πρέπει να συμφωνεί με αυτόν που έχει η δήλωση της συνάρτησης
  - Τερματίζει την εκτέλεση της συνάρτησης
    Κώδικας μετά την εντολή `return` δεν εκτελείται
  - Μόνο μια τιμή επιστρέφεται
    Ενώ μια συνάρτηση μπορεί να δέχεται πολλά ορίσματα, δεν μπορεί να επιστρέφει περισσότερες από μια τιμές

    υπάρχουν τρόποι να κάμψουμε αυτόν τον περιορισμό

```cpp
#include <iostream>
using namespace std;

//function declaration
float kgrtolbs(float kgs);

int main() {
  float lbs, kgs;

  cout << "Your weight in kgs:";
  cin >> kgs;
  lbs = kgrtolbs(kgs);

  cout << "Your weight in pounds is: "
       << lbs << endl;
  return 0;
}

//function definition
float kgrtolbs(float kgs) {
  float pounds = kgs / 0.453592;
  return pounds;
}
```

Part02/convert.cpp

Κλήση συνάρτησης

- Για την κλήση μιας συνάρτησης χρειάζεται το όνομα της συνάρτησης ακολουθούμενο από την λίστα των ορισμάτων της
  - π.χ. `lbs = kgrtolbs(kgs);`
    `repchar('*', 43);`
- Ορίσματα σε λάθος σειρά οδηγούν σε λάθος αποτέλεσμα ή σε σφάλμα κατά την μετάφραση
  - π.χ. η κλήση `repchar(43, '-')` θα προκαλούσε σφάλμα κατά την μετάφραση
- Αν η συνάρτηση είναι συνάρτηση-μέλος (μέθοδος) ενός αντικειμένου, τότε πρέπει να κληθεί από ένα αντικείμενο
  - π.χ. `Circle c1; c1.setRadious(1);`

```cpp
#include <iostream>
using namespace std;

//function declaration
float kgrtolbs(float kgs);

int main() {
  float lbs, kgs;

  cout << "Your weight in kgs:";
  cin >> kgs;
  lbs = kgrtolbs(kgs);

  cout << "Your weight in pounds is: "
       << lbs << endl;
  return 0;
}

//function definition
float kgrtolbs(float kgs) {
  float pounds = kgs / 0.453592;
  return pounds;
}
```

Part02/convert.cpp

Κλήση συναρτήσεων από συναρτήσεις

- Ήδη το κάνουμε αυτό:
  - η `main()` είναι συνάρτηση
- Η μοναδική απαίτηση:
  - Η δήλωση της συνάρτησης πρέπει να εμφανίζεται πρώτα
- Συχνά συναρτήσεις καλούν άλλες συναρτήσεις
  - Μια συνάρτηση μπορεί επίσης να καλεί τον εαυτό της $\rightarrow$ αναδρομή

```cpp
#include <iostream>
using namespace std;

bool isOdd(int number);
bool isEven(int number);

int main() {
  int input;
  cout << "Enter an integer number: ";
  cin >> input;
  if (isEven(input))
    cout << "It's even" << endl;
  else
    cout << "It's odd" << endl;
}

bool isOdd(int number) {
  if ((number%2)!=0) return true;
  return false;
}
bool isEven(int number) {
  if (!isOdd(number)) return true;
  return false;
}
```

Part02/odd_even.cpp

Κλήση συναρτήσεων από συναρτήσεις

- Ήδη το κάνουμε αυτό:
  - η `main()` είναι συνάρτηση
- Η μοναδική απαίτηση:
  - Η δήλωση της συνάρτησης πρέπει να εμφανίζεται πρώτα
- Συχνά συναρτήσεις καλούν άλλες συναρτήσεις
  - Μια συνάρτηση μπορεί επίσης να καλεί τον εαυτό της $\rightarrow$ αναδρομή
- Αντίστοιχα, μια μέθοδος εντός αντικειμένου μπορεί να καλέσει μια μέθοδο του ίδιου ή διαφερετικού αντικειμένου

```cpp
#include <iostream>
using namespace std;
#include <cmath>
#include "Point.cpp"

class LineSegment
{
private:
  Point start, end;

public:
  ...
  //function declaration
  double getLength() const;
  ...
};

double LineSegment::getLength() const {
  double dx, dy;
  dx = pow(start.getX()-end.getX(),2);
  dy = pow(start.getY()-end.getY(),2);
  return sqrt(dx + dy);
}
```

Part01/LineSegment.cpp

Αντικείμενα ως παράμετροι συναρτήσεων

- Οι παράμετροι των συναρτήσεων που έχουμε δει ως τώρα ήταν απλοί τύποι δεδομένων
- Ωστόσο, μια συνάρτηση μπορεί να δέχεται και αντικείμενα ως παραμέτρους
- Αντίστοιχα, μια συνάρτηση μπορεί να κατασκευάζει και να επιστρέψει ένα αντικείμενο

```
#include <iostream>
using namespace std;

class Fraction {
private:
  int numerator, denominator;
public:
  void setNumerator (int n);
  void setDenominator (int d);
  int getNumerator() const;
  int getDenominator() const;
  ...
};

Fraction add(Fraction f, Fraction g) {
  int n1 = f.getNumerator();
  int n2 = g.getNumerator();
  int d1 = f.getDenominator();
  int d2 = g.getDenominator();
  Fraction r;
  r.setNumerator(n1 * d2 + n2 * d1);
  r.setDenominator(d1 * d2);
  return r;
}
```

Part02/Fraction.cpp

Ασκήσεις

- Τι εκτυπώνουν τα παρακάτω προγράμματα;

```
#include <iostream>
using namespace std;

char mystery(int a, int b);

int main() {
  cout << mystery(10,9) << "ow" << endl;
  return 0;
}

char mystery(int a, int b) {
  if( a > b ) {
    return 'W';
  }
  else {
    return 'H';
  }
}
```

```
#include <iostream>
using namespace std;

void friendly();
void shy(int);

int main() {
  friendly(); shy(6);
  cout << "Again:" << endl;
  shy(2); friendly();
  cout << "End" << endl;
  return 0;
}

void friendly() {
  cout << "Hi!" << endl;
}
void shy(int a) {
  if (a < 5)
    return;
  cout << "Bye." << endl;
}
```

Μπορείτε να έχετε εντολή `return` μέσα σε συν/ση `void`.

Ασκήσεις

- Γράψτε μια δήλωση και έναν ορισμό συνάρτησης που παίρνει δύο ορίσματα τύπου int και επιστρέφει το άθροισμά τους.

- Γράψτε μια δήλωση και έναν ορισμό συνάρτησης που παίρνει ένα όρισμα τύπου double. Η συνάρτηση επιστρέφει τον χαρακτήρα ’+’ αν το όρισμα είναι θετικό και ’-’ αν το όρισμα είναι αρνητικό.

Μέρος 3°:
Αναδρομικές συναρτήσεις

Αναδρομή

- Αναδρομικές συναρτήσεις:
  - Συναρτήσεις που καλούν το εαυτό τους
  - Λύνουν την βασική περίπτωση χωρίς αναδρομή
- Διαιρούν το πρόβλημα σε:
  - Βασική περίπτωση που λύνουν “εύκολα”
  - Σε άλλες περιπτώσεις που μοιάζουν με το αρχικό πρόβλημα και τότε καλούν ένα αντίγραφο του εαυτού τους για την λύση τους

```cpp
#include <iostream>
using namespace std;

// Computes the i-th power of an integer
long power(long, long);

int main() {
    long base, exp, result;

    cout << "Enter the base: ";
    cin  >> base;

    cout << "Enter the exponent: ";
    cin  >> exp;

    result = power(base, exp);
    cout << base << "^" << exp << " = "
         << result << endl;
}

long power(long base, long exponent) {
    if (exponent==0) return 1;
    return base * power(base,exponent-1);
}
```

Part02/Recursion/power.cpp

Παραγοντικό (n!)

- `n! = 1 * 2 * ... * n`

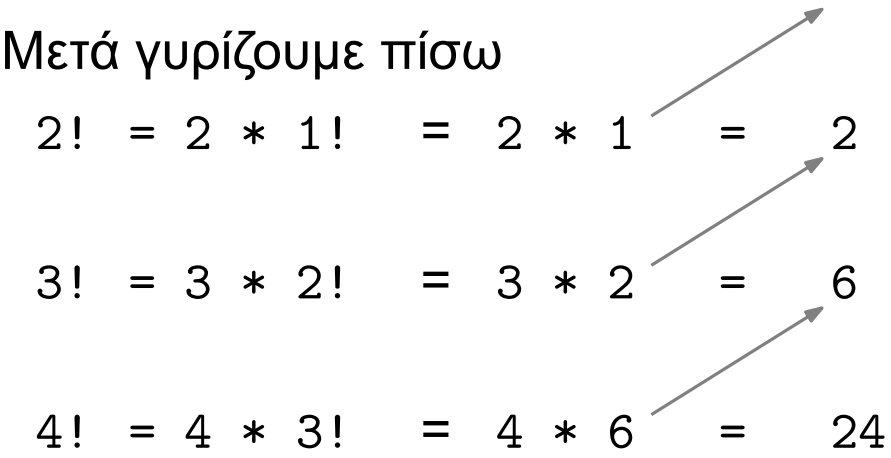
  ή

  `n! = n * (n-1)!  με  1! = 0! =1`

- Το παραγοντικό μπορεί να υπολογιστεί αναδρομικα:

  - Βάση της αναδρομής (`1! = 0! = 1`)

  - Μετά γυρίζουμε πίσω



- **Προσοχή:** Το παραγοντικό είναι εκθετική συνάρτηση

```cpp
#include <iostream>
using namespace std;

// Function to compute n!
unsigned long factorial(unsigned long n);

int main() {
  unsigned long input, result;

  cout << "Enter an integer number: ";
  cin  >> input;

  result = factorial(input);

  cout << input << "! = "
       << result << endl;
}

unsigned long factorial(unsigned long n)
{
  if (n<=1)                    //base case
    return 1;
  return n * factorial(n-1); //recursion
}
```

Part02/Recursion/factorial.cpp

Η σειρά Fibonacci

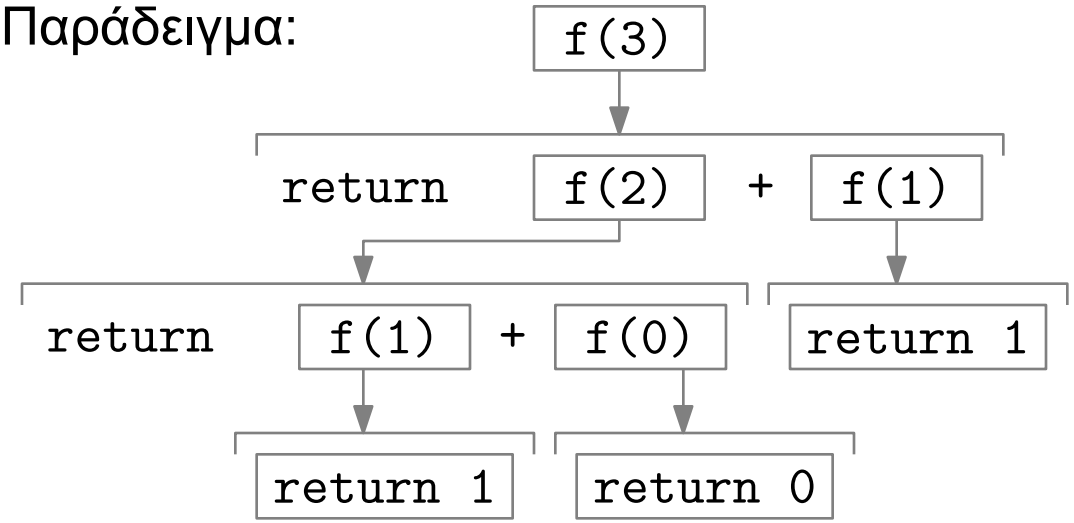
- Κάθε αριθμός είναι άθροισμα των δύο προηγουμένων: `0, 1, 1, 2, 3, 5, 8...`

- Αναδρομικός ορισμός:

  - `fib( n ) = fib( n-1 ) + fib( n-2 )`

  - Βάση της αναδρομής: `fib(0) = 0, fib(1) = 1`

- Παράδειγμα:




```
#include <iostream>
using namespace std;

// Computes the n-th Fibonacci number
unsigned long fibonacci(unsigned long);

int main() {
  unsigned long input, result;

  cout << "Enter an integer number: ";
  cin  >> input;

  result = fibonacci(input);

  cout << "Fib(" << input << ") = "
       << result << endl;
}

unsigned long fibonacci(unsigned long n)
{
  if (n==0) return 0;
  if (n==1) return 1;
  return fibonacci(n-1) + fibonacci(n-2);
}
```

Part02/Recursion/fibonacci.cpp

Πρώτοι αριθμοί

- Διαιρούνται μόνο με το 1 και τον εαυτό τους, Π.χ. `2, 3, 5, 7, 11, 23...`
- **Πρόβλημα:** Με δεδομένο έναν θετικό ακέραιο n να εξεταστεί αν ο n είναι πρώτος
- Ένας απλός αλγόριθμος επίλυσης του προβλήματος:
  1. Εξέτασε κάθε αριθμό από το 2 ως το $n - 1$
  2. Αν κάποιος από αυτούς τους αριθμούς διαιρεί τον n, τότε ο n είναι σύνθετος
  3. Διαφορετικά, ο n είναι πρώτος

```cpp
#include <iostream>
using namespace std;

// Check whether a number is prime
bool isPrime(int, int = 2);

int main() {
  int input;

  cout << "Enter an integer number: ";
  cin  >> input;

  if (isPrime(input))
   cout<<"The given number is prime";
  else
   cout<<"The given number is composite";
}

bool isPrime(int n, int i) {
  if (n == 1) return false;
  if (n == 2 || i >= n) return true;
  if (n % i == 0) return false;
  return isPrime(n, i + 1);
}
```

Part02/Recursion/prime_numbers.cpp

Πρώτοι αριθμοί

- Διαιρούνται μόνο με το 1 και τον εαυτό τους, Π.χ. `2, 3, 5, 7, 11, 23...`
- **Πρόβλημα:** Με δεδομένο έναν θετικό ακέραιο n να εξεταστεί αν ο n είναι πρώτος
- Ένας απλός αλγόριθμος επίλυσης του προβλήματος:
  1. Εξέτασε κάθε αριθμό από το 2 ως το $n - 1$
  2. Αν κάποιος από αυτούς τους αριθμούς διαιρεί τον n, τότε ο n είναι σύνθετος
  3. Διαφορετικά, ο n είναι πρώτος

  ↳ Είναι ορθή η προσέγγιση;

```cpp
#include <iostream>
using namespace std;

// Check whether a number is prime
bool isPrime(int, int = 2);

int main() {
  int input;

  cout << "Enter an integer number: ";
  cin  >> input;

  if (isPrime(input))
   cout<<"The given number is prime";
  else
   cout<<"The given number is composite";
}

bool isPrime(int n, int i) {
  if (n == 1) return false;
  if (n == 2 || i >= n) return true;
  if (n % i == 0) return false;
  return isPrime(n, i + 1);
}
```

Part02/Recursion/prime_numbers.cpp

Αλγόριθμοι

- Ένας αλγόριθμος πρέπει να είναι:
  - Σαφώς ορισμένος
  - Αποτελεσματικός
  - Πεπερασμένος, δηλ. τερματίζει μετά από έναν αριθμό βημάτων
- Όταν τερματίζει έχει το επιθυμητό αποτέλεσμα !
- **Στο παράδειγμα:**
  - Ο αλγόριθμος ελέγχει όλες τις τιμές του *i* μεταξύ 1 και *n*
  - Όταν αυτό συμβεί, ο αλγόριθμος **τερματίζει**
  - Όταν τερματίσει έχουν ελεγχθεί όλοι οι πιθανοί διαιρέτες του *n*

```
#include <iostream>
using namespace std;

// Check whether a number is prime
bool isPrime(int, int = 2);

int main() {
  int input;

  cout << "Enter an integer number: ";
  cin  >> input;

  if (isPrime(input))
   cout<<"The given number is prime";
  else
   cout<<"The given number is composite";
}

bool isPrime(int n, int i) {
  if (n == 1) return false;
  if (n == 2 || i >= n) return true;
  if (n % i == 0) return false;
  return isPrime(n, i + 1);
}
```

Part02/Recursion/prime_numbers.cpp

Βελτίωση της αποδοτικότητας αλγορίθμου

- Αν το πρόγραμμα ελέγξει ότι ο αριθμός δεν διαιρείται με το 2 τότε δεν χρειάζεται να ελεγχθούν οι υπόλοιποι άρτιοι αριθμοί.

  Απόδειξη: Προφανής

- Ελέγχουμε μόνο τους αριθμούς μέχρι το ακέραιο μέρος του $\sqrt{n}$

  Απόδειξη:

  Έστω ότι ο n διαιρείται από έναν αριθμό $d_1 \in \mathbb{N}$

  $\Rightarrow \exists\ d_2 \in \mathbb{N} :\ \ n = d_1 \cdot d_2$

  $\Rightarrow d_1 \leq \sqrt{n}$ ή $d_2 \leq \sqrt{n}$

  $\Rightarrow$ δεν χρειάζεται να ελεγχθούν οι αριθμοί $\geq \sqrt{n}$

Βελτίωση της αποδοτικότητας αλγορίθμου

- Αν το πρόγραμμα ελέγξει ότι ο αριθμός δεν διαιρείται με το 2 τότε δεν χρειάζεται να ελεγχθούν οι υπόλοιποι άρτιοι αριθμοί.

  Απόδειξη: Προφανής

- Ελέγχουμε μόνο τους αριθμούς μέχρι το ακέραιο μέρος του $\sqrt{n}$

  Απόδειξη:

  Έστω ότι ο n διαιρείται από έναν αριθμό $d_1 \in \mathbb{N}$

  $\Rightarrow \exists\ d_2 \in \mathbb{N} :\ \ n = d_1 \cdot d_2$

  $\Rightarrow d_1 \leq \sqrt{n}$ ή $d_2 \leq \sqrt{n}$

  $\Rightarrow$ δεν χρειάζεται να ελεγχθούν οι αριθμοί $\geq \sqrt{n}$

```
bool isPrime(int n, int i) {
 if (n == 1) return false;
 if (n == 2 || i > sqrt(n)) return true;
 if (n % i == 0) return false;

 if (i==2) {
   return isPrime(n, i + 1);
 }
 return isPrime(n, i + 2);
}
```

Μέρος 4°:

Παράμετροι κλήσης με τιμή/αναφορά

Παράμετροι κλήσης με τιμή

- Στην κλήση της συνάρτησης τοποθετούνται αντίγραφα των τιμών
- Ουσιαστικά είναι σαν τοπικές μεταβλητές
- Αν αλλάξουν τιμή, τότε η μεταβολή είναι “τοπική”
  - Η συνάρτηση δεν έχει πρόσβαση στη “πραγματική παράμετρο”
- Αυτή είναι η εξ’ορισμού κλήση
  - σε όλα τα παραδείγματα ως τώρα

```
#include <iostream>
using namespace std;

double cost(int hours, int minutes);
const double RATE = 9.00; //cost per 15'

int main() {
  int hours, minutes; double fee;

  cout << "Enter hours and minutes: ";
  cin >> hours >> minutes;

  fee = cost(hours, minutes);

  cout << "The cost for " << hours <<
  "h & " << minutes << "m is: " << fee;
}

//Compute the cost for hours and minutes
double cost(int hours, int minutes) {
  minutes = hours * 60 + minutes;
  double quarterHours = minutes/15.0;
  return quarterHours * RATE;
}
```

Παράμετροι κλήσης με τιμή

- Στην κλήση της συνάρτησης τοποθετούνται αντίγραφα των τιμών
- Ουσιαστικά είναι σαν τοπικές μεταβλητές
- Αν αλλάξουν τιμή, τότε η μεταβολή είναι “τοπική”
  - Η συνάρτηση δεν έχει πρόσβαση στη “πραγματική παράμετρο”
- Αυτή είναι η εξ’ορισμού κλήση
  - σε όλα τα παραδείγματα ως τώρα

```
#include <iostream>
using namespace std;

double cost(int hours, int minutes);
const double RATE = 9.00; //cost per 15'

int main() {
  int hours, minutes; double fee;

  cout << "Enter hours and minutes: ";
  cin >> hours >> minutes;

  fee = cost(hours, minutes);

  cout << "The cost for " << hours <<
  "h & " << minutes << "m is: " << fee;
}

//Compute the cost for hours and minutes
double cost(int hours, int minutes) {
  minutes = hours * 60 + minutes;
  double quarterHours = minutes/15.0;
  return quarterHours * RATE;
}
```

Part02/by_value.cpp

Δεν αλλάζει η τιμή της μεταβλητής `minutes` από την κλήση της `cost()`

Παράμετροι κλήσης με αναφορά

- Χρησιμοποιούνται για να έχουν πρόσβαση στις πραγματικές παραμέτρους
- Τα δεδομένα που καλεί κάποιος μπορεί να αλλάξουν κατά το κάλεσμα μιας συν/σης
- Δηλώνονται με & μετά τον τύπο στη λίστα παραμέτρων
- Συνήθως χρησιμοποιούνται για είσοδο
  - Αυτός που καλεί θέλει τις τιμές που θα διαβάσει η συν/ση
- Τρόπος επιστροφής περισ/ρων από μιας τιμής

```cpp
#include <iostream>
using namespace std;

void getNumbers(int& x, int& y);
void swap(int& x, int& y);

int main() {
  int x, y;
  getNumbers(x,y);
  swap(x,y);
  cout << "After swapping: ";
  cout << "x=" << x << ", y=" << y;
}

void getNumbers(int& x, int& y) {
  cout << "x="; cin >> x;
  cout << "y="; cin >> y;
}

void swap(int& x, int& y) {
  int temp = x;
  x = y;
  y = temp;
}
```

Part02/by_reference.cpp

Μικτές λίστες παραμέτρων

- Συνδυασμός παραμέτρων στο κάλεσμα της συνάρτησης
- Η λίστα παραμέτρων μπορεί να περιέχει παραμέτρους με τιμή και παραμέτρους με αναφορά
- Η σειρά των ορισμάτων στη λίστα είναι σημαντική

```
#include <iostream>
using namespace std;

void multiplyBy(int& number,
                int factor);

int main() {
  int x = 25;

  cout << "The original value of x is "
       << x << endl;

  multiplyBy(x, 2);

  cout << "The new value of x is "
       << x << endl;

  return 0;
}

void multiplyBy(int& number,
                int factor) {
  number *= factor;
}
```

Part02/by_reference_value.cpp

Σταθερές Παράμετροι αναφοράς

- Οι παράμετροι με αναφορά εγκυμονούν κινδύνους
  - Τα δεδομένα μπορούν να αλλάξουν
  - Μερικές φορές είναι επιθυμητό, άλλες όχι
- Για να “προστατεύσουμε” τα δεδομένα, περνώντας παράλληλα παραμέτρους με αναφορά:
  - Χρήση της δεσμευμένης λέξης const
  - Οι παράμετροι γίνονται “μόνο για διάβασμα” (read only) από την συνάρτηση
  - Δεν επιτρέπονται αλλαγές στο σώμα της συνάρτησης

```
#include <iostream>
using namespace std;

void multiplyBy(int& number,
                const int& factor);

int main() {
  int x = 25;

  cout << "The original value of x is "
       << x << endl;

  multiplyBy(x, 2);

  cout << "The new value of x is "
       << x << endl;

  return 0;
}

void multiplyBy(int& number,
                const int& factor) {
  number *= factor;
}
```

Part02/by_reference_value.cpp

Μηχανισμός κλήσης με αναφορά

- Τι πραγματικά περνάμε σαν αναφορά;
- Μια αναφορά στην πραγματική παράμετρο όταν κάλεσε την συνάρτηση
  - Αναφέρεται στην θέση μνήμης της πραγματικής παραμέτρου
  - Καλείται συχνά ως “διεύθυνση”, που είναι ένα μοναδικό νούμερο που δείχνει σε διακριτή θέση μνήμης

Ασκήσεις

- Τι εκτυπώνει το παρακάτω πρόγραμμα;

```
#include <iostream>
using namespace std;

void figureMeOut(int& x, int y, int& z);

int main() {
  int a = 10, b = 20, c = 30;

  figureMeOut(a, b, c);

  cout << a << " " << b << " " << c << endl;

  return 0;
}

void figureMeOut(int& x, int y, int& z) {
  cout << x << " " << y << " " << z << endl;

  x = 1; y = 2; z = 3;

  cout << x << " " << y << " " << z << endl;
}
```

Ασκήσεις

- Βρείτε το σφάλμα στο παρακάτω τμήμα κώδικα:

```
#include <iostream> using namespace std;

const double RATE = 9.00; //cost per 15'
double cost(int hours, int minutes);

int main() {
  int hours, minutes; double fee;

  cout << "Enter hours and minutes: ";
  cin >> hours >> minutes;

  fee = cost(hours, minutes);

  cout << "The cost for " << hours <<
  "h & " << minutes << "m is: " << fee;
}

// Compute the cost for hours and minutes
double cost(int hours , int minutes) {
  int minutes = hours * 60 + minutes;
  int quarterHours = minutes/15;
  return quarterHours * RATE;
}
```

Ασκήσεις

- Βρείτε το σφάλμα στο παρακάτω τμήμα κώδικα:

```
#include <iostream> using namespace std;

const double RATE = 9.00; //cost per 15'
double cost(int hours, int minutes);

int main() {
  int hours, minutes; double fee;

  cout << "Enter hours and minutes: ";
  cin >> hours >> minutes;

  fee = cost(hours, minutes);

  cout << "The cost for " << hours <<
  "h & " << minutes << "m is: " << fee;
}

// Compute the cost for hours and minutes
double cost(int hours , int minutes) {
  int minutes = hours * 60 + minutes;
  int quarterHours = minutes/15;
  return quarterHours * RATE;
}
```

Απαιτούνται δύο γραμμές κώδικα

Κλασικό λάθος: Δήλωση παραμέτρου “ξανά” μέσα σε συν/ση

Μέρος 5°:

Προεπιλεγμένα ορίσματα - Υπερφόρτωση συναρτήσεων

Προεπιλεγμένα ορίσματα

- Κατά την κλήση επιτρέπει την αποφυγή μερικών ορισμάτων
  - Μόνο τα τελευταία ορίσματα μπορεί να είναι προεπιλεγμένα (defaulted)
  - Δεν θα πρέπει να υπάρχει “σύγχυση” με άλλες υπογραφές συναρτήσεων → σφάλμα στον μεταφραστή
- Υποστηρίζει την αποφυγή επανάληψης κώδικα [code duplication]

```
#include <iostream>
using namespace std;

//function declaration with default args
void repchar(char='*', int=43);

//contains 3 calls to repchar()
int main() {
  repchar();
  cout << "Data type Range" << endl;
  repchar('-');
  cout << "char      [-128,127]" << endl;
  cout << "int       System Dep" << endl;
  repchar('+', 45);
}

//function definition; no defaults here!
void repchar(char c, int n) {
  for (int i=0; i<n; i++) {
    cout << c;
  }
  cout << endl;
}
```

Part02/repchar.cpp

Προεπιλεγμένα ορίσματα

- Κατά την κλήση επιτρέπει την αποφυγή μερικών ορισμάτων
  - Μόνο τα τελευταία ορίσματα μπορεί να είναι προεπιλεγμένα (defaulted)
  - Δεν θα πρέπει να υπάρχει “σύγχυση” με άλλες υπογραφές συναρτήσεων → σφάλμα στον μεταφραστή
- Υποστηρίζει την αποφυγή επανάληψης κώδικα [code duplication]

```
#include <iostream>
using namespace std;

//function declaration with default args
void repchar(char='*', int=43);

//contains 3 calls to repchar()
int main() {
  repchar();
  cout << "Data type Range" << endl;
  repchar('-');
  cout << "char      [-128,127]" << endl;
  cout << "int       System Dep" << endl;
  repchar('+', 45);
}

//function definition; no defaults here!
void repchar(char c, int n) {
  for (int i=0; i<n; i++) {
    cout << c;
  }
  cout << endl;
}
```

Part02/repchar.cpp

Τα προκαθορισμένα ορίσματα δεν δίνονται στο κυρίως σώμα της συν/σης

Υπερφόρτωση συναρτήσεων

- Συναρτήσεις με το ίδιο όνομα
  - Διαφορετική λίστα παραμέτρων
  - Δυο διαφορετικές δηλώσεις συναρτήσεων
- Υπογραφή συνάρτησης:
  - Όνομα συνάρτησης & λίστα παραμέτρων
  - Μοναδικά για κάθε ορισμό συνάρτησης
- Επιτρέπει ίδια ενέργεια σε διαφορετικά δεδομένα

```cpp
#include <iostream>
using namespace std;

//Two functions with the same name
int add(int a, int b);
double add(double a, double b);

int main() {
  cout << add(10,20) << endl;
  cout << add(10.00,20.00) << endl;
  return 0;
}

//Computes the sum of two ints
int add(int a, int b) {
  return a + b;
}

//Computes the sum of two doubles
double add(double a, double b) {
  return a + b;
}
```

Part02/function_overloading.cpp

Υπερφόρτωση συναρτήσεων

- Ποια συνάρτηση καλείται;
- Εξαρτάται από το ίδιο το κάλεσμα:
  - `int result = add(10, 20);`
  - `double sum = add(10.50, 20.75);`
- Ο μεταφραστής επιλύει τέτοια θέματα βασιζόμενος στην υπογραφή της συνάρτησης στο κάλεσμα
  - “Ταιριάζει” το κάλεσμα με την αντίστοιχη συνάρτηση

```cpp
#include <iostream>
using namespace std;

//Two functions with the same name
int add(int a, int b);
double add(double a, double b);

int main() {
  cout << add(10,20) << endl;
  cout << add(10.00,20.00) << endl;
  return 0;
}

//Computes the sum of two ints
int add(int a, int b) {
  return a + b;
}

//Computes the sum of two doubles
double add(double a, double b) {
  return a + b;
}
```

Part02/function_overloading.cpp

Υπερφόρτωση συναρτήσεων

- Αντίστοιχα γίνεται η υπερφόρτωση σε μια κλάση:
  - Η υπερφόρτωση μπορεί να αφορά συναρτήσεις-μέλη (μεθόδους)
  - ή ακόμη και κατασκευαστές
- Η υπερφόρτωση μπορεί να γίνει και μέσω των προεπιλεγμένων ορισμάτων

```
#include <iostream>
using namespace std;

class Complex {
private:
  double real; // Real part
  double imag; // Imaginary part

public:
  Complex()  {
    real = 0;
    imag = 0;
  }
  Complex(double r, double i) {
    real = r;
    imag = i;
  }
  void increase(double r=0, double i=0) {
    real += r;
    imag += i;
  }
  ...
};
```

Part01/Complex.cpp

Επίλυση Υπερφόρτωσης

- 1: Απόλυτο ταίριασμα
  - Κοιτάζει για ίδια υπογραφή
  - Όπου δεν απαιτείται κάποια μετατροπή τύπου
- 2: Συμβατό ταίριασμα
  - Κοιτάζει για “συμβατή” υπογραφή όπου μια αυτόματη μετατροπή τύπου είναι δυνατή
  - Είτε με προώθηση (π.χ. `int` → `double`)
    - δεν χάνονται δεδομένα
  - Είτε με αποκοπή (π.χ. `double` → `int`)
    - πιθανή απώλεια δεδομένων
- 3: Ομοιότητες σε όλα τα επίπεδα
  - Σφάλμα στο μεταφραστή

```
void foo(int n, int m);
void foo(double n, double m);

int main() {
  foo(33, 44);
}
```

```
void foo(double n, double m);

int main() {
  foo(33, 44);
}
```

```
void foo(int n, double m);
void foo(double n, int m);

int main() {
  foo(33, 44);
}
```

Μέρος 6°:
Παραδείγματα

Μέγιστος Κοινός Διαιρέτης

- Ένα απλός αλγόριθμος:
  - Υπολόγισε το ελάχιστο των δύο αριθμών
  - Δοκίμασε διαδοχικά τιμές μία-μία, ξεκινώντας από το ελάχιστο και ελαττώνοντας προς μικρότερους αριθμούς.
  - Ο μέγιστος κοινός διαιρέτης είναι ο πρώτος αριθμός ο οποίος διαιρεί και τους δύο αριθμούς

```cpp
#include <iostream>
using namespace std;

int gcd(unsigned int a, unsigned int b);

int main() {
  unsigned int a = 98, b = 56;
  cout << "GCD of " << a << " and " << b
       << " is " << gcd(a, b);
  return 0;
}

int gcd(unsigned int a, unsigned int b){
  //find the minimum of a and b
  int result = min(a, b);

  for (int i=result; i>0; i--)
  {
    if (a % i == 0 && b % i == 0)
    {
      return i;
    }
  }
}
```

Part02/Examples/gcd.cpp

Μέγιστος Κοινός Διαιρέτης

- Ένας πιο αποδοτικός αλγόριθμος
  ο αλγόριθμος του Ευκλείδη:
  - Αν οι δύο αριθμοί είναι ίσοι, ο μέγιστος κοινός διαιρέτης ισούται μ' αυτούς.
  - Αν δεν είναι ίσοι, αντικαθιστούμε τον μεγαλύτερο με τη διαφορά τους και επαναλαμβάνουμε τη σύγκριση

- Donald Knuth: Ο αλγόριθμος του Ευκλείδη είναι ο παππούς όλων των αλγορίθμων.

  "[The Euclidean algorithm] is the granddaddy of all algorithms, because it is the oldest nontrivial algorithm that has survived to the present day."

  D. Knuth, The Art of Computer Programming Vol. 2, p. 318

```cpp
#include <iostream>
using namespace std;

int gcd(unsigned int a, unsigned int b);

int main() {
  unsigned int a = 98, b = 56;
  cout << "GCD of " << a << " and " << b
       << " is " << gcd(a, b);
  return 0;
}

int gcd(unsigned int a, unsigned int b){
  // Everything divides 0
  if (a == 0) return b;
  if (b == 0) return a;

  // base case
  if (a == b) return a;

  // recursive step
  if (a > b) return gcd(a-b, b);
  return gcd(a, b-a);
}
```

Part02/Examples/gcd.cpp

Υπολογισμός δύναμης με αναδρομή

- Έχουμε ήδη δει ένα απλό αναδρομικό αλγόριθμο, βασιζόμενο στην αναδρομική σχέση:
  - $a^b = a \cdot (a^{b-1})$
- Αν $b = 10$ πόσες κλήσεις (επαναλήψεις) θα κάνει;
- Μπορούμε πιο αποτελεσματικά (με λιγότερες αναδρομικές κλήσεις);

```cpp
#include <iostream>
using namespace std;

// Computes the i-th power of an integer
long power(long, long);

int main() {
  long base, exp;

  cout << "Enter the base: ";
  cin  >> base;
  cout << "Enter the exponent: ";
  cin  >> exp;

  cout << base << "^" << exp << " = "
       << power(base, exp) << endl;
}

long power(long base, long exponent) {
  if (exponent==0) return 1;
  return base * power(base,exponent-1);
}
```

Part02/power.cpp

Υπολογισμός δύναμης με αναδρομή

- Ένας πιο αποδοτικός αλγόριθμος βασίζεται στην αναδρομική σχέση:
  - $a^b = (a^{b/2}) \cdot (a^{b/2})$, για b άρτιο
  - $a^b = (a^{b/2}) \cdot (a^{b/2}) \cdot a$, για b περιττό
- Πόσες αναδρομικές κλήσεις γίνοται;

```cpp
#include <iostream>
using namespace std;

// Computes the i-th power of an integer
long power(long, long);

int main() {
  long base, exp;

  cout << "Enter the base: ";
  cin  >> base;
  cout << "Enter the exponent: ";
  cin  >> exp;

  cout << base << "^" << exp << " = "
       << power(base, exp) << endl;
}

long power(long base, long exponent) {
  if (exponent == 0) return 1;
  long half = power(base, exponent/2) ;
  if (exponent%2 == 0) return half * half;
  return half * half * base;
}
```

Part02/power_revisited.cpp

Τετράγωνο αριθμού

- Το τετράγωνο ενός ακεραίου αριθμού n μπορεί να υπολογιστεί προσθέτοντας όλους τους ακέραιους από το 1 έως το n και επιστρέφοντας πάλι πίσω στο 1,
  - π.χ. $4^2 = 1 + 2 + 3 + 4 + 3 + 2 + 1 = 16$
- Είναι ο παραπάνω αλγόριθμος σωστός;

```
#include <iostream>
using namespace std;

// Computes the square of an integer
int square(int);

int main() {
  int n;

  cout << "Enter a number: ";
  cin  >> n;

  cout << n << "^2 = "
       << square(n) << endl;
}

int square(int n) {
  int result = 0;
  for(int i = 1; i <= n; i++)
    result = result + i;
  for(int i = n-1; i >= 1; i--)
    result = result + i;
  return result;
}
```

Part02/Examples/square.cpp

Φίλιοι αριθμοί

- Δύο θετικοί ακέραιοι αριθμοί είναι “φίλιοι” [amicable] αν ο καθένας ισούται με το άθροισμα όσων διαιρούν τον άλλον (λαμβάνονται υπόψη μόνον οι γνήσιοι διαιρέτες).
  - Οι πιο διάσημοι φίλιοι αριθμοί είναι οι αριθμοί 220 και 284 (αποδίδονται στον Πυθαγόρα).
  - Διαιρέτες του 220: 1, 2, 4, 5, 10, 11, 20, 22, 44, 55, 110
    Άθροισμα=284
  - Διαιρέτες του 284: 1, 2, 4, 71, 142
    Άθροισμα=220

```cpp
#include <iostream>
using namespace std;

bool amicable(int x, int y);

int main() {
  //print all amicable pairs in [1,1500]
  for (int i=1; i<=1500; i++)
    for (int j=i; j<=1500; j++)
      if (amicable(i,j))
        cout << i << ", " << j << endl;
}

bool amicable(int x, int y) {
  int xsum=0, ysum=0;

  for(int i = 1; i < x; i++)
    if(x%i == 0) xsum = xsum + i;

  for(int i = 1; i < y; i++)
    if(y%i == 0) ysum = ysum + i;

    return (xsum == y && ysum == x);
}
```

Part02/Examples/amicable.cpp

Μέρος 7°:

Συναρτήσεις: Κανόνες εμβέλειας

Τοπικές μεταβλητές

- Δηλώνονται μέσα σε μια συνάρτηση
- **Εμβέλεια:** μέσα σε αυτή την συνάρτηση

```
// A simple program to illustrate
// the usage of local variables
#include<iostream>
using namespace std;

int teenager() {
  // this variable is local to the
  // function teenager() and cannot
  // be accessed outside the function
  int age = 18;
  return age;
}

int main() {
  cout << "Teenager's age: " << age;
  return 0;
}
```

```
$ Error: age was not declared in this scope
```

Τοπικές μεταβλητές

- Δηλώνονται μέσα σε μια συνάρτηση
- **Εμβέλεια:** μέσα σε αυτή την συνάρτηση
  - Αν μια μεταβλητή είναι τοπική, τότε μπορείτε να έχετε μια άλλη μεταβλητή με το ίδιο όνομα σε μια άλλη συνάρτηση
  - Δυο διαφορετικές μεταβλητές με το ίδιο όνομα

```
// Another program to illustrate
// the usage of local variables
#include<iostream>
using namespace std;

int teenager() {
  // this variable is local to the
  // function teenager()
  int age = 18;
  return age;
}

int retired() {
  // this variable is local to the
  // function retired()
  int age = 65;
  return age;
}

int main() {
  cout<<"Teenager's age: "<<teenager();
  cout<<"Retired's age: "<<retired();
  return 0;
}
```

Καθολικές μεταβλητές

- Μεταβλητές που δεν ορίζονται μέσα σε μια συνάρτηση ονομάζονται καθολικές
- Δηλώνονται στην αρχή του προγράμματος:
  - έξω από το σώμα όλων των συναρτήσεων
  - μπορούν να χρησιμοποιηθούν σε οποιαδήποτε συνάρτηση ή κλάση
- Συνήθης τακτική:
  - Δηλώνονται μετά τις οδηγίες `include <...>` και την οδηγία `using ...`

```
#include <iostream>
#include <cmath>
using namespace std;

const double PI = 3.14159;

double area(double radius);
double volume(double radius);

int main() {
  double r;
  cout << "Enter radius";  cin >> r;

  cout << "Volume: " << volume(r) <<
          ", area: " << area(r) << endl;
}

double area(double radius) {
  return PI * pow (radius,2);
}

double volume(double radius) {
  return (4.0/3.0) * PI * pow (radius,3);
}
```

Σύνθηκες εντολές (μπλόκ)

- Μπλοκ: Τμήμα εντολών μέσα σε `{..}`
- Μια μεταβλητή που δηλώνεται μέσα σε μπλοκ έχει εμβέλεια μόνο σε αυτό το μπλοκ
  - τοπική μεταβλητή του μπλοκ
  - ο ορισμός της ισχύει από την δήλωση της μέχρι την }
- **Παρατήρηση:** Οι συναρτήσεις είναι μπλοκ

```
int sum = 0;
for(int i = 0; i < 10; i++)
{
  cout << i << endl;
  sum += i;
}
cout << sum << endl;
cout << i << endl; //<-- error!
```

```
while(true)
{
  cout << "Give a number, zero to exit:";
  int num;
  cin >> num;
  if(num == 0)
    break;
  else
    cout "Non zero!!" << endl;
}
```

Σύνθηκες εντολές (μπλόκ)

- Μπλοκ: Τμήμα εντολών μέσα σε `{..}`
- Μια μεταβλητή που δηλώνεται μέσα σε μπλοκ έχει εμβέλεια μόνο σε αυτό το μπλοκ
  - τοπική μεταβλητή του μπλοκ
  - ο ορισμός της ισχύει από την δήλωση της μέχρι την }
- **Παρατήρηση:** Οι συναρτήσεις είναι μπλοκ
- Αν ένα όνομα μεταβλητής χρησιμοποιείται σε δυο μπλοκ (το ένα μέσα στο άλλο) τότε είναι δυο διαφορετικές μεταβλητές με το ίδιο όνομα

```cpp
#include <iostream>

int x = 1;  //global variable

void foo();
void boo();

int main() {
  int x = 5;  //local variable
  foo();
  boo();
  std::cout << x << std::endl;
}

void foo() {
  int x = 14;  //local variable
  x++;
  std::cout << x << std::endl;
}

void boo() {
  x *= 10;
  std::cout << x << std::endl;
}
```

Part02/variable_scope.cpp

Εμφωλιασμένα μπλοκ

- Τα προγράμματα στην C++ αποτελούν μια σειρά από εμφωλιασμένα μπλοκς (nested blocks).
  - Αρχικά βρίσκονται στο επίπεδο 0.
  - Κάθε φορά που εισάγουμε ένα νέο εμφωλιασμένο μπλοκ, τότε πηγαίνουμε στο επόμενο επίπεδο.
- Κανόνες:
  - Η εσωτερική μεταβλητή δεν μπορεί να προσπελαστεί από το έξω μπλοκ
  - Η εξωτερική μεταβλητή δεν μπορεί να προσπελαστεί από το μέσα μπλοκ

```
#include <iostream>
using namespace std;

int main() {
  int weeks = 3;
  int days = 7;

  // outer loop prints weeks
  for (int i=1; i<=weeks; i++)
  {
    cout << "Week: " << to_string(i);
    cout << endl;

    // inner loop prints days
    for (int j=1; j<=days; j++)
    {
      cout << "  Day: " << to_string(j);
      cout << endl;
    }
  }
}
```

Επιπλέον υλικό

- Κεφάλαιο 5:
  R. Lafore, Αντικειμενοστρεφής προγραμματισμός με τη C++,
  ISBN: 960-209-904-6, εκδόσεις Κλειδάριθμος, 2006.