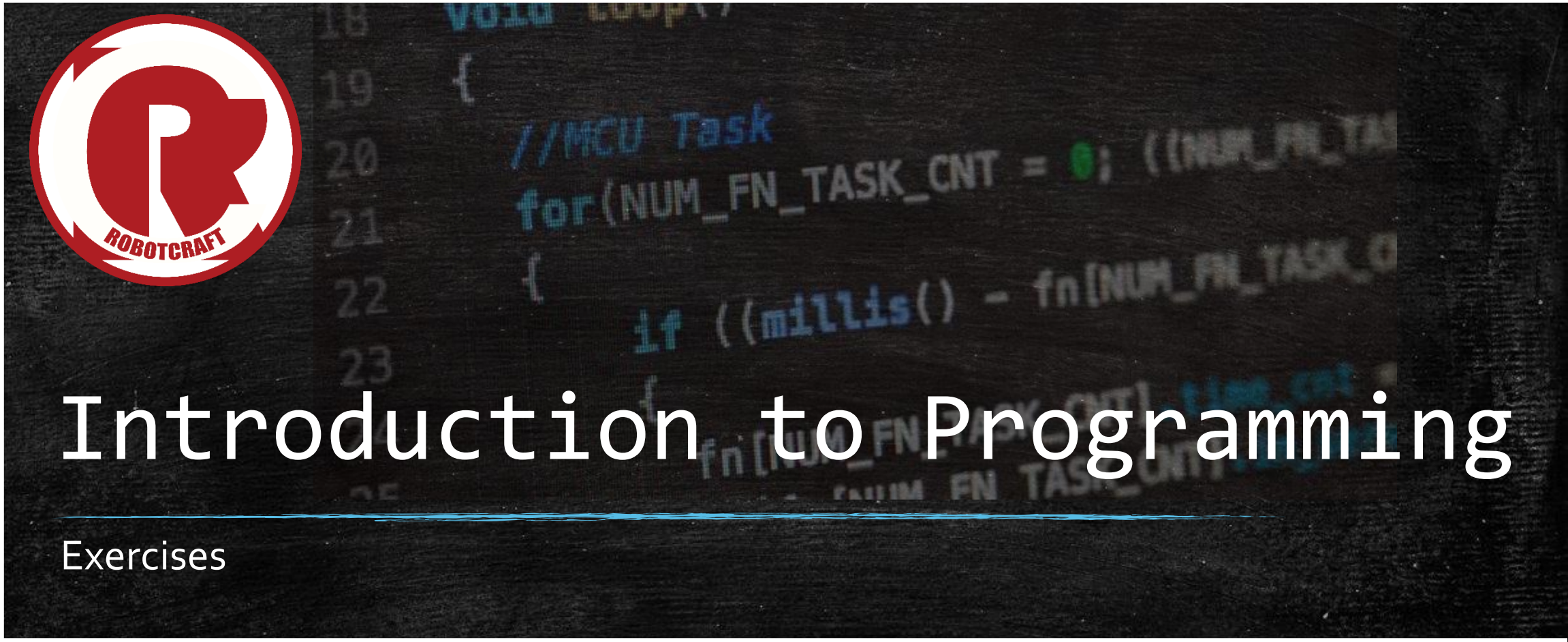
ROBOTCRAFT©

ROSin

INGENIARIUS



Introduction to Programming

Exercises

Micael S. Couceiro micael@ingeniarius.pt

Panagiotis Karfakis panagiotis@ingeniarius.pt

Craft #2

02/07/2024

Content

Exercise 1. Temperature check

Exercise 1. Temperature check

Create a program that asks a *temperature* number to the user and checks for two conditions:

1) if *temperature* is less than 10 print “cold”
2) else if *temperature* is greater than 35 print “hot”.
3) else print “normal”

* Improve the previous program by asking the user **two numbers** instead of one and output the results of both.

Exercise 2. Functions

Exercise 2: Functions

1) Create a function which prints “Hello world” message.
2) Improve the function to receive chars as an input and print them
3) Create another function to return the sum two floats (pass as arguments).
4) Create a function *checkAge()* which checks if a person is an adult . The function must have a **bool** return type and an **integer** argument.

Organize the code from exercise 1 into **functions**, creating the function *checkTemperature()* and *readNumber()* to read the keyboard typed number .

Exercise 3. For-loop

Exercise 3. For-loop

1) Write a program to display the first 10 natural numbers.
2) Modify a program to calculate the sum of first n natural numbers. Positive integers 1,2,3...n are known as natural numbers.
3) Modify the program that asks for a amount of numbers to input and then calculates the sum of the next numbers which are being typed.

* Improve the program with functions

```
The first 10 natural numbers are:
1 2 3 4 5 6 7 8 9 10

Enter a positive integer: 3
Sum = 6

Enter the amount of numbers to input: 3
Input num: 1
Input num: 0
Input num: 2
Sum of input numbers is 3
```

Exercise 4. Odd or even?

Exercise 4. Odd or Even?

Create a program that asks a number to the user and distinguishes if it is odd or even.

* Improve the previous program by asking the user two numbers instead of one and identify odd and even numbers in-between these two numbers.

Organize the previous code into different functions, creating the function ***verifyOddEven()* (to check if the number is odd or even) and a function ***readNumber()*** to read the integer numbers. If you are brave enough, confirm and switch, if needed, the two numbers introduced in case the first is greater than the second.

Exercise 5. While loop

Exercise 5. While loop

1) Create a while loop that increases and prints numbers from 0 to 10.
2) Create a while loop that decreases and prints numbers from 10 to 0.

1) Create do-while loop with increasing values
2) Create do-while loop with decreasing values

1) Create a while loop that asks two numbers and prints values with decreasing or increasing step depending on which one and if such is greater.

* Add while loop for looping the second part of the program (asking numbers) and continue/exit on hit y/n characters.

Exercise 6. Break and continue

Exercise 6. Break and continue

1) Create a program which calculates the sum of numbers (10 numbers max), if the user enters a negative number, the loop terminates.

1) Create a program which calculates the sum of numbers (10 numbers max), if the user enters a negative number, it's not added to the result.

Exercise 7. Guess the number.

Exercise 7. Guess the number.

Create a program that generates a random number between 1 and 100, and the player tries to guess that number in less than 10 attempts.

Each time the player enters a guess, the computer tells him whether the guess is too high, too low, or right. Once the player guesses the number, the game is over.

Exercise 8. Prime numbers.

Exercise 8. Prime numbers.

As before, create a program that asks the user for two numbers, though this time identify the ones that are prime.

Create a function to verify if numbers are prime called *isPrime()* which returns 0 if it is not, and 1 if it is. Afterwards, create a function *identifyPrimes()* that receives the range and calls the previous function to identify primes in-between.

Exercise 9. Student Information

Exercise 9. Student Information

Create a program that stores the information of a student in a structure and displays it on the screen.

```
Enter information:
Enter name: Julia
Enter roll number: 23
Enter average marks: 91

Displaying Information:
Name: Julia
Roll number: 23
Marks: 91.0
```

Exercise 10. Input and Output String

Exercise 10. Input and Output String

Create a program that asks a person name using *gets* function and prints it to the screen with *puts* function.

```
Enter your name: Daryna
Your name is: Daryna
```

* Add asking for surname and improve with structures

```
Enter your name: Micael
Enter your surname: Jackson

Enter your name: Lisa Marie
Enter your surname: Presley
Persons information is:
Micael
Jackson
Lisa Marie
Presley
```

Exercise 11. Who’s the oldest?

Exercise 11. Who's the oldest?

Create a program that receives the name and birthdate of two users (day, month and year), indicating which of the two is the oldest. Use the function *oldestFcn()* to assess who is the oldest.

* Go one step further by creating the structure Person, which shall include *Name* and *Date*, where in *Date* is yet another structure comprising *day*, *month* and *year*.

Exercise 12. Diamond.

Exercise 12. Diamond.

Create a program that prints the diamond pattern . The user defines number of rows.

To print the shape create recursive function and use char symbol of your preference (e.g. '*', '#' etc).

```
Enter number of rows:
4
   *
  ***
 *****
*******
 *****
  ***
   *
```

Exercise 13. Array input/output.

Exercise 13. Array input/output

Write a program that takes 5 values from the user and store them in an array.

Print the elements stored in the array

```
Enter 5 integers:
2
3
6
1
2
Displaying integers: 2 3 6 1 2
```

Exercise 14. Reverse an array.

Exercise 14. Reverse an array.

Write a program to reverse the elements of an integer 1-D array.

The number of elements and values of the array must be provided by user.

```
Enter the number of elements in array:
9
Enter array elements:
0 1 2 3 4 5 6 7 8
Reverse array is:
8 7 6 5 4 3 2 1 0
```

Exercise 15. Find array element.

Exercise 15. Find array element.

Write a program to search for a value and its index in one-dimensional array of integers.

Create a function to efficiently search for a data VAL. If VAL is present in the array then the function should return its index and o otherwise.

Exercise 16. Tic-Tac-Toe.

Exercise 16. Tic-Tac-Toe.

Build a program that allows to play the Tic-Tac-Toe game with two users. The program must show the game and allow each player to play in turns

Exercise 17. Pointers

Exercise 17. Pointers

1) Write a program to add two numbers using pointers.
2) Create *addTwoNumbers()* function which accepts two pointers

```
Pointer : Add two numbers :
-------------------------------
Input the first number : 2
Input the second  number : 3
The sum of the entered numbers is : 5
```

Exercise 18. Memory allocation

Exercise 18. Memory allocation

Implement a program that asks the user for a number of elements in pointers array, and finds the largest element in that array. Use Dynamic Memory Allocation for memory allocation (*calloc* or *malloc*)

```
//Memory allocation for n elements in array
ptr=(int*) calloc(n,sizeof(int));
ptr = (int*) malloc(n * sizeof(int));
```

Exercise 19. Size of string.

Exercise 19. Size of string.

The idea is to implement a program that asks the user for a string (array of chars) and counts the number of characters using pointers.

You can resort to *fgets* to read the string and identify the end of the string by detecting the character '\0'

Exercise 20. Time parsing.

Exercise 20. Time parsing.

Create a function that receives 3 parameters (inputs); one as value and two as reference (pointers)

a. Integer value which represents the total time in minutes (*timeT*)

b. Address of an integer variable to represent the hours (*hours*)

c. Address of an integer variable to represent the minutes (*minutes*)

For instance, *timeT* = 75 yields *hours* = 1 and *minutes* = 15

The function should pick the value of parameter *timeT* (time in minutes), converting it into *hours* and *minutes* into the variables as reference.

Exercise 21. Maths.

Exercise 21. Maths.

Create a function that receives 3 real numbers (inputs):

a. The radius of a sphere (*radS*)

b. Address of two variables (*areaS*, *volS*)

The function must use the radius to calculate the area and volume of the sfere.

The math.h library provides the value to π through *M_PI* and the power function with *pow*.

Exercise 22. Fun with Arrays.

Exercise 22. Fun with Arrays.

Implement a program that creates two arrays of random numbers, with size defined by the user, and, as a result, presents the elements unique to the first array and the elements in common between both.

The program should:

a. Ask the user for two integer numbers: *size1* and *size2*

b. Create two arrays of integers (*a1* and *a2*) with the previous sizes using the *malloc()* function

c. Fill in the arrays with random numbers between 0 and 10, or allow the user to introduce the numbers

d. Identify the values in *a1* that do not exist in *a2* and save them in the *diffArray* array

e. Identify the values in common between *a1* and *a2*, saving them in the *equalArray* array

f. Show the content of all the arrays: *a1*, *a2*, *diffArray* and *equalArray*

Exercise 22. Fun with Arrays.

Suggestion: Create your own library *Array.h* (and *Array.c*) with the following functions:

- *createArray*(int *sizeA*) – allocate memory for an array of size *sizeA*
- *insertValue*(int **a*, int *value*, int * *sizeA*) – add *value* to array *a* in *sizeA+1* position (dynamically increase the size of array *a*)
- *copyArray*(int **originArray*, int *originSize*, int **targetArray*, int *targetSize*) – Copy data from *originArray* to *targetArray*

Exercise 23. Class Rectangle.

Exercise 23. Class Rectangle.

1) Write a class *Rectangle* having two float private variables (height, width) and one member function which will return the *area* of the rectangle.
2) Create default constructor without parameters which sets both variables to 1.
3) Create method *Set* and *Get* for setting and getting variables values.
4) Improve a program so it can input two integers in main and pass them to default **constructor** of the class. Show the result of the area calculation.

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

class Rectangle {
  private:
    // Private attribute
    float width, height;

  public:
    //Constructor
    Rectangle(){
        width = 1;
        height = 1;
    }
    Rectangle(float w, float h){
        width = w;
        height = h;
    }
    // Setters
    void setWidth(float w) {
      width = w;
    }
    void setHeight(float h) {
      height = h;
    }
    // Getters
    float getWidth() {
      return width;
    }
    float getHeight() {
      return height;
    }
    float calcArea(){
        return width*height;
    }
};
```

```cpp
int main() {
  Rectangle r1, r2(1,2);
  cout << "Area is " << r1.calcArea() << endl;
  cout << "Area is " << r2.calcArea() << endl;

  cout << "\nModifying width and height of r1" << endl;
  r1.setHeight(5);
  r1.setWidth(5);
  cout << "Area of r1 is " << r1.calcArea() << endl;

  cout << "\nInput width and height of r2:" << endl;
  float h,w;
  cin>>h;
  cin>>w;
  r2.setHeight(h);
  r2.setWidth(w);
  cout << "Area of r2 is " << r2.calcArea() << endl;


  return 0;
}
```

```
Area is 1
Area is 2

Modifying width and height of r1
Area of r1 is 25

Input width and height of r2:
2
3
Area of r2 is 6
```

Exercise 24. Class Date.

Exercise 24. Class Date: Set and Get

Create the Date class with the following attributes and methods. Write class code in .h and .cpp files.

1) Implement two constructors for the Date class. A default constructor (without parameters) and a constructor with user-defined parameters)
2) Implement all Get() and Set() methods so that they are *inline* methods

Inline method has the normal features of methods (syntax, argument checking, etc), but are expanded in compile-time instead of invoked in run-time. The code for these methods must be in the specification file (.h), so changing them implies the recompilation. The inline declaration is only justified for simple methods.

Date.h

```
#pragma once


#include <iostream>
#include <fstream>

using namespace std;

class Date
{
private:
    int year, month, day;

public:
    Date();

    Date(int _day, int _month, int _year);
    virtual ~Date();

    void SetDay(int _day) { day = _day; }
    void SetMonth(int _month) { month = _month; }
    void SetYear(int _year) { year = _year; }
    int GetDay(void);
    int GetMonth(void) { return month; }
    int GetYear(void) { return year; }
};

```

Date.cpp

```
#include "Date.h"


Date::Date()
{
    day = month = year = 0;
}

Date::~Date()
{
}

int Date::GetDay(void)
{return day;}

Date::Date(int _day, int _month, int _year)
{
    day = _day;
    month = _month;
    year = _year;
}
```

main.cpp

```
#include "Date.h"

int main()
{
    Date da(1, 1, 2014), db;
    Date my_data(1, 1, 2000);

    return 0;
}
```

Exercise 24. Class Date: Show and Update

3) Implement the *Show()* method to write the object's attributes on the screen.
4) Implement the *Update()* method that allows updating all the attributes of a Date object

A program should:

- Create 2 objects of type Date using the default constructor for one and the parameter constructor for the other.
- Invoke the *Show()* method using each of the previously created objects.
- Set the attributes of the object using default constructor with data entered by the user via the keyboard.
- Change the attributes of the object created with the parameter constructor using the Update() method.

Exercise 24. Class Date.

Date.cpp

```cpp
void Date::Show(void)
{
cout << "Date:" << day << "/" << GetMonth() << "/" << GetYear() << "\n";
}

void Date::Update(int _day, int _month, int _year)
{
    SetDay(_day);// day = _day;
    SetMonth(_month);// month = _month;
    SetYear(_year);//   year = _year;
}
```

Date.h

```cpp
void Show(void);
void Update(int _day, int _month, int _year);
```

main.cpp

```cpp
#include "Date.h"

int main()
{
    Date da(1, 1, 2014), db;
    Date my_data(1, 1, 2000);

    da.Show();
    db.Show();

    db.Update(2, 2, 2014);
    db.Show();

    return 0;
}
```

Output

```
Date:1/1/2014
Date:0/0/0
Date:2/2/2014
```

Exercise 24. Class Date: Check for equality and overload operators

5) Implement the *Same()* method for the Date class to check if two objects of type Date are the same (equal). The method must return **true** if they are equal and **false** otherwise.
6) Check whether two objects of type Date are equal by overloading the "==" operator
7) Implement overloading the "<" and ">" operators

Date.cpp

```cpp
bool Date::Same (const Date date)
{
    if (date.year != year || date.month != month || date.day != day)
        return(false);
    return(true);
}

bool Date::operator == ( Date data) const
{
    if (data.year != year || data.month != month || data.day != day)
        return(false);
    return(true);
}

bool Date::operator < (const Date data) const
{
    if (data.year < year)
        return(false);
    else    if (data.year == year && data.month < month)
        return(false);
    else    if (data.year == year && data.month == month && data.day < day)
        return(false);
    return(true);
}

bool Date::operator >(const Date data) const
{
    if (data.year > year)
        return(false);
    else    if (data.year == year && data.month > month)
        return(false);
    else    if (data.year == year && data.month == month && data.day > day)
        return(false);
    return(true);
}
```

Date.h

```cpp
bool Same(const Date date);
bool operator == ( Date date) const;
bool operator < (const Date date) const;
bool operator > (const Date date) const;
```

main.cpp

```cpp
#include "Date.h"

int main()
{
    Date da(1, 1, 2014), db;
    Date my_data(1, 1, 2000);

    da.Show();
    db.Show();

    db.Update(2, 2, 2014);
    db.Show();

    if (da.Same(db))
        cout << "Same Dates" << endl;
    else
        cout << "Different Dates" << endl;

    if (da == db)
        cout << "Same Dates" << endl;
    else
        cout << "Different Dates" << endl;

    my_data.Update(1, 1, 2014);

    if (da == my_data)
        cout << "Same Dates" << endl;
    else
        cout << "Different Dates" << endl;

    return 0;
}
```

Output

```
Date:1/1/2014
Date:0/0/0
Date:2/2/2014
Different Dates
Different Dates
Same Dates
```

Exercise 24. Class Date: Output to screen and file

8) Create a new Date object and read its data via the keyboard using the operator created in the previous steps.
9) Show all created Date objects on the screen using the "<<" operator.
10) Implement in the Date class methods for reading (*ReadFile*) and writing (*SaveFile*) to file. Then invoke these methods in your program.

Date.cpp

```cpp
ostream & operator << (ostream &os,  Date date)
{
    os << "Date:" << date.GetDay() << "/" << date.month << "/" << date.year << "\n";

    return os;
}

istream & operator >> (istream &is, Date &date)
{   int aux;

    cout << "day: ";
    is >> aux;
    date.SetDay(aux);
    cout << "month: ";
    is >> date.month;
    cout << "year: ";
    is >> date.year;

    return is;
}

void Date::SaveFile(ofstream& os)
{
    os << "Date: "<< GetDay() << "/" << GetMonth() << "/" << GetYear() << "\n";
}

void Date::ReadFile(ifstream& is)
{   char aux[10];

    is.getline(aux,10,' ');

    is.getline(aux, 10, '/');
    day = atoi(aux);
    is.getline(aux, 10, '/');
    month = atoi(aux);
    is.getline(aux, 10, ';');
    year = atoi(aux);
}
```

Date.h

```cpp
friend ostream & operator << (ostream &os, Date date);
friend istream & operator >> (istream &is, Date &date);

void SaveFile(ofstream& os);
void ReadFile(ifstream& is);
```

main.cpp

```cpp
Date dc;
cin >> dc;

cout << da << endl << db << endl << dc << endl;

my_data.Update(12, 1, 2015);
ofstream os;
os.open("datas.txt");
if (!os)
{
    cout << "ERROR: not possible to open datas.txt" << '\n';
    exit(1);
}

my_data.SaveFile(os);

os.close();

ifstream is;

is.open("datas.txt");
if (!is) {
    cout << "ERROR: not possible to open datas.txt" << '\n';
    exit(1);
}
my_data.ReadFile(is);

cout << my_data;
cin.get();
os.close();

return 0;
```

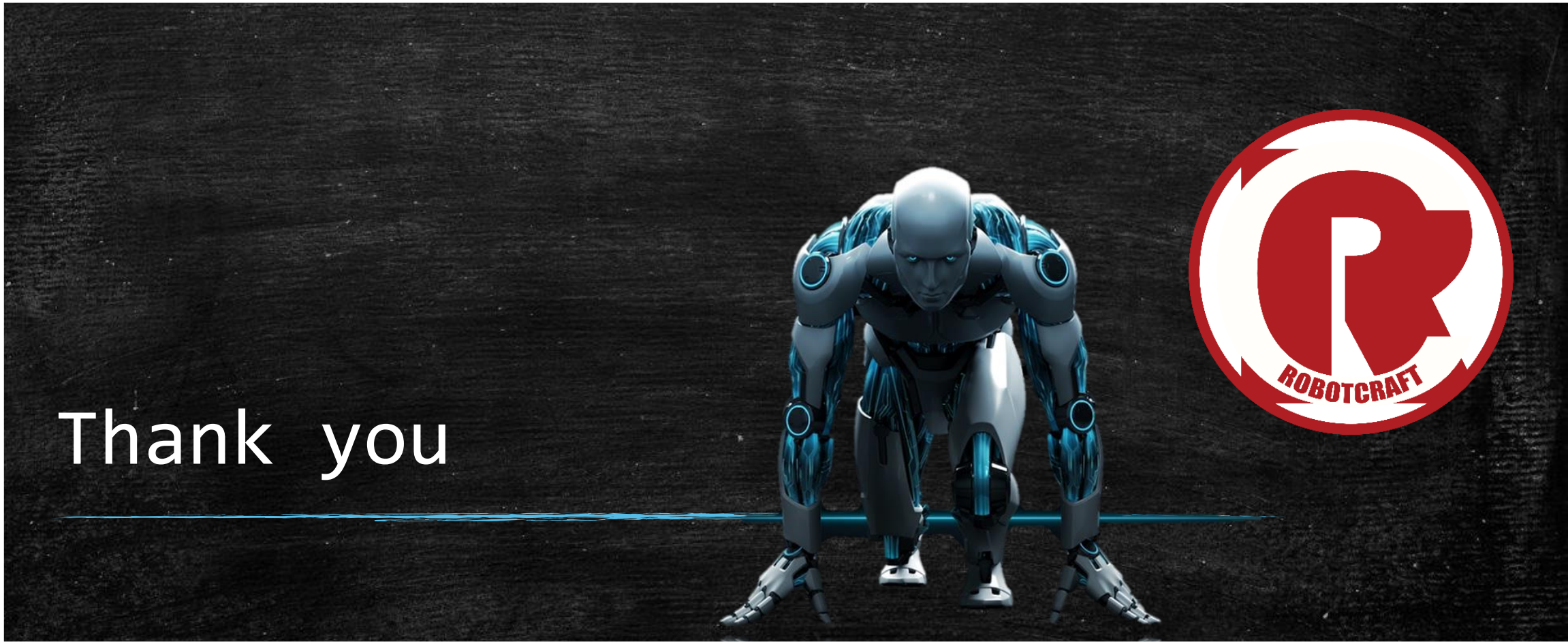
Output

```
Date:1/1/2014
Date:0/0/0
Date:2/2/2014
Different Dates
Different Dates
day: 3
month: 4
year: 2016
Date:1/1/2014

Date:2/2/2014

Date:3/4/2016

Date:12/1/2015
```

ROBOTCRAFT©

ROSin

INGENIARIUS



Micael S. Couceiro micael@ingeniarius.pt

Panagiotis Karfakis panagiotis@ingeniarius.pt

Craft #2

02/07/2024