

## Laboratory for *in silico* life construction

My laboratory for *in silico* life construction is digital. Its infrastructure consists of an Integrated Design/Development Environment (IDE) needed for designing/developing programs for *in silico* life construction experiments. Once designed/developed, the program then conducts the experiment every time it runs.

I use C++ as programming language and, respectively, Microsoft Visual C++ 2008 Express Edition as IDE. Alternatively, I use Dev-C++ too. C++ is a general-purpose programming language supporting multiple programming styles. It is rather small and simple. Additionally, it comes with a large library of useful components that may be easily included in program to support it.

In the IDE, the program design/development begins with writing source code in C++ using an editor. The source code is typically saved in a file with the extension .cpp. Another part of the IDE – a compiler – is then used to translate source code into machine code saving it in a file with the extension .obj. Finally, the linker takes one or more files generated by compiler and combines them into a single executable machine code saving it in a file with the extension .exe. The file with the executable machine code is the ready-made program that instructs the computer what is to do every time it runs. An IDE can include additional tools dedicated to maximize programming productivity.

Once the IDE is installed, it is reasonable first to warm up with exercises introducing into C++ fundamentals and then proceed to designing/developing of C++ programs for *in silico* life construction experiments.

## Objectives

<b>Exercise 1:</b>	Learn how to design/develop C++ program using IDE
<b>Exercise 2:</b>	Learn how to write C++ statements instructing the computer to construct objects in the memory and to operate with them
<b>Exercise 3:</b>	Learn how to write C++ code for control structures
<b>Exercise 4:</b>	Learn how to write C++ code for functions

<b>Experiment 1:</b>	Learn how to design/develop C++ program for an <i>in silico</i> life construction experiment
<b>Experiment 2:</b>	Learn how to construct <i>in silico</i> gene expression networks (GENs)
<b>Experiment 3:</b>	Learn how to construct <i>in silico</i> genome expression networks (GENomes)
<b>Experiment 4:</b>	Learn how to construct <i>in silico</i> genome multiplication networks

## Exercise 1

**Objective:** Learn how to design/develop C++ program using IDE.

Start IDE (here, Visual C++ 2008). A collection of IDE windows appears. To the left, there is the window with three view tabs. Select **Solution Explorer**. To the right, there is the editor window showing **Start Page**. At the bottom, there is the window with three view tabs. Select **Output**.

Go to the menu **File** and select **New -> Project**. A dialog box **New Project** will pop up. To the left, there is the pane **Project Types**. Select **Win32**. To the right, there is the pane **Templates**. Select **Win32 Console Application**. At the bottom, there are three fields: **Name**, **Location**, and **Solution Name**. Enter a project name (here, Exercise1-1) in the field **Name**. The solution name (here, Exercise1-1) appears automatically in the field **Solution Name** and is the same as the project name. If necessary, change the path for solution location in the field **Location** or select the path by using the button **Browse**. Click the button **OK**. A **Win32 Application Wizard** will pop up showing the page **Overview** with the settings currently in effect. Click the button **Next**. The page **Application Settings** appears. Activate the checkbox **Empty Project** and click the button **Finish**. In the window **Solution Explorer**, the solution folder appears. It includes the project folder with three subdirectories: **Header Files**, **Source Files**, and **Resources Files**.

Select the subdirectory **Source Files** and click the right button of the mouse. A context menu appears. Select **Add -> New Item**. A dialog box **Add New Item** will pop up. To the left, there is the pane **Categories**. Select **Code**. To the right, there is the pane **Templates**. Select **C++ File (.cpp)**. At the bottom, there are two fields: **Name** and **Location**. Enter a file name (here, Exercise1-1) in the field **Name** and click the button **Add**. In the window **Solution Explorer**, a new file (here, Exercise1-1.cpp) will be added to the project folder in the subdirectory **Source Files**. In the editor window, an empty file with the same name appears.

Enter the following code within the file Exercise1-1.cpp:

```

1 //Exercise 1-1
2 /*Nikita Tirjatkin
3   Laboratory for in silico life construction
4   Jabuar 2010*/
5
6 #include <string>
7 #include <iostream>
8 using namespace std;
9
10 int main()
11 {
12     string Welcome("Welcome to the laboratory for in silico life construction!");
13     cout << Welcome << endl << endl;
14     return 0;
15 }
```

Note that the numbers to the left of the vertical punctured line represent the line numbers and do not belong to the source code. The source code is to the right of the punctured line. The line numbers must appear automatically by the entering the source code. If this is not the case, go to the menu **Tools** and select **Options**. The dialog box

**Options** will popup. In the left pane, select **Text Editor -> C/C++ -> General**. Activate the checkbox **Line Numbers** and click the button **OK**.

Although extremely simple, the source code in the file Exercise1-1.cpp already contains all components that every C++ source code usually has.

Lines from 1 to 4 contain comments. Comments have no effect on the behaviour of the program and are usually used to include explanations. Here, comments provide information to the program. One-line comments begin with two slash signs `//`. Long comments are typically included between `/*` and `*/`.

Lines 6 and 7 contain two directives for the preprocessor of the compiler. Each directive begins with hash sign `#`. Here, directives tell the preprocessor to include two files `string` and `iostream` from the Standard C++ Library. Their functionality is going to be used later in the program. The filenames are surrounded by signs `<` and `>`. Line 8 contains expression that is very frequent for the source codes that use the Standard C++ Library because all its components are usually declared within what is called a `namespace`. Here, the `namespace` has the name `std`.

Lines from 10 to 15 contain the definition of the `main`-function. The `main`-function is mandatory. Every C++ program must have the `main`-function. Even if the C++ program contains other functions, the program execution begins by the `main`-function, independently of its location within the source code. The name `main` of the `main`-function is usually followed by a pair of parentheses `()` and then by the body of the function enclosed in braces `{}`. Here, the body of the `main`-function begins with the statement instructing the computer to construct in the memory an object of the type `string` named `Welcome` and to set up this object as a sequence of characters `Welcome to the laboratory for in silico life construction!`. The next statement instructs the computer to construct the standard output stream `cout` and to use it to transport the image of the object `Welcome` and two end line signs to the console window of the screen. Finally, the statement

```
return 0;
```

is the most usual instruction to end C++ program for console window.

Additionally, the source code also contains whitespace – spaces, tabs, new lines, blank lines. Like comments, they have no effect on the behaviour of the program but serve for the better readability of the code.

After entering the source code to the file Exercise1-1.cpp, go to the menu **Build** and select **Build Exercise1-1**. In the window **Output**, the protocol with various status messages will appear. If all goes well, the protocol ends with the message:

```
Exercisel - 0 error(s), 0 warning(s)
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
```

To run the executable machine code, go to the menu **Debug** and select **Start Without Debugging**. The console window will appear and show the image of the object `Welcome`:

```
Welcome to the laboratory for in silico life construction!  
Press any key . . .
```

After program execution, the console window usually pauses automatically, exhibits the stream `Press any key . . .`, and waits for user action. However, in some IDEs (for example, Dev-C++), the console window closes immediately after program execution so that the short blink is only seen on the screen. In this case, it is necessary to extend the source code by three additional instructions

```
cin.ignore(255, '\\n');  
cout << "Press any key . . .";  
cin.get();
```

before the return instruction.

## Exercise 2

**Objective:** Learn how to write C++ statements instructing the computer to construct objects in the memory and to operate with them.

Create a new solution/project Exercise2-1 and add a new file Exercise2-1.cpp. Enter the following code within the file Exercise2-1.cpp:

```

1 //Exercise 2-1
2 /*Nikita Tirjatkin
3   Laboratory for in silico life construction
4   Januar 2010*/
5
6 #include <iostream>
7 using namespace std;
8
9 int main()
10 {
11     char Letter1;
12     Letter1 = 'a';
13
14     char Letter2 = 'b';
15     char Letter3('c');
16
17     cout << Letter1 << "\t" << Letter2 << "\t" << Letter3 << "\n\n";
18
19     return 0;
20 }
```

Line 11 contains a declaration statement instructing the computer to construct an object in the memory. The smallest unit of memory is a binary digit (bit), which can hold a value of 0 or 1. The memory in computer is organized into individual sections called addresses. The smallest addressable unit of memory is a group of 8 bits known as a byte. One byte is enough to construct a relatively tiny object such as a single character or small integer (between 0 and 255). To construct more complex objects, the computer needs to group several bytes in any way. In C++, there are few built-in object types. C++ also provides users with features to design their own object types. These features were intensively used by C++ community to invent a large number of useful object types. Many of them are included in the Standard C++ Library. Thus, a declaration must contain at least two identifiers/names. The first identifier/name is for the object type while the second identifier/name is for the object itself. Here, the computer is instructed to construct an object of the type `char` with the name `Letter1` in the memory. Once constructed, the object can be used within the rest of its scope in the program. This scope is limited to the block enclosed in braces `{}`, where the object has been declared (here, to the body of the `main`-function).

Line 12 contains an assignment statement instructing the computer to assign a value to the object `Letter1`. To assign a value to the object, the assignment operator `=` is used. An object of the type `char` can hold either a small number or a character from the ASCII set of characters. ASCII defines a mapping between the keys on the American keyboard and numbers from 1 to 127. For example, the character for the letter `a` is mapped to the number 97. Here, the computer is instructed to assign a character `a` to the object `Letter1`. Characters to be assigned are always placed between single quotes `' '`.

Lines 14 and 15 contain statements combining declaration with assignment. In line 14, the assignment occurs explicitly by using assignment operator =. In line 15, the assignment occurs implicitly by using constructor function of the object.

Line 17 contains a statement instructing the computer to construct the standard output stream `cout` and to use it to transport the image of the objects `Letter1`, `Letter2`, and `Letter3` to the console window of the screen. Additionally, some characters with special meaning must be transported to the console window. These characters are called escape sequences and serve for formatting the standard output stream. Here, the tabulator `\t` and the end line `\n` are used. Operator `<<` is used to send images of all the objects to the standard output stream `cout`.

Build Exercise2-1 and run the executable machine code. The console window will show images of the objects `Letter1`, `Letter2`, and `Letter3`:

```
a      b      c
Press any key . . .
```

Create a new solution/project Exercise2-2 and add a new file Exercise2-2.cpp. Enter the following code within the file Exercise2-2.cpp:

```
1 //Exercise 2-2
2 /*Nikita Tirjatkin
3   Laboratory for in silico life construction
4   Januar 2010*/
5
6 #include <iostream>
7 using namespace std;
8
9 int main()
10 {
11     int Number1 = 5, Number2 = 10, Number3 = 20;
12     float Number4 = 5.2, Number5 = 10.4, Number6 = 20.6;
13
14     cout << Number1 << "\t" << Number2 << "\t" << Number3 << "\n"
15          << Number4 << "\t" << Number5 << "\t" << Number6 << "\n\n"
16          << Number1 + Number2 << "\t" << Number4 + Number5 << "\n"
17          << Number3 - Number2 << "\t" << Number6 - Number5 << "\n"
18          << Number1 * Number2 << "\t" << Number4 * Number5 << "\n"
19          << Number3 / Number2 << "\t" << Number6 / Number5 << "\n"
20          << Number3 % 7 << "\n\n";
21
22     Number1 += 3; //is equivalent to Number1 = Number1 + 3
23     Number2 -= 3; //is equivalent to Number2 = Number2 - 3
24     Number3 %= 3; //is equivalent to Number3 = Number3 % 3
25     Number4 *= 3; //is equivalent to Number4 = Number4 * 3
26     Number5 /= 3; //is equivalent to Number5 = Number5 / 3
27     cout << Number1 << "\t" << Number2 << "\t" << Number3 << "\n"
28          << Number4 << "\t" << Number5 << "\t" << Number6 << "\n\n";
29
30     int i = 1, j = 10;
31     i++; //is equivalent to i = i + 1 and to i += 1
32     j--; //is equivalent to j = j - 1 and to j -= 1
33     cout << i << "\t" << j << "\n\n";
34
35     return 0;
36 }
```

Lines 11 and 12 contain statements instructing the computer to construct in the memory three objects of the type `int` and three objects of the type `float` respectively. Objects of the type `int` can only hold whole numbers. To hold floating point numbers, objects of the type `float` must be used.

Lines from 14 to 20 contain a statement instructing the computer to construct the standard output stream `cout` and to use it to transport the image of the objects `Number1`, `Number2`, `Number3`, `Number4`, `Number5`, and `Number6` to the console window of the screen. Additionally, the computer is instructed to use the standard output stream to transport some nameless objects to the console window. These nameless objects only emerge to hold numbers resulting from arithmetic operations on the named objects. Operators `+`, `-`, `*`, and `/` correspond to addition, subtraction, multiplication, and division respectively. The modulus operator `%` gives the remainder after doing division and is very useful for testing whether a number is evenly divisible by another number. However, it only works on integer numbers.

Lines from 22 to 26 show some examples for statements using compound assignment operators `+=`, `-=`, `*=`, `/=`, and `%=`. These operators serve to shorten expressions shown in comments. Lines 31 and 32 contain statements using operators `++` and `--` for even more shortening of expressions.

Build `Exercise2-2` and run the executable machine code. The console window will show images of all the objects transported by the standard output stream:

```

5      10      20
5.2    10.4    20.6

15     15.6
10     10.2
50     54.08
2      1.98077
6

8      7      2
15.6   3.46667 20.6

2      9

Press any key . . .

```

## Exercise 3

**Objective:** Learn how to write C++ code for control structures.

Create a new solution/project Exercise3-1 and add a new file Exercise3-1.cpp. Enter the following code within the file Exercise3-1.cpp:

```

1 //Exercise 3-1
2 /*Nikita Tirjatkin
3   Laboratory for in silico life construction
4   Januar 2010*/
5
6 #include <iostream>
7 using namespace std;
8
9 int main()
10 {
11     char Letter1 = 'a', Letter2 = 'b', Letter3 = 'c';
12     int Number;
13
14     cout << "Enter number: ";
15     cin >> Number;
16
17     if(Number == 10)
18         cout << Letter1 << "\n\n";
19     else if(Number < 10)
20         cout << Letter2 << "\n\n";
21     else
22         cout << Letter3 << "\n\n";
23
24     return 0;
25 }
```

Lines 14 and 15 introduce some interactivity to the program. The standard output stream `cout` must transport to the console windows a message that prompts the user to enter any number. Line 15 contains a statement instructing computer to construct the standard input stream `cin` and to use it to transport the number from the keyboard to the object `Number` in the memory of the computer. Operator `>>` is used to send the number from the standard output stream `cin` to the object `Number`.

Lines from 17 to 22 contain an example for a conditional control structure. The table summarizes three different forms of this control structure:

<code>if(condition)</code> statement	The program evaluates condition. If condition is true, statement will be executed. If condition is false, statement will be skipped and the program will proceed to the statement after the control structure.
<code>if(condition)</code> statement1 <code>else</code> statement2	The program evaluates condition. If condition is true, statement1 will be executed, statement2 will be skipped, and the program proceeds to the statement after the control structure. If condition is false, statement1 will be skipped, statement2 will be executed, and the program will proceed to the statement after the control structure.
<code>if(condition1)</code> statement1 <code>else if(condition2)</code> statement2 <code>else if(condition3)</code> statement3 ... <code>else</code> statement	The program evaluates conditions to select the statement to be executed.

In conditions, the equality operator `==` and relational operators `!=`, `>`, `<`, `>=`, `<=` are usually used. Here, the choice is made dependent from the value of the number to be entered by the user.

Build Exercise3-1 and run the executable machine code. The console window will first show the following:

```
Enter number:
```

Depending on the number entered, three variants are possible. For example:

```
Enter number: 10
a
Press any key . . .
```

```
Enter number: 1
b
Press any key . . .
```

```
Enter number: 100
c
Press any key . . .
```

Create a new solution/project Exercise3-2 and add a new file Exercise3-2.cpp. Enter the following code within the file Exercise3-2.cpp:

```
1 //Exercise 3-2
2 /*Nikita Tirjatkin
3   Laboratory for in silico life construction
4   Januar 2010*/
5
6 #include <iostream>
7 using namespace std;
8
9 int main()
10 {
11     int i;
12     cout << "Enter number to start countdown: ";
13     cin >> i;
14
15     while(i > 0)
16     {
17         cout << i << "! ";
18         i--;
19     }
20     cout << "ZERO-O-O!!!\n\n";
21
22     return 0;
23 }
```

Lines from 15 to 19 contain an example for an iterative control structure. The table summarizes two different forms of this control structure:

<code>while(condition)</code> statement	The program evaluates <code>condition</code> . If <code>condition</code> is false, <code>statement</code> will be skipped and the program will proceed to the statement after the control structure. If <code>condition</code> is true, <code>statement</code> will be executed and the program will loop back to evaluate <code>condition</code> again. This cycle will be repeated a certain number of times until <code>condition</code> will become false.
<code>do</code> statement <code>while(condition)</code>	The program first executes <code>statement</code> and then evaluates <code>condition</code> . If <code>condition</code> is false, the program will proceed to the statement after the control structure. If <code>condition</code> is true, the program will loop back to execute <code>statement</code> and evaluate <code>condition</code> again. This cycle will be repeated a certain number of times until <code>condition</code> will become false.

Here, the choice is made dependent from the value of the number to be entered by the user. Note that the statement is actually a block of two statements enclosed in braces `{}`.

Build Exercise3-2 and run the executable machine code. The console window will first show the following:

```
Enter number to start countdown:
```

After entering the number, the console window will show the complete countdown:

```
Enter number to start countdown: 10
10! 9! 8! 7! 6! 5! 4! 3! 2! 1! ZERO-O-O!!!

Press any key . . .
```

Create a new solution/project Exercise3-3 and add a new file Exercise3-3.cpp. Enter the following code within the file Exercise3-3.cpp:

```
1 //Exercise 3-3
2 /*Nikita Tirjatkin
3 Laboratory for in silico life construction
4 Januar 2010*/
5
6 #include <iostream>
7 using namespace std;
8
9 int main()
10 {
11     int i;
12     cout << "Enter number to start countdown: ";
13     cin >> i;
14
15     for(int j = i; j > 0; j--)
16         cout << j << "! ";
17     cout << "ZERO-O-O!!!\n\n";
18
19     return 0;
20 }
```

Lines from 15 to 17 contain another example for an iterative control structure. Its form is:

```
for(statement1; condition; statement3)
    statement2
```

The program first executes `statement1`. Generally, `statement1` is an initialization statement declaring a counter variable and assigning an initial value to it. Then, the program evaluates `condition`. If `condition` is false, `statement2` will be skipped and the program will proceed to the statement after the control structure. If `condition` is true, the program will execute `statement2` and `statement3`. Generally, `statement3` is an increase/decrease statement changing a value of a counter variable in any regular way. Next, the program will loop back to evaluate `condition` again. This cycle will be repeated a certain number of times until `condition` will become false.

Build Exercise3-3 and run the executable machine code. The console window will first show the following:

```
Enter number to start countdown:
```

After entering the number, the console window will show the complete countdown:

```
Enter number to start countdown: 10
10! 9! 8! 7! 6! 5! 4! 3! 2! 1! ZERO-O-O!!!

Press any key . . .
```

Note, the program functionality is the same as in the Exercise 3-2.

## Exercise 4

**Objective:** Learn how to write C++ code for functions.

Create a new solution/project Exercise4-1 and add a new file Exercise4-1.cpp. Enter the following code within the file Exercise4-1.cpp:

```

1 //Exercise 4-1
2 /*Nikita Tirjatkin
3   Laboratory for in silico life construction
4   Januar 2010*/
5
6 #include <iostream>
7 using namespace std;
8
9 float addNumber(float a, float b)
10 {
11     float c;
12     c = a + b;
13     return (c);
14 }
15
16 int main()
17 {
18     float x, y, z;
19     x = addNumber(12.34, 23.45);
20     y = addNumber(34.56, 45.67);
21     z = addNumber(x, y);
22     cout << x << "\t" << y << "\t" << z << "\n\n";
23
24     return 0;
25 }

```

Lines from 9 to 14 contain an example for a function. Generally, a function is a named group of statements. Its form is:

```
returntype functionname(parameter1, parameter2,...){statements}
```

Once defined, it can be called many times. To call a function, the following form must be used:

```
functionname(argument1, argument2,...)
```

Note, the parameters and arguments must have a clear correspondence. Here, the function `addNumber` is designed for addition. If called, it will take two float point numbers and return their sum as a float point number.

Lines 19, 20, and 21 show examples how the function `addNumber` is called from within the `main`-function.

Build Exercise4-1 and run the executable machine code. The console window will show the following:

```

35.79    80.23    116.02
Press any key . . .

```

Create a new solution/project Exercise4-2 and add a new file Exercise4-2.cpp. Enter the following code within the file Exercise4-2.cpp:

```

1 //Exercise 4-2
2 /*Nikita Tirjatkin
3    Laboratory for in silico life construction
4    Januar 2010*/
5
6 #include <iostream>
7 using namespace std;
8
9 void countDown(int a)
10 {
11     cout << a << "! ";
12     if(a > 1)
13         countDown(a - 1);
14 }
15
16 int main()
17 {
18     int i;
19     cout << "Enter number to start countdown: ";
20     cin >> i;
21
22     countDown(i);
23     cout << "ZERO-O-O!!!\n\n";
24
25     return 0;
26 }

```

Lines from 9 to 14 contain an example for a recursive function. A recursive function is a function that calls itself. Here, the function `countDown` is designed for countdown. Note, it does not return any value. In this case, a special type name `void` is used.

Build Exercise4-2 and run the executable machine code. The console window will first show the following:

```
Enter number to start countdown:
```

After entering the number, the console window will show the complete countdown:

```
Enter number to start countdown: 10
10! 9! 8! 7! 6! 5! 4! 3! 2! 1! ZERO-O-O!!!

Press any key . . .
```

## Experiment 1

**Objective:** Learn how to design/develop C++ program for an *in silico* life construction experiment.

Create a new solution/project Experiment1-1 and add a new file Experiment1-1.cpp. Enter the following code within the file Experiment1-1.cpp:

```

1 //Experiment 1-1
2 /*Nikita Tirjatkin
3   Laboratory for in silico life construction
4   Januar 2010*/
5
6 #include <string>
7 #include <iostream>
8 #include <ctime>
9 using namespace std;
10
11 void waitSeconds(int Seconds)
12 {
13     clock_t Stop;
14     Stop = clock() + Seconds * CLOCKS_PER_SEC;
15     while(clock() < Stop){}
16 }
17
18 int main()
19 {
20     char Monomers[10] = {'A', 'B', 'C', 'B', 'C', 'A', 'C', 'B', 'A', 'C'};
21     cout << "Picture:\n\n";
22     for(int i = 0; i < 10; i++)
23         cout << Monomers[i] << " ";
24     cout << "\n\n";
25
26     string Polymer;
27     cout << "Movie:\n\n";
28     for(int i = 0; i < 10; i++)
29     {
30         waitSeconds(1);
31         Polymer.push_back(Monomers[i]);
32         cout << Polymer[i];
33     }
34     cout << "\n\n";
35
36     return 0;
37 }

```

Generally, a program for an *in silico* life construction experiment has to

1. set up an *in silico* experiment,
2. set up an *in silico* instrumentation,
3. run the *in silico* experiment,
4. display/record the outputs of the *in silico* instrumentation.

To set up an *in silico* experiment for *in silico* life construction, it is necessary first to choose object types for *in silico* chemicals. In living world, all chemicals can be roughly divided in two categories: monomers and polymers. The most familiar form of a polymer is a covalently bounded chain of monomers. For example, the DNA molecule strand is a chain of deoxyribonucleotides, the RNA molecule is a chain of ribonucleotides, the polypeptide is a chain of amino acids. For simplicity, objects of the type `char` can be used as *in silico* monomers. Respectively, objects of the type `string`

can be used as *in silico* polymers. Remember, `char` is a built-in object type while `string` is an object type defined in the file `string` of the Standard C++ Library. In order to be able to instruct computer to construct objects of the type `string` and operate with them, C++ program must contain a directive shown in line 6. The object type `string` is specifically designed for construction of objects that can hold a sequence of characters and for operation on them.

Here, C++ program has to set up an experiment for *in silico* polymerization. First, it instructs the computer to construct in the memory a pool of 10 *in silico* monomers. This pool is an array – a named memory location designed to store a series of objects of the same type. Objects are assigned to the array using braces `{}`. They thus share the same name and can be individually referenced by the index from 0 to  $N - 1$ , where  $N$  is the number of objects in the array. Note, the name of an array (here, `Monomers`) is followed by the number embraced within brackets `[]`. Depending on context, this number can mean either the number of objects in the array to be constructed or the index of the object to be referenced. Then, the program instructs the computer to construct in the memory an *in silico* polymer. Initially, it is a place where the polymerization will begin. Subsequently, the computer is instructed to add *in silico* monomers to the *in silico* polymer in a consecutive order using function `push_back`. Note how this function is called. Its name is connected to the name of the object `Polymer` by a dot sign `..`

In C++ program for console application, the standard output stream `cout` is the best device to be used as *in silico* instrumentation. It is an object of the object type `ostream` defined in the file `iostream` of the Standard C++ Library. Remember, standard output streams are suited very well to be placed in the memory of computer, snapshot objects there and transport their images to the console window. They are actually *in silico* nanoscopes with camera. Here, standard output streams are used to deliver pictures of the *in silico* polymerization. The function `waitSeconds` defined in lines from 11 to 15 serves to combine some pictures into a movie showing the *in silico* polymerization in slow motion. The function `waitSeconds` itself uses some objects and functions defined in the file `ctime` of the Standard C++ Library.

Build Experiment1-1 and run the executable machine code.

The console window will display the picture and the movie documenting how the experiment for *in silico* polymerization proceeds:

```
Picture:
A B C B C A C B A C
Movie:
ABCBCACBAC
Press any key . . .
```

## Experiment 2

**Objective:** Learn how to construct *in silico* gene expression networks (GENs).

Create a new solution/project Experiment2-1 and add a new file Experiment2-1.cpp. Enter the following code within the file Experiment2-1.cpp:

```

1 //Experiment2-1
2 /*Nikita Tirjatkin
3   Laboratory for in silico life construction
4   Januar 2010*/
5
6 #include <string>
7 #include <iostream>
8 using namespace std;
9
10 class gena
11 {
12 public:
13   string RNA;
14   gena(string Gene)
15   {
16     for(int i = 0; i < Gene.size(); i++)
17     {
18       if(Gene[i] == 'A') RNA.push_back('U');
19       else if(Gene[i] == 'C') RNA.push_back('G');
20       else if(Gene[i] == 'G') RNA.push_back('C');
21       else if(Gene[i] == 'T') RNA.push_back('A');
22       else
23         RNA.push_back('-');
24     }
25   };
26
27 int main()
28 {
29   string s0a, s0b;
30   s0a="CGTACGCTGCTTAAGCCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC";
31   s0b="GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG";
32   cout << "Picture 1:\n\n"
33     << s0a << endl
34     << s0b << "\n\n";
35   cout << "Picture 2:\n\n"
36     << s0a << "\n\n\n"
37     << s0b << "\n\n";
38
39   gena g1(s0a.substr(0, 9));
40   cout << "Picture 3:\n\n"
41     << s0a << endl
42     << g1.RNA << "\n\n"
43     << s0b << "\n\n";
44   cout << "Picture 4:\n\n"
45     << s0a << endl
46     << s0b << "\n\n"
47     << g1.RNA << "\n\n";
48
49   return 0;
50 }

```

Living world is an extremely complex network composed of huge numbers of different chemical reactions continuously creating an enormously complex matrix composed of bewildering numbers of different chemicals involved in these reactions. Additionally, awfully large numbers of interactions between non-living and living worlds contribute to life complexity. However, the living world becomes at once comprehensible as soon as one becomes familiar with how such basic chemical reactions as DNA transcription,

RNA translation, and catalysis arrange in strong hierarchy of life patterns shown in the table:

Level	Life pattern...	... is roughly equal to:
4	Genome diversification network	General cell progression (living world or biosphere)
3	Genome multiplication network	Individual cell progression
2	Genome expression network (GENome)	Cell
1	Gene expression network (GEN)	

Here, C++ program has to set up an experiment for construction of an *in silico* gene expression network (abbreviated GEN). In order to be able to instruct computer to construct such objects as *in silico* GENs, some new object types must be defined. The table summarizes two different forms for definition of new object types in C++:

<pre>class typename {     member object1;     member object2;     ...     member function1;     member function2;     ... };</pre>	By default, member objects and member functions have private access.
<pre>struct typename {     member object1;     member object2;     ...     member function1;     member function2;     ... };</pre>	By default, member objects and member functions have public access.

Lines from 10 to 26 contain code defining new object type `gena` for *in silico* GENs restricted to DNA transcription.

During DNA transcription, one strand of DNA molecule separates from another and exposes a particular sequence of deoxyribonucleotides – gene – serving as a template by guiding the synthesis of RNA molecule. This is possible because a ribonucleotide is allowed to be attached to the deoxyribonucleotide according to strict rules of base pairing:

Deoxyribonukleotide with the base...	Adenin	Cytosin	Guanin	Thymin
... attaches ribonukleotide with the base...	Uracil	Guanin	Cytosin	Adenin

So, the sequence of deoxyribonucleotides within a gene determines the sequence of ribonucleotides within RNA molecule to be synthesized. Accordingly, a complementary RNA replica appears on the DNA template and then separates from it. Note that the DNA molecule persists DNA transcription and remains unexhausted.

Here, the new object type `gena` contains one member object `RNA` of the object type `string` and one member function `gena`. This member function is a constructor function which will be automatically called if an object of the type `gena` will be constructed in the memory of the computer. The constructor function must have the same name as the new object type and cannot have any return type. Here, the constructor function has one

parameter `Gene` of the type `string`. The body of the constructor function contains code for control structure synthesizing RNA replica (here, `RNA`) on the DNA templates (here, `Gene`). Characters for *in silico* deoxyribonucleotides (**A**, **C**, **G**, and **T**) and ribonucleotides (**A**, **C**, **G**, and **U**) are chosen according to their conventional designation.

Lines from 29 to 37 contain code instructing the computer to construct in the memory an *in silico* DNA molecule. Its strands must first lie together and then separate. Line 39 contains statement instructing the computer to construct in the memory an *in silico* `GEN g1` of the type `gena` taking a substring from position 0 to 8 from the first strand of the DNA molecule as the argument for its constructor function.

Build Experiment2-1 and run the executable machine code.

The console window will display pictures documenting how the experiment for *in silico* gene expression (here, DNA transcription) proceeds:

```

Picture 1:
CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

Picture 2:
CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC

GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

Picture 3:
CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCAUGCGAC

GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

Picture 4:
CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

GCAUGCGAC

Press any key . . .

```

Create a new solution/project Experiment2-2 and add a new file Experiment2-2.cpp. Enter the following code within the file Experiment2-2.cpp:

```

1 //Experiment 2-2
2 /*Nikita Tirjatkin
3   Laboratory for in silico life construction
4   Januar 2010*/
5
6 #include <string>
7 #include <iostream>
8 using namespace std;
9
10 class gena
11 {
12     public:
13         string RNA;

```

```

14     gena(string Gene)
15     {
16         for(int i = 0; i < Gene.size(); i++)
17         {
18             if(Gene[i] == 'A') RNA.push_back('U');
19             else if(Gene[i] == 'C') RNA.push_back('G');
20             else if(Gene[i] == 'G') RNA.push_back('C');
21             else if(Gene[i] == 'T') RNA.push_back('A');
22             else RNA.push_back('-');
23         }
24     }
25 };
26
27 class genb:public gena
28 {
29     public:
30     string Polypeptide;
31     genb(string Gene):gena(Gene)
32     {
33         for(int i = 0; i < RNA.size(); i += 3)
34         {
35             if(RNA.substr(i, 3) == "GCA" ||
36                RNA.substr(i, 3) == "GCC" ||
37                RNA.substr(i, 3) == "GCG" ||
38                RNA.substr(i, 3) == "GCU") Polypeptide.push_back('A');
39             else if(RNA.substr(i, 3) == "UGC" ||
40                RNA.substr(i, 3) == "UGU") Polypeptide.push_back('C');
41             else if(RNA.substr(i, 3) == "GAC" ||
42                RNA.substr(i, 3) == "GAU") Polypeptide.push_back('D');
43             else if(RNA.substr(i, 3) == "GAA" ||
44                RNA.substr(i, 3) == "GAG") Polypeptide.push_back('E');
45             else if(RNA.substr(i, 3) == "UUC" ||
46                RNA.substr(i, 3) == "UUU") Polypeptide.push_back('F');
47             else if(RNA.substr(i, 3) == "GGA" ||
48                RNA.substr(i, 3) == "GGC" ||
49                RNA.substr(i, 3) == "GGG" ||
50                RNA.substr(i, 3) == "GGU") Polypeptide.push_back('G');
51             else if(RNA.substr(i, 3) == "CAC" ||
52                RNA.substr(i, 3) == "CAU") Polypeptide.push_back('H');
53             else if(RNA.substr(i, 3) == "AUA" ||
54                RNA.substr(i, 3) == "AUC" ||
55                RNA.substr(i, 3) == "AUU") Polypeptide.push_back('I');
56             else if(RNA.substr(i, 3) == "AAA" ||
57                RNA.substr(i, 3) == "AAG") Polypeptide.push_back('K');
58             else if(RNA.substr(i, 3) == "CUA" ||
59                RNA.substr(i, 3) == "CUC" ||
60                RNA.substr(i, 3) == "CUG" ||
61                RNA.substr(i, 3) == "CUU" ||
62                RNA.substr(i, 3) == "UUA" ||
63                RNA.substr(i, 3) == "UUG") Polypeptide.push_back('L');
64             else if(RNA.substr(i, 3) == "AUG") Polypeptide.push_back('M');
65             else if(RNA.substr(i, 3) == "AAC" ||
66                RNA.substr(i, 3) == "AAU") Polypeptide.push_back('N');
67             else if(RNA.substr(i, 3) == "CCA" ||
68                RNA.substr(i, 3) == "CCC" ||
69                RNA.substr(i, 3) == "CCG" ||
70                RNA.substr(i, 3) == "CCU") Polypeptide.push_back('P');
71             else if(RNA.substr(i, 3) == "CAA" ||
72                RNA.substr(i, 3) == "CAG") Polypeptide.push_back('Q');
73             else if(RNA.substr(i, 3) == "AGA" ||
74                RNA.substr(i, 3) == "AGG" ||
75                RNA.substr(i, 3) == "CGA" ||
76                RNA.substr(i, 3) == "CGC" ||
77                RNA.substr(i, 3) == "CGG" ||
78                RNA.substr(i, 3) == "CGU") Polypeptide.push_back('R');
79             else if(RNA.substr(i, 3) == "AGC" ||
80                RNA.substr(i, 3) == "AGU" ||
81                RNA.substr(i, 3) == "UCA" ||
82                RNA.substr(i, 3) == "UCC" ||
83                RNA.substr(i, 3) == "UCG" ||
84                RNA.substr(i, 3) == "UCU") Polypeptide.push_back('S');
85             else if(RNA.substr(i, 3) == "ACA" ||
86                RNA.substr(i, 3) == "ACC" ||
87                RNA.substr(i, 3) == "ACG" ||

```

```

88         RNA.substr(i, 3) == "ACU") Polypeptide.push_back('T');
89     else if(RNA.substr(i, 3) == "GUA" ||
90            RNA.substr(i, 3) == "GUC" ||
91            RNA.substr(i, 3) == "GUG" ||
92            RNA.substr(i, 3) == "GUU") Polypeptide.push_back('V');
93     else if(RNA.substr(i, 3) == "UGG") Polypeptide.push_back('W');
94     else if(RNA.substr(i, 3) == "UAC" ||
95            RNA.substr(i, 3) == "UAU") Polypeptide.push_back('Y');
96     else Polypeptide.push_back('-');
97     }
98 }
99 };
100
101 int main()
102 {
103     string s0a, s0b;
104     s0a="CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC";
105     s0b="GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG";
106     cout << "Picture 1:\n\n";
107     << s0a << endl
108     << s0b << "\n\n";
109     cout << "Picture 2:\n\n";
110     << s0a << "\n\n";
111     << s0b << "\n\n";
112
113     genb g3(s0a.substr(18, 9));
114     cout << "Picture 3:\n\n";
115     << s0a << endl
116     << "                " << g3.RNA << "\n\n";
117     << s0b << "\n\n";
118     cout << "Picture 4:\n\n";
119     << s0a << endl
120     << s0b << "\n\n";
121     << "                " << g3.RNA << "\n\n";
122     cout << "Picture 5:\n\n";
123     << s0a << endl
124     << s0b << "\n\n";
125     << "                " << g3.RNA << endl
126     << "                " << g3.Polypeptide << "\n\n";
127     cout << "Picture 6:\n\n";
128     << s0a << endl
129     << s0b << "\n\n";
130     << "                " << g3.RNA << "\n\n";
131     << "                " << g3.Polypeptide << "\n\n";
132
133     return 0;
134 }

```

Here, C++ program has to set up an extended experiment for construction of an *in silico* gene expression network. In addition to DNA transcription, this *in silico* GEN must involve RNA translation.

During RNA translation, the RNA molecule serves as a template for the synthesis of a polypeptide molecule. In this process, the triplets of ribonucleotides in RNA molecule – codons – determine amino acids to be attached to the polypeptide:

Codons..	... for amino acid
GCA, GCC, GCG, GCU	Alanin (Ala, A)
UGC, UGU	Cystein (Cys, C)
GAC, GAU	Asparaginsäure (Asp, D)
GAA, GAG	Glutaminsäure (Glu, E)
UUC, UUU	Phenylalanin (Phe, F)
GGA, GGC, GGG, GGU	Glycin (Gly, G)
CAC, CAU	Histidin (His, H)
AUA, AUC, AUU	Isoleucin (Ile, I)

AAA, AAG	Lysin (Lys, K)
CUA, CUC, CUG, CUU, UUA, UUG	Leucin (Leu, L)
AUG	Methionin (Met, M)
AAC, AAU	Asparagin (Asn, N)
CCA, CCC, CCG, CCU	Prolin (Pro, P)
CAA, CAG	Glutamin (Gln, Q)
AGA, AGG, CGA, CGC, CGG, CGU	Arginin (Arg, R)
AGC, AGU, UCA, UCC, UCG, UCU	Serin (Ser, S)
ACA, ACC, ACG, ACU	Threonin (Thr, T)
GUA, GUC, GUG, GUU	Valin (Val, V)
UGG	Tryptophan (Trp, W)
UAC, UAU	Tyrosin (Tyr, Y)

So, the sequence of codons within RNA molecule determines the sequence of amino acids within polypeptide molecule to be synthesized. Accordingly, a complementary polypeptide replica appears on the RNA template. Note that RNA molecule persists RNA translation without to be exhausted.

A new object type `genb` for *in silico* GENs involving DNA transcription and RNA translation can be defined as derived from the object type `gena` as shown in lines from 27 to 99. To declare an object type as derived from another object type, a colon `:` is used. If one object type is declared as derived from another object type, the derived object type automatically equires some member objects and member functions of the base object type in addition to its own. Here, the derived object type `genb` acquires from the base object type `gena` the member object `RNA` in addition to its own member object `Polypeptide` and the constructor function `gena` in addition to its own constructor function `genb`. If an object of the type `genb` will be constructed in the memory of computer, the constructor function `gena` will be called before the constructor function `genb`. The body of the constructor function `genb` contains code for control structure synthesizing polypeptide replica (here, `Polypeptide`) on the RNA templates (here, `RNA`). Characters for *in silico* ribonucleotides (**A**, **C**, **G**, and **U**) and amino acids (**A**, **C**, **D**, **E**, **F**, **G**, **H**, **I**, **K**, **L**, **M**, **N**, **P**, **Q**, **R**, **S**, **T**, **V**, **W** and **Y**) are chosen according to their conventional designation. Note, the sign `||` is used as logical operator OR.

Lines from 103 to 111 contain code instructing the computer to construct in the memory an *in silico* DNA molecule. Its strands must first lie together and then separate. Line 113 contains statement instructing the computer to construct in the memory an *in silico* GEN `g3` of the type `genb` taking a substring from position 18 to 26 from the first strand of the DNA molecule as the argument for its constructor function.

Build Experiment2-2 and run the executable machine code.

The console window will display pictures documenting how the experiment for *in silico* gene expression (here, DNA transcription + RNA translation) proceeds:

Picture 1:

```
CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTTCGGACACATAAACTAATGAACCCACAAGAAGCACAGTATGGTACGCCTGTGATGAG
```

Picture 2:

```
CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
```

```
GCATGCGACGAATTCGGACACATAAAAATAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG
```

Picture 3:

```
CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
CACAUAAAA
```

```
GCATGCGACGAATTCGGACACATAAAAATAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG
```

Picture 4:

```
CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAAATAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG
```

```
CACAUAAAA
```

Picture 5:

```
CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAAATAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG
```

```
CACAUAAAA
```

```
HIK
```

Picture 6:

```
CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAAATAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG
```

```
CACAUAAAA
```

```
HIK
```

Press any key . . .

Create a new solution/project Experiment2-3 and add a new file Experiment2-3.cpp. Enter the following code within the file Experiment2-3.cpp:

```
1 //Experiment 2-3
2 /*Nikita Tirjatkin
3 Laboratory for in silico life construction
4 Januar 2010*/
5
6 #include <string>
7 #include <iostream>
8 using namespace std;
9
10 class gena
11 {
12 public:
13     string RNA;
14     gena(string Gene)
15     {
16         for(int i = 0; i < Gene.size(); i++)
17         {
18             if(Gene[i] == 'A') RNA.push_back('U');
19             else if(Gene[i] == 'C') RNA.push_back('G');
20             else if(Gene[i] == 'G') RNA.push_back('C');
21             else if(Gene[i] == 'T') RNA.push_back('A');
22             else RNA.push_back('-');
23         }
24     }
25 };
26
27 class genb:public gena
28 {
```

```

29 public:
30     string Polypeptide;
31     genb(string Gene):gena(Gene)
32     {
33         for(int i = 0; i < RNA.size(); i += 3)
34         {
35             if(RNA.substr(i, 3) == "GCA" ||
36                RNA.substr(i, 3) == "GCC" ||
37                RNA.substr(i, 3) == "GCG" ||
38                RNA.substr(i, 3) == "GCU") Polypeptide.push_back('A');
39             else if(RNA.substr(i, 3) == "UGC" ||
40                    RNA.substr(i, 3) == "UGU") Polypeptide.push_back('C');
41             else if(RNA.substr(i, 3) == "GAC" ||
42                    RNA.substr(i, 3) == "GAU") Polypeptide.push_back('D');
43             else if(RNA.substr(i, 3) == "GAA" ||
44                    RNA.substr(i, 3) == "GAG") Polypeptide.push_back('E');
45             else if(RNA.substr(i, 3) == "UUC" ||
46                    RNA.substr(i, 3) == "UUU") Polypeptide.push_back('F');
47             else if(RNA.substr(i, 3) == "GGA" ||
48                    RNA.substr(i, 3) == "GGC" ||
49                    RNA.substr(i, 3) == "GGG" ||
50                    RNA.substr(i, 3) == "GGU") Polypeptide.push_back('G');
51             else if(RNA.substr(i, 3) == "CAC" ||
52                    RNA.substr(i, 3) == "CAU") Polypeptide.push_back('H');
53             else if(RNA.substr(i, 3) == "AUA" ||
54                    RNA.substr(i, 3) == "AUC" ||
55                    RNA.substr(i, 3) == "AUU") Polypeptide.push_back('I');
56             else if(RNA.substr(i, 3) == "AAA" ||
57                    RNA.substr(i, 3) == "AAG") Polypeptide.push_back('K');
58             else if(RNA.substr(i, 3) == "CUA" ||
59                    RNA.substr(i, 3) == "CUC" ||
60                    RNA.substr(i, 3) == "CUG" ||
61                    RNA.substr(i, 3) == "CUU" ||
62                    RNA.substr(i, 3) == "UUA" ||
63                    RNA.substr(i, 3) == "UUG") Polypeptide.push_back('L');
64             else if(RNA.substr(i, 3) == "AUG") Polypeptide.push_back('M');
65             else if(RNA.substr(i, 3) == "AAC" ||
66                    RNA.substr(i, 3) == "AAU") Polypeptide.push_back('N');
67             else if(RNA.substr(i, 3) == "CCA" ||
68                    RNA.substr(i, 3) == "CCC" ||
69                    RNA.substr(i, 3) == "CCG" ||
70                    RNA.substr(i, 3) == "CCU") Polypeptide.push_back('P');
71             else if(RNA.substr(i, 3) == "CAA" ||
72                    RNA.substr(i, 3) == "CAG") Polypeptide.push_back('Q');
73             else if(RNA.substr(i, 3) == "AGA" ||
74                    RNA.substr(i, 3) == "AGG" ||
75                    RNA.substr(i, 3) == "CGA" ||
76                    RNA.substr(i, 3) == "CGC" ||
77                    RNA.substr(i, 3) == "CGG" ||
78                    RNA.substr(i, 3) == "CGU") Polypeptide.push_back('R');
79             else if(RNA.substr(i, 3) == "AGC" ||
80                    RNA.substr(i, 3) == "AGU" ||
81                    RNA.substr(i, 3) == "UCA" ||
82                    RNA.substr(i, 3) == "UCC" ||
83                    RNA.substr(i, 3) == "UCG" ||
84                    RNA.substr(i, 3) == "UCU") Polypeptide.push_back('S');
85             else if(RNA.substr(i, 3) == "ACA" ||
86                    RNA.substr(i, 3) == "ACC" ||
87                    RNA.substr(i, 3) == "ACG" ||
88                    RNA.substr(i, 3) == "ACU") Polypeptide.push_back('T');
89             else if(RNA.substr(i, 3) == "GUA" ||
90                    RNA.substr(i, 3) == "GUC" ||
91                    RNA.substr(i, 3) == "GUG" ||
92                    RNA.substr(i, 3) == "GUU") Polypeptide.push_back('V');
93             else if(RNA.substr(i, 3) == "UGG") Polypeptide.push_back('W');
94             else if(RNA.substr(i, 3) == "UAC" ||
95                    RNA.substr(i, 3) == "UAU") Polypeptide.push_back('Y');
96             else
97                 Polypeptide.push_back('-');
98         }
99     };
100
101 class gena:public genb
102 {

```

```

103     public:
104         char Monomer;
105         genca(string Gene, char a, char b, char c):genb(Gene)
106         {
107             Monomer = c;
108         }
109     };
110
111     int main()
112     {
113         string s0a, s0b;
114         s0a="CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC";
115         s0b="GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG";
116         cout << "Picture 1:\n\n"
117             << s0a << endl
118             << s0b << "\n\n";
119         cout << "Picture 2:" << "\n\n"
120             << s0a << "\n\n\n"
121             << s0b << "\n\n";
122
123         genca g6(s0a.substr(45, 9), 'U', 'V', 'X');
124         cout << "Picture 3:\n\n"
125             << s0a << endl
126             << "                " << g6.RNA << "\n\n"
127             << s0b << "\n\n";
128         cout << "Picture 4:\n\n"
129             << s0a << endl
130             << s0b << "\n\n"
131             << "                " << g6.RNA << "\n\n";
132         cout << "Picture 5:\n\n"
133             << s0a << endl
134             << s0b << "\n\n"
135             << "                " << g6.RNA << endl
136             << "                " << g6.Polypeptide
137             << "\n\n";
138         cout << "Picture 6:\n\n"
139             << s0a << endl
140             << s0b << "\n\n"
141             << "                " << g6.RNA << "\n\n"
142             << "                " << g6.Polypeptide
143             << "\n\n";
144         cout << "Picture 7:\n\n"
145             << s0a << endl
146             << s0b << "\n\n"
147             << "                " << g6.RNA << "\n\n"
148             << "                " << g6.Polypeptide
149             << endl
150             << "                U V" << "\n\n";
151         cout << "Picture 8:\n\n"
152             << s0a << endl
153             << s0b << "\n\n"
154             << "                " << g6.RNA << "\n\n"
155             << "                " << g6.Polypeptide
156             << endl
157             << "                " << g6.Monomer
158             << "\n\n";
159         cout << "Picture 9:\n\n"
160             << s0a << endl
161             << s0b << "\n\n"
162             << "                " << g6.RNA << "\n\n"
163             << "                " << g6.Polypeptide
164             << "\n\n"
165             << "                " << g6.Monomer
166             << "\n\n";
167
168         return 0;
169     }

```

Here, C++ program has to set up an extended experiment for construction of an *in silico* gene expression network. In addition to DNA transcription and RNA translation, this *in silico* GEN must involve catalysis.

During catalysis, the catalyst serves as a template for the reaction that otherwise could occur too slowly for life. The catalyst does its job of catalysis by grappling with one or more substrate molecules and interacting with them to make or break chemical bonds. The catalyst is usually very specific for the chemical reaction it catalyses, and the specificity lies in a sophisticated configuration of atoms at one or more active sites of catalyst. Only restricted set of substrate molecules can recognize this configuration and bind it. In catalysts, this binding causes a conformational shift that promotes the reaction in any way. Thereafter, the catalyst releases reaction products, acquires its original conformation and is available for catalysis anew. Thus, the catalyst persists catalysis without to be exhausted.

A new object type for *in silico* GENs involving DNA transcription, RNA translation, and catalysis can be defined as derived from the object type `genb`. Lines from 101 to 109 contain definition of the object type `genca` for *in silico* GENs involving catalysis of the reaction  $A + B = C$ , where A, B, and C are monomers. Since the object type `genb` itself derives from the object type `gena`, the object type `genca` acquires from the base object type `genb` the member objects `RNA` and `Polypeptide` in addition to its own member object `Monomer` and the constructor functions `gena` and `genb` in addition to its own constructor function `genca`. If an object of the type `genca` will be constructed in the memory of computer, the constructor functions will be called in order `gena`, `genb`, `genca`. In addition to one parameter `Gene` of the type `string`, the constructor function `genca` has three parameters `a`, `b`, and `c` of the type `char`. Its body contains code for *in silico* catalysis of reaction type  $A + B = C$  on the polypeptide replica (here, `Polypeptide`). Characters for *in silico* monomers (`u`, `v` and `x`) are chosen arbitrary.

Lines from 113 to 121 contain code instructing the computer to construct in the memory an *in silico* DNA molecule. Its strands must first lie together and then separate. Line 123 contains statement instructing the computer to construct in the memory an *in silico* GEN `g6` of the type `genca` taking a substring from position 45 to 53 from the first strand of the DNA molecule as the first argument for its constructor function. Additionally, characters `u`, `v`, and `x` are used as other three arguments.

Build Experiment2-3 and run the executable machine code.

The console window will display pictures documenting how the experiment for *in silico* gene expression (here, DNA transcription + RNA translation + catalysis, where the reaction of type  $A + B = C$  is catalysed) proceeds:

Picture 1:

```
CGTACGCTGCTTAAGCCTGTGATATTTGATTAAGTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG
```

Picture 2:

```
CGTACGCTGCTTAAGCCTGTGATATTTGATTAAGTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
```

```
GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG
```

Picture 3:

```
CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
AGCACAGUA
```

```
GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG
```

Picture 4:

```
CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG
```

```
AGCACAGUA
```

Picture 5:

```
CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG
```

```
AGCACAGUA
STV
```

Picture 6:

```
CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG
```

```
AGCACAGUA
STV
```

Picture 7:

```
CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG
```

```
AGCACAGUA
STV
U V
```

Picture 8:

```
CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG
```

```
AGCACAGUA
STV
X
```

Picture 9:

```
CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG
```

```
AGCACAGUA
STV
X
```

Press any key . . .

Create a new solution/project Experiment2-4 and add a new file Experiment2-4.cpp. Enter the following code within the file Experiment2-4.cpp:

```
1 //Experiment 2-4
2 /*Nikita Tirjatkin
3 Laboratory for in silico life construction
```

```

4   Januar 2010*/
5
6   #include <string>
7   #include <iostream>
8   using namespace std;
9
10  class gena
11  {
12  public:
13      string RNA;
14      gena(string Gene)
15      {
16          for(int i = 0; i < Gene.size(); i++)
17          {
18              if(Gene[i] == 'A') RNA.push_back('U');
19              else if(Gene[i] == 'C') RNA.push_back('G');
20              else if(Gene[i] == 'G') RNA.push_back('C');
21              else if(Gene[i] == 'T') RNA.push_back('A');
22              else RNA.push_back('-');
23          }
24      }
25  };
26
27  class genb:public gena
28  {
29  public:
30      string Polypeptide;
31      genb(string Gene):gena(Gene)
32      {
33          for(int i = 0; i < RNA.size(); i += 3)
34          {
35              if(RNA.substr(i, 3) == "GCA" ||
36                 RNA.substr(i, 3) == "GCC" ||
37                 RNA.substr(i, 3) == "GCG" ||
38                 RNA.substr(i, 3) == "GCU") Polypeptide.push_back('A');
39              else if(RNA.substr(i, 3) == "UGC" ||
40                     RNA.substr(i, 3) == "UGU") Polypeptide.push_back('C');
41              else if(RNA.substr(i, 3) == "GAC" ||
42                     RNA.substr(i, 3) == "GAU") Polypeptide.push_back('D');
43              else if(RNA.substr(i, 3) == "GAA" ||
44                     RNA.substr(i, 3) == "GAG") Polypeptide.push_back('E');
45              else if(RNA.substr(i, 3) == "UUC" ||
46                     RNA.substr(i, 3) == "UUU") Polypeptide.push_back('F');
47              else if(RNA.substr(i, 3) == "GGA" ||
48                     RNA.substr(i, 3) == "GGC" ||
49                     RNA.substr(i, 3) == "GGG" ||
50                     RNA.substr(i, 3) == "GGU") Polypeptide.push_back('G');
51              else if(RNA.substr(i, 3) == "CAC" ||
52                     RNA.substr(i, 3) == "CAU") Polypeptide.push_back('H');
53              else if(RNA.substr(i, 3) == "AUA" ||
54                     RNA.substr(i, 3) == "AUC" ||
55                     RNA.substr(i, 3) == "AUU") Polypeptide.push_back('I');
56              else if(RNA.substr(i, 3) == "AAA" ||
57                     RNA.substr(i, 3) == "AAG") Polypeptide.push_back('K');
58              else if(RNA.substr(i, 3) == "CUA" ||
59                     RNA.substr(i, 3) == "CUC" ||
60                     RNA.substr(i, 3) == "CUG" ||
61                     RNA.substr(i, 3) == "CUU" ||
62                     RNA.substr(i, 3) == "UUA" ||
63                     RNA.substr(i, 3) == "UUG") Polypeptide.push_back('L');
64              else if(RNA.substr(i, 3) == "AUG") Polypeptide.push_back('M');
65              else if(RNA.substr(i, 3) == "AAC" ||
66                     RNA.substr(i, 3) == "AAU") Polypeptide.push_back('N');
67              else if(RNA.substr(i, 3) == "CCA" ||
68                     RNA.substr(i, 3) == "CCC" ||
69                     RNA.substr(i, 3) == "CCG" ||
70                     RNA.substr(i, 3) == "CCU") Polypeptide.push_back('P');
71              else if(RNA.substr(i, 3) == "CAA" ||
72                     RNA.substr(i, 3) == "CAG") Polypeptide.push_back('Q');
73              else if(RNA.substr(i, 3) == "AGA" ||
74                     RNA.substr(i, 3) == "AGG" ||
75                     RNA.substr(i, 3) == "CGA" ||
76                     RNA.substr(i, 3) == "CGC" ||
77                     RNA.substr(i, 3) == "CGG" ||

```

```

78         RNA.substr(i, 3) == "CGU") Polypeptide.push_back('R');
79     else if(RNA.substr(i, 3) == "AGC" ||
80            RNA.substr(i, 3) == "AGU" ||
81            RNA.substr(i, 3) == "UCA" ||
82            RNA.substr(i, 3) == "UCC" ||
83            RNA.substr(i, 3) == "UCG" ||
84            RNA.substr(i, 3) == "UCU") Polypeptide.push_back('S');
85     else if(RNA.substr(i, 3) == "ACA" ||
86            RNA.substr(i, 3) == "ACC" ||
87            RNA.substr(i, 3) == "ACG" ||
88            RNA.substr(i, 3) == "ACU") Polypeptide.push_back('T');
89     else if(RNA.substr(i, 3) == "GUA" ||
90            RNA.substr(i, 3) == "GUC" ||
91            RNA.substr(i, 3) == "GUG" ||
92            RNA.substr(i, 3) == "GUU") Polypeptide.push_back('V');
93     else if(RNA.substr(i, 3) == "UGG") Polypeptide.push_back('W');
94     else if(RNA.substr(i, 3) == "UAC" ||
95            RNA.substr(i, 3) == "UAU") Polypeptide.push_back('Y');
96     else
97         Polypeptide.push_back('-');
98     }
99 };
100
101 class gencb:public genb
102 {
103     public:
104     char Monomer1;
105     char Monomer2;
106     gencb(string Gene, char a, char b, char c):genb(Gene)
107     {
108         Monomer1 = b;
109         Monomer2 = c;
110     }
111 };
112
113 int main()
114 {
115     string s0a, s0b;
116     s0a="CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC";
117     s0b="GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCCTGTGATGAG";
118     cout << "Picture 1:\n\n"
119           << s0a << endl
120           << s0b << "\n\n";
121     cout << "Picture 2:\n\n"
122           << s0a << "\n\n\n"
123           << s0b << "\n\n";
124
125     gencb g7(s0a.substr(54, 9), 'W', 'Y', 'Z');
126     cout << "Picture 3:\n\n"
127           << s0a << endl
128           << " " << g7.RNA
129           << "\n\n" << s0b << "\n\n";
130     cout << "Picture 4:\n\n"
131           << s0a << endl
132           << s0b << "\n\n"
133           << " " << g7.RNA
134           << "\n\n";
135     cout << "Picture 5:\n\n"
136           << s0a << endl
137           << s0b << "\n\n"
138           << " " << g7.RNA
139           << endl
140           << " "
141           << g7.Polypeptide << "\n\n";
142     cout << "Picture 6:\n\n"
143           << s0a << endl
144           << s0b << "\n\n"
145           << " " << g7.RNA
146           << "\n\n"
147           << " "
148           << g7.Polypeptide << "\n\n";
149     cout << "Picture 7:\n\n"
150           << s0a << endl
151           << s0b << "\n\n"

```

```

152         << "                                     " << g7.RNA
153         << "\n\n"
154         << "                                     "
155         << g7.Polypeptide << endl
156         << "                                     W"
157         << "\n\n";
158     cout << "Picture 8:\n\n"
159         << s0a << endl
160         << s0b << "\n\n"
161         << "                                     " << g7.RNA
162         << "\n\n"
163         << "                                     "
164         << g7.Polypeptide << endl
165         << "                                     "
166         << g7.Monomer1 << " " << g7.Monomer2 << "\n\n";
167     cout << "Picture 9:\n\n"
168         << s0a << endl
169         << s0b << "\n\n"
170         << "                                     " << g7.RNA
171         << "\n\n"
172         << "                                     "
173         << g7.Polypeptide << "\n\n"
174         << "                                     "
175         << g7.Monomer1 << " " << g7.Monomer2 << "\n\n";
176
177     return 0;
178 }

```

Here, C++ program has to set up an experiment for construction of an *in silico* gene expression network involving DNA transcription, RNA translation, and catalysis, where the reaction of type  $A = B + C$  must be catalysed. Lines from 101 to 111 contain definition of the new object type `gencb` for such *in silico* GENs. Expectedly, `gencb` looks like `genca`. However, it has two member objects. The difference is also seen in the body of the constructor function specifying the reaction type to be catalysed. Characters for *in silico* monomers (`w`, `y` and `z`) are chosen arbitrary.

Build Experiment2-4 and run the executable machine code.

The console window will display pictures documenting how the experiment for *in silico* gene expression (here, DNA transcription + RNA translation + catalysis, where the reaction of type  $A = B + C$  is catalysed) proceeds:

```

Picture 1:
CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

Picture 2:
CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC

GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

Picture 3:
CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
UGGUACGCC

GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

Picture 4:
CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC

```

```
GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG
```

```
UGGUACGCC
```

Picture 5:

```
CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG
```

```
UGGUACGCC
```

```
WYA
```

Picture 6:

```
CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG
```

```
UGGUACGCC
```

```
WYA
```

Picture 7:

```
CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG
```

```
UGGUACGCC
```

```
WYA
```

```
W
```

Picture 8:

```
CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG
```

```
UGGUACGCC
```

```
WYA
```

```
Y Z
```

Picture 9:

```
CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG
```

```
UGGUACGCC
```

```
WYA
```

```
Y Z
```

Press any key . . .

Create a new solution/project Experiment2-5 and add a new file Experiment2-5.cpp. Enter the following code within the file Experiment2-5.cpp:

```
1 //Experiment 2-5
2 /*Nikita Tirjatkin
3   Laboratory for in silico life construction
4   Januar 2010*/
5
6 #include <string>
7 #include <iostream>
8 using namespace std;
9
10 class gena
11 {
12     public:
```

```

13     string RNA;
14     gena(string Gene)
15     {
16         for(int i = 0; i < Gene.size(); i++)
17         {
18             if(Gene[i] == 'A') RNA.push_back('U');
19             else if(Gene[i] == 'C') RNA.push_back('G');
20             else if(Gene[i] == 'G') RNA.push_back('C');
21             else if(Gene[i] == 'T') RNA.push_back('A');
22             else
23                 RNA.push_back('-');
24         }
25     };
26
27     class genb:public gena
28     {
29     public:
30         string Polypeptide;
31         genb(string Gene):gena(Gene)
32         {
33             for(int i = 0; i < RNA.size(); i += 3)
34             {
35                 if(RNA.substr(i, 3) == "GCA" ||
36                    RNA.substr(i, 3) == "GCC" ||
37                    RNA.substr(i, 3) == "GCG" ||
38                    RNA.substr(i, 3) == "GCU") Polypeptide.push_back('A');
39                 else if(RNA.substr(i, 3) == "UGC" ||
40                    RNA.substr(i, 3) == "UGU") Polypeptide.push_back('C');
41                 else if(RNA.substr(i, 3) == "GAC" ||
42                    RNA.substr(i, 3) == "GAU") Polypeptide.push_back('D');
43                 else if(RNA.substr(i, 3) == "GAA" ||
44                    RNA.substr(i, 3) == "GAG") Polypeptide.push_back('E');
45                 else if(RNA.substr(i, 3) == "UUC" ||
46                    RNA.substr(i, 3) == "UUU") Polypeptide.push_back('F');
47                 else if(RNA.substr(i, 3) == "GGA" ||
48                    RNA.substr(i, 3) == "GGC" ||
49                    RNA.substr(i, 3) == "GGG" ||
50                    RNA.substr(i, 3) == "GGU") Polypeptide.push_back('G');
51                 else if(RNA.substr(i, 3) == "CAC" ||
52                    RNA.substr(i, 3) == "CAU") Polypeptide.push_back('H');
53                 else if(RNA.substr(i, 3) == "AUA" ||
54                    RNA.substr(i, 3) == "AUC" ||
55                    RNA.substr(i, 3) == "AUU") Polypeptide.push_back('I');
56                 else if(RNA.substr(i, 3) == "AAA" ||
57                    RNA.substr(i, 3) == "AAG") Polypeptide.push_back('K');
58                 else if(RNA.substr(i, 3) == "CUA" ||
59                    RNA.substr(i, 3) == "CUC" ||
60                    RNA.substr(i, 3) == "CUG" ||
61                    RNA.substr(i, 3) == "CUU" ||
62                    RNA.substr(i, 3) == "UUA" ||
63                    RNA.substr(i, 3) == "UUG") Polypeptide.push_back('L');
64                 else if(RNA.substr(i, 3) == "AUG") Polypeptide.push_back('M');
65                 else if(RNA.substr(i, 3) == "AAC" ||
66                    RNA.substr(i, 3) == "AAU") Polypeptide.push_back('N');
67                 else if(RNA.substr(i, 3) == "CCA" ||
68                    RNA.substr(i, 3) == "CCC" ||
69                    RNA.substr(i, 3) == "CCG" ||
70                    RNA.substr(i, 3) == "CCU") Polypeptide.push_back('P');
71                 else if(RNA.substr(i, 3) == "CAA" ||
72                    RNA.substr(i, 3) == "CAG") Polypeptide.push_back('Q');
73                 else if(RNA.substr(i, 3) == "AGA" ||
74                    RNA.substr(i, 3) == "AGG" ||
75                    RNA.substr(i, 3) == "CGA" ||
76                    RNA.substr(i, 3) == "CGC" ||
77                    RNA.substr(i, 3) == "CGG" ||
78                    RNA.substr(i, 3) == "CGU") Polypeptide.push_back('R');
79                 else if(RNA.substr(i, 3) == "AGC" ||
80                    RNA.substr(i, 3) == "AGU" ||
81                    RNA.substr(i, 3) == "UCA" ||
82                    RNA.substr(i, 3) == "UCC" ||
83                    RNA.substr(i, 3) == "UCG" ||
84                    RNA.substr(i, 3) == "UCU") Polypeptide.push_back('S');
85                 else if(RNA.substr(i, 3) == "ACA" ||
86                    RNA.substr(i, 3) == "ACC" ||

```

```

87         RNA.substr(i, 3) == "ACG" ||
88         RNA.substr(i, 3) == "ACU") Polypeptide.push_back('T');
89     else if(RNA.substr(i, 3) == "GUA" ||
90            RNA.substr(i, 3) == "GUC" ||
91            RNA.substr(i, 3) == "GUG" ||
92            RNA.substr(i, 3) == "GUU") Polypeptide.push_back('V');
93     else if(RNA.substr(i, 3) == "UGG") Polypeptide.push_back('W');
94     else if(RNA.substr(i, 3) == "UAC" ||
95            RNA.substr(i, 3) == "UAU") Polypeptide.push_back('Y');
96     else
97         Polypeptide.push_back('-');
98     }
99 };
100
101 class gencz:public genb
102 {
103     public:
104     string Replica;
105     gencz(string Gene, string Template):genb(Gene)
106     {
107         for(int i = 0; i < Template.size(); i++)
108         {
109             if(Template[i] == 'A') Replica.push_back('T');
110             else if(Template[i] == 'C') Replica.push_back('G');
111             else if(Template[i] == 'G') Replica.push_back('C');
112             else if(Template[i] == 'T') Replica.push_back('A');
113             else
114                 Replica.push_back('-');
115         }
116     };
117
118     int main()
119     {
120         string s0a, s0b;
121         s0a="CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC";
122         s0b="GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCCTGTGATGAG";
123         cout << "Picture 1:\n\n"
124              << s0a << endl
125              << s0b << "\n\n";
126         cout << "Picture 2:\n\n"
127              << s0a << "\n\n\n"
128              << s0b << "\n\n";
129
130         gencz g0a(s0a.substr(63, 9), s0a);
131         gencz g0b(s0a.substr(63, 9), s0b);
132         cout << "Picture 3:\n\n"
133              << s0a << endl
134              << "
135              << g0a.RNA << "\n\n"
136              << s0b << "\n\n";
137         cout << "Picture 4:\n\n"
138              << s0a << endl
139              << s0b << "\n\n"
140              << "
141              << g0a.RNA << "\n\n";
142         cout << "Picture 5:\n\n"
143              << s0a << endl
144              << s0b << "\n\n"
145              << "
146              << g0a.RNA << endl
147              << "
148              << g0a.Polypeptide << "\n\n";
149         cout << "Picture 6:\n\n"
150              << s0a << endl
151              << s0b << "\n\n"
152              << "
153              << g0a.RNA << "\n\n"
154              << "
155              << g0a.Polypeptide << "\n\n";
156         cout << "Picture 7:\n\n"
157              << s0a << endl
158              << g0a.Polypeptide << "\n\n"
159              << "
160              << g0b.Polypeptide << endl

```

```

161         << s0b << "\n\n";
162     cout << "Picture 8:\n\n"
163         << s0a << endl
164         << g0a.Replica << "\n\n"
165         << g0b.Replica << endl
166         << s0b << "\n\n";
167
168     return 0;
169 }

```

In living world, virtually all reactions are to be catalysed, inclusive all reactions of polymerization. The most important reaction of polymerization is DNA replication.

During DNA replication, strands of DNA molecule separate and each strand serves as a template by guiding the synthesis of complementary strand according to strict rules of base pairing:

Deoxyribonukleotide with the base...	Adenin	Cytosin	Guanin	Thymin
... attaches deoxyribonukleotide with the base...	Thymin	Guanin	Cytosin	Adenin

When the DNA replication is complete, each original DNA molecule is replaced by its two identical copies. Note that the DNA molecule strands persist DNA transcription without to be exhausted.

Here, C++ program has to set up an experiment for construction of an *in silico* gene expression network involving DNA transcription, RNA translation, and catalysis, where the DNA replication must be catalysed. Lines from 101 to 116 contain definition of the new object type `gencz` for such *in silico* GENs. It too is declared as derived from `genb`. Its constructor function has two parameters `Gene` and `Template` of the type `string`. The body of the constructor function contains code for control structure synthesizing DNA replica (here, `Replica`) on the DNA templates (here, `Template`). Characters for *in silico* deoxyribonucleotides (**A**, **C**, **G**, and **T**) are chosen according to their conventional designation.

Lines from 120 to 128 contain code instructing the computer to construct in the memory an *in silico* DNA molecule. Its strands must first lie together and then separate. Lines 130 and 131 contain statements instructing the computer to construct in the memory two *in silico* GENs `g0a` and `g0b` of the type `gencz` taking a substring from position 63 to 71 from the first strand of the DNA molecule as the first argument for their constructor functions. Additionally, each constructor function takes the corresponding DNA molecule strand as the second argument.

Build Experiment2-5 and run the executable machine code.

The console window will display pictures documenting how the experiment for *in silico* gene expression (here, DNA transcription + RNA translation + catalysis, where the DNA replication is catalysed) proceeds:

Picture 1:

```

CGTACGCTGCTTAAGCCTGTGATATTTGATTAAGTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

```

Picture 2:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC

GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

Picture 3:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC  
UGUGAUGAG

GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

Picture 4:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC  
GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

UGUGAUGAG

Picture 5:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC  
GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

UGUGAUGAG  
CDE

Picture 6:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC  
GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

UGUGAUGAG  
CDE

Picture 7:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC  
CDE

CDE

GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

Picture 8:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC  
GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC  
GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

Press any key . . .

## Experiment 3

**Objective:** Learn how to construct *in silico* genome expression networks (GENomes).

Create a new solution/project Experiment3-1 and add a new file Experiment3-1.cpp. Enter the following code within the file Experiment3-1.cpp:

```

1 //Experiment 3-1
2 /*Nikita Tirjatkin
3 Laboratory for in silico life construction
4 Januar 2010*/
5
6 #include <string>
7 #include <vector>
8 #include <iostream>
9 using namespace std;
10
11 class gena
12 {
13 public:
14     string RNA;
15     gena(string Gene)
16     {
17         for(int i = 0; i < Gene.size(); i++)
18         {
19             if(Gene[i] == 'A') RNA.push_back('U');
20             else if(Gene[i] == 'C') RNA.push_back('G');
21             else if(Gene[i] == 'G') RNA.push_back('C');
22             else if(Gene[i] == 'T') RNA.push_back('A');
23             else RNA.push_back('-');
24         }
25     }
26 };
27
28 class genb:public gena
29 {
30 public:
31     string Polypeptide;
32     genb(string Gene):gena(Gene)
33     {
34         for(int i = 0; i < RNA.size(); i += 3)
35         {
36             if(RNA.substr(i, 3) == "GCA" ||
37                RNA.substr(i, 3) == "GCC" ||
38                RNA.substr(i, 3) == "GCG" ||
39                RNA.substr(i, 3) == "GCU") Polypeptide.push_back('A');
40             else if(RNA.substr(i, 3) == "UGC" ||
41                RNA.substr(i, 3) == "UGU") Polypeptide.push_back('C');
42             else if(RNA.substr(i, 3) == "GAC" ||
43                RNA.substr(i, 3) == "GAU") Polypeptide.push_back('D');
44             else if(RNA.substr(i, 3) == "GAA" ||
45                RNA.substr(i, 3) == "GAG") Polypeptide.push_back('E');
46             else if(RNA.substr(i, 3) == "UUC" ||
47                RNA.substr(i, 3) == "UUU") Polypeptide.push_back('F');
48             else if(RNA.substr(i, 3) == "GGA" ||
49                RNA.substr(i, 3) == "GGC" ||
50                RNA.substr(i, 3) == "GGG" ||
51                RNA.substr(i, 3) == "GGU") Polypeptide.push_back('G');
52             else if(RNA.substr(i, 3) == "CAC" ||
53                RNA.substr(i, 3) == "CAU") Polypeptide.push_back('H');
54             else if(RNA.substr(i, 3) == "AUA" ||
55                RNA.substr(i, 3) == "AUC" ||
56                RNA.substr(i, 3) == "AUU") Polypeptide.push_back('I');
57             else if(RNA.substr(i, 3) == "AAA" ||
58                RNA.substr(i, 3) == "AAG") Polypeptide.push_back('K');
59             else if(RNA.substr(i, 3) == "CUA" ||
60                RNA.substr(i, 3) == "CUC" ||
61                RNA.substr(i, 3) == "CUG" ||
62                RNA.substr(i, 3) == "CUU" ||
63                RNA.substr(i, 3) == "UUA" ||

```

```

64         RNA.substr(i, 3) == "UUG") Polypeptide.push_back('L');
65     else if(RNA.substr(i, 3) == "AUG") Polypeptide.push_back('M');
66     else if(RNA.substr(i, 3) == "AAC" ||
67             RNA.substr(i, 3) == "AAU") Polypeptide.push_back('N');
68     else if(RNA.substr(i, 3) == "CCA" ||
69             RNA.substr(i, 3) == "CCC" ||
70             RNA.substr(i, 3) == "CCG" ||
71             RNA.substr(i, 3) == "CCU") Polypeptide.push_back('P');
72     else if(RNA.substr(i, 3) == "CAA" ||
73             RNA.substr(i, 3) == "CAG") Polypeptide.push_back('Q');
74     else if(RNA.substr(i, 3) == "AGA" ||
75             RNA.substr(i, 3) == "AGG" ||
76             RNA.substr(i, 3) == "CGA" ||
77             RNA.substr(i, 3) == "CGC" ||
78             RNA.substr(i, 3) == "CGG" ||
79             RNA.substr(i, 3) == "CGU") Polypeptide.push_back('R');
80     else if(RNA.substr(i, 3) == "AGC" ||
81             RNA.substr(i, 3) == "AGU" ||
82             RNA.substr(i, 3) == "UCA" ||
83             RNA.substr(i, 3) == "UCC" ||
84             RNA.substr(i, 3) == "UCG" ||
85             RNA.substr(i, 3) == "UCU") Polypeptide.push_back('S');
86     else if(RNA.substr(i, 3) == "ACA" ||
87             RNA.substr(i, 3) == "ACC" ||
88             RNA.substr(i, 3) == "ACG" ||
89             RNA.substr(i, 3) == "ACU") Polypeptide.push_back('T');
90     else if(RNA.substr(i, 3) == "GUA" ||
91             RNA.substr(i, 3) == "GUC" ||
92             RNA.substr(i, 3) == "GUG" ||
93             RNA.substr(i, 3) == "GUU") Polypeptide.push_back('V');
94     else if(RNA.substr(i, 3) == "UGG") Polypeptide.push_back('W');
95     else if(RNA.substr(i, 3) == "UAC" ||
96             RNA.substr(i, 3) == "UAU") Polypeptide.push_back('Y');
97     else
98         Polypeptide.push_back('-');
99     }
100 };
101
102 class genca:public genb
103 {
104     public:
105     char Monomer;
106     genca(string Gene, char a, char b, char c):genb(Gene)
107     {
108         Monomer = c;
109     }
110 };
111
112 class gencb:public genb
113 {
114     public:
115     char Monomer1;
116     char Monomer2;
117     gencb(string Gene, char a, char b, char c):genb(Gene)
118     {
119         Monomer1 = b;
120         Monomer2 = c;
121     }
122 };
123
124 class gen cz:public genb
125 {
126     public:
127     string Replica;
128     gen cz(string Gene, string Template):genb(Gene)
129     {
130         for(int i = 0; i < Template.size(); i++)
131         {
132             if(Template[i] == 'A') Replica.push_back('T');
133             else if(Template[i] == 'C') Replica.push_back('G');
134             else if(Template[i] == 'G') Replica.push_back('C');
135             else if(Template[i] == 'T') Replica.push_back('A');
136             else
137                 Replica.push_back('-');

```

```

138     }
139 };
140
141 class cell
142 {
143     public:
144         vector<string> DNAs;
145         vector<string> tRNAs;
146         vector<string> rRNAs;
147         vector<string> Polypeptides1;
148         vector<string> Polypeptides2;
149         vector<string> Polypeptides3;
150         vector<char> Monomers1;
151         vector<char> Monomers2;
152         vector<char> Monomers3;
153
154         cell(vector<string> v0,
155             vector<string> v1,
156             vector<string> v2,
157             vector<string> v3,
158             vector<string> v4,
159             vector<string> v5,
160             vector<char> v6,
161             vector<char> v7,
162             vector<char> v8)
163         {
164             DNAs = v0;
165             tRNAs = v1;
166             rRNAs = v2;
167             Polypeptides1 = v3;
168             Polypeptides2 = v4;
169             Polypeptides3 = v5;
170             Monomers1 = v6;
171             Monomers2 = v7;
172             Monomers3 = v8;
173
174             for(int i = 0; i < 10; i++)
175             {
176                 gena g1(DNAs[0].substr(0, 9)); tRNAs.push_back(g1.RNA);
177                 gena g2(DNAs[0].substr(9, 9)); rRNAs.push_back(g2.RNA);
178                 genb g3(DNAs[0].substr(18, 9)); Polypeptides1.push_back(g3.Polypeptide);
179                 genb g4(DNAs[0].substr(27, 9)); Polypeptides2.push_back(g4.Polypeptide);
180                 genb g5(DNAs[0].substr(36, 9)); Polypeptides3.push_back(g5.Polypeptide);
181                 gena g6(DNAs[0].substr(45, 9), 'U', 'V', 'X');
182                 Monomers1.push_back(g6.Monomer);
183                 genb g7(DNAs[0].substr(54, 9), 'W', 'Y', 'Z');
184                 Monomers2.push_back(g7.Monomer1);
185                 Monomers3.push_back(g7.Monomer2);
186             }
187             vector<string> DNArePLICAS;
188             genzc g0a(DNAs[0].substr(63, 9), DNAs[0]);
189             DNArePLICAS.push_back(g0a.Replica);
190             genzc g0b(DNAs[0].substr(63, 9), DNAs[1]);
191             DNArePLICAS.push_back(g0b.Replica);
192             DNAs.insert(DNAs.begin() + 1, DNArePLICAS.begin(), DNArePLICAS.end());
193         }
194 };
195
196 int main()
197 {
198     string s0a, s0b;
199     s0a="CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC";
200     s0b="GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG";
201     vector<string> V0;
202     V0.push_back(s0a);
203     V0.push_back(s0b);
204     vector<string> V1(10, "GCAUGCGAC"), V2(10, "GCAUGCGAC");
205     vector<string> V3(10, "HIK"), V4(10, "LMN"), V5(10, "PQR");
206     vector<char> V6(10, 'X'), V7(10, 'Y'), V8(10, 'Z');
207     cout << "Picture 1:" << "\n\n";
208     for(int i = 0; i < V0.size(); i++)
209         cout << V0[i] << endl;
210     cout << "\n";
211     for(int i = 0; i < V1.size(); i++)

```

```

212         cout << V1[i] << " ";
213     cout << "\n\n";
214     for(int i = 0; i < V2.size(); i++)
215         cout << V2[i] << " ";
216     cout << "\n\n";
217     for(int i = 0; i < V3.size(); i++)
218         cout << V3[i] << " ";
219     cout << "\n\n";
220     for(int i = 0; i < V4.size(); i++)
221         cout << V4[i] << " ";
222     cout << "\n\n";
223     for(int i = 0; i < V5.size(); i++)
224         cout << V5[i] << " ";
225     cout << "\n\n";
226     for(int i = 0; i < V6.size(); i++)
227         cout << V6[i] << " ";
228     cout << "\n\n";
229     for(int i = 0; i < V7.size(); i++)
230         cout << V7[i] << " ";
231     cout << "\n\n";
232     for(int i = 0; i < V8.size(); i++)
233         cout << V8[i] << " ";
234     cout << "\n\n";
235
236     cell Cell(V0, V1, V2, V3, V4, V5, V6, V7, V8);
237     cout << "Picture 2:" << "\n\n";
238     for(int i = 0; i < Cell.DNAs.size(); i++)
239         cout << Cell.DNAs[i] << endl;
240     cout << "\n";
241     for(int i = 0; i < Cell.tRNAs.size(); i++)
242         cout << Cell.tRNAs[i] << " ";
243     cout << "\n\n";
244     for(int i = 0; i < Cell.rRNAs.size(); i++)
245         cout << Cell.rRNAs[i] << " ";
246     cout << "\n\n";
247     for(int i = 0; i < Cell.Polypeptides1.size(); i++)
248         cout << Cell.Polypeptides1[i] << " ";
249     cout << "\n\n";
250     for(int i = 0; i < Cell.Polypeptides2.size(); i++)
251         cout << Cell.Polypeptides2[i] << " ";
252     cout << "\n\n";
253     for(int i = 0; i < Cell.Polypeptides3.size(); i++)
254         cout << Cell.Polypeptides3[i] << " ";
255     cout << "\n\n";
356     for(int i = 0; i < Cell.Monomers1.size(); i++)
257         cout << Cell.Monomers1[i] << " ";
258     cout << "\n\n";
259     for(int i = 0; i < Cell.Monomers2.size(); i++)
260         cout << Cell.Monomers2[i] << " ";
261     cout << "\n\n";
262     for(int i = 0; i < Cell.Monomers3.size(); i++)
263         cout << Cell.Monomers3[i] << " ";
264     cout << "\n\n";
265
266     return 0;
267 }

```

Here, C++ program has to set up an experiment for construction of an *in silico* genome expression network (GENome). This life pattern is roughly equal to the cell. In cell (GENome), the information processing involves two tightly coupled reactions: genome expression and genome replication.

The genome is a limited set of genes and each gene is usually expressed separately to be fully converted into the corresponding element of the cell structure or function. Respectively, the cell is a composition of GENs. In cell, its GENs interact to maintain each other. In cell, its GENs interact to maintain each other. During gene expression in particular GEN, it is just the job of other GENs to provide necessary elements for gene

expression machinery. Collectively, GENs work to replicate the complete DNA. Respectively, the life history of the single cell begins with one cell but ends with two. Generally, the cell life history begins at the point where two newly produced sister cells halve the matrix inherited from the mother cell and each starts a self-dependent life. What the newborn cell has to do is just what its mother done: it starts its own genome expression which results in genome replication and in division in two daughter cells.

Here, it is reasonable to begin with an experiment for construction of an *in silico* genome expression network (GENome) with the simplest genome and the simplest life history. Environmental conditions too must be first extremely favourable: energy and all monomers for synthesis of DNA, RNA, and polypeptide molecules are in overflow. Lines from 141 to 194 contain definition of the new object type `cell` for *in silico* genome expression network (GENome). For simplicity, the number of genes in genome is restricted to 8. They must be expressed in linear order to double contents of 9 pools of *in silico* chemicals.

Respectively, the new object type `cell` has 9 member objects of the object type `vector` defined in the file `vector` of the Standard C++ Library. The object type `vector` is specifically designed for construction of objects that can hold/contain a pool of other objects and for operation on them. In Standard C++ Library, the object type `vector` is defined as a template. A template takes other object types as parameters. Note, the template parameters are surrounded by signs `<` and `>`. Here, object types `string` and `char` are used as template parameters. One container is necessary to hold DNA molecule strands. Two containers are for pools of RNAs (here, tRNAs and ribosomal RNAs). Three containers are for pools of polypeptides (here, RNA polymerases, ribosomal polypeptides, and polypeptides involved in cell division). Other three containers are for pools of monomers (here, X, Y, and Z).

Expectedly, the constructor function of the new object type `cell` has 9 parameters. The body of the constructor function contains code for control structure specifying how genes must be expressed to double contents of 9 pools.

Lines from 198 to 206 contain code instructing the computer to construct in the memory 9 pools of *in silico* chemicals. Line 236 contains statement instructing the computer to construct in the memory an *in silico* genome expression network `Cell` of the type `cell` taking all 9 pools as arguments for its constructor functions.

Build Experiment3-1 and run the executable machine code.

The console window will display pictures documenting how the experiment for *in silico* genome expression proceeds:

Picture 1:

```
CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

GCAUGCAC GCAUGCAC GCAUGCAC GCAUGCAC GCAUGCAC GCAUGCAC GCAUGCAC GCAUGCAC
GCAUGCAC GCAUGCAC

GCAUGCAC GCAUGCAC GCAUGCAC GCAUGCAC GCAUGCAC GCAUGCAC GCAUGCAC GCAUGCAC
```

```
GCAUGCGAC GCAUGCGAC
HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK
LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN
PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR
X X X X X X X X X X
Y Y Y Y Y Y Y Y Y Y
Z Z Z Z Z Z Z Z Z Z
```

Picture 2:

```
CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG
CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC
GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC
GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC

GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC
GCAUGCGAC GCAUGCGAC GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA
GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA

HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK
LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN
PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR
X X X X X X X X X X X X X X X X X X X X X
Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y
Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z
Press any key . . .
```

## Experiment 4

**Objective:** Learn how to construct *in silico* genome multiplication networks.

Create a new solution/project Experiment4-1 and add a new file Experiment4-1.cpp. Enter the following code within the file Experiment4-1.cpp:

```

1 //Experiment 4-1
2 /*Nikita Tirjatkin
3   Laboratory for in silico life construction
4   Januar 2010*/
5
6 #include <string>
7 #include <vector>
8 #include <deque>
9 #include <iostream>
10 using namespace std;
11
12 class gena
13 {
14     public:
15         string RNA;
16         gena(string Gene)
17         {
18             for(int i = 0; i < Gene.size(); i++)
19             {
20                 if(Gene[i] == 'A') RNA.push_back('U');
21                 else if(Gene[i] == 'C') RNA.push_back('G');
22                 else if(Gene[i] == 'G') RNA.push_back('C');
23                 else if(Gene[i] == 'T') RNA.push_back('A');
24                 else RNA.push_back('-');
25             }
26         }
27 };
28
29 class genb:public gena
30 {
31     public:
32         string Polypeptide;
33         genb(string Gene):gena(Gene)
34         {
35             for(int i = 0; i < RNA.size(); i += 3)
36             {
37                 if(RNA.substr(i, 3) == "GCA" ||
38                    RNA.substr(i, 3) == "GCC" ||
39                    RNA.substr(i, 3) == "GCG" ||
40                    RNA.substr(i, 3) == "GCU") Polypeptide.push_back('A');
41                 else if(RNA.substr(i, 3) == "UGC" ||
42                    RNA.substr(i, 3) == "UGU") Polypeptide.push_back('C');
43                 else if(RNA.substr(i, 3) == "GAC" ||
44                    RNA.substr(i, 3) == "GAU") Polypeptide.push_back('D');
45                 else if(RNA.substr(i, 3) == "GAA" ||
46                    RNA.substr(i, 3) == "GAG") Polypeptide.push_back('E');
47                 else if(RNA.substr(i, 3) == "UUC" ||
48                    RNA.substr(i, 3) == "UUU") Polypeptide.push_back('F');
49                 else if(RNA.substr(i, 3) == "GGA" ||
50                    RNA.substr(i, 3) == "GGC" ||
51                    RNA.substr(i, 3) == "GGG" ||
52                    RNA.substr(i, 3) == "GGU") Polypeptide.push_back('G');
53                 else if(RNA.substr(i, 3) == "CAC" ||
54                    RNA.substr(i, 3) == "CAU") Polypeptide.push_back('H');
55                 else if(RNA.substr(i, 3) == "AUA" ||
56                    RNA.substr(i, 3) == "AUC" ||
57                    RNA.substr(i, 3) == "AUU") Polypeptide.push_back('I');
58                 else if(RNA.substr(i, 3) == "AAA" ||
59                    RNA.substr(i, 3) == "AAG") Polypeptide.push_back('K');
60                 else if(RNA.substr(i, 3) == "CUA" ||
61                    RNA.substr(i, 3) == "CUC" ||
62                    RNA.substr(i, 3) == "CUG" ||
63                    RNA.substr(i, 3) == "CUU" ||

```

```

64         RNA.substr(i, 3) == "UUA" ||
65         RNA.substr(i, 3) == "UUG") Polypeptide.push_back('L');
66     else if(RNA.substr(i, 3) == "AUG") Polypeptide.push_back('M');
67     else if(RNA.substr(i, 3) == "AAC" ||
68         RNA.substr(i, 3) == "AAU") Polypeptide.push_back('N');
69     else if(RNA.substr(i, 3) == "CCA" ||
70         RNA.substr(i, 3) == "CCC" ||
71         RNA.substr(i, 3) == "CCG" ||
72         RNA.substr(i, 3) == "CCU") Polypeptide.push_back('P');
73     else if(RNA.substr(i, 3) == "CAA" ||
74         RNA.substr(i, 3) == "CAG") Polypeptide.push_back('Q');
75     else if(RNA.substr(i, 3) == "AGA" ||
76         RNA.substr(i, 3) == "AGG" ||
77         RNA.substr(i, 3) == "CGA" ||
78         RNA.substr(i, 3) == "CGC" ||
79         RNA.substr(i, 3) == "CGG" ||
80         RNA.substr(i, 3) == "CGU") Polypeptide.push_back('R');
81     else if(RNA.substr(i, 3) == "AGC" ||
82         RNA.substr(i, 3) == "AGU" ||
83         RNA.substr(i, 3) == "UCA" ||
84         RNA.substr(i, 3) == "UCC" ||
85         RNA.substr(i, 3) == "UCG" ||
86         RNA.substr(i, 3) == "UCU") Polypeptide.push_back('S');
87     else if(RNA.substr(i, 3) == "ACA" ||
88         RNA.substr(i, 3) == "ACC" ||
89         RNA.substr(i, 3) == "ACG" ||
90         RNA.substr(i, 3) == "ACU") Polypeptide.push_back('T');
91     else if(RNA.substr(i, 3) == "GUA" ||
92         RNA.substr(i, 3) == "GUC" ||
93         RNA.substr(i, 3) == "GUG" ||
94         RNA.substr(i, 3) == "GUU") Polypeptide.push_back('V');
95     else if(RNA.substr(i, 3) == "UGG") Polypeptide.push_back('W');
96     else if(RNA.substr(i, 3) == "UAC" ||
97         RNA.substr(i, 3) == "UAU") Polypeptide.push_back('Y');
98     else
99         Polypeptide.push_back('-');
100     }
101 };
102
103 class genca:public genb
104 {
105     public:
106         char Monomer;
107         genca(string Gene, char a, char b, char c):genb(Gene)
108         {
109             Monomer = c;
110         }
111 };
112
113 class gencb:public genb
114 {
115     public:
116         char Monomer1;
117         char Monomer2;
118         gencb(string Gene, char a, char b, char c):genb(Gene)
119         {
120             Monomer1 = b;
121             Monomer2 = c;
122         }
123 };
124
125 class gencz:public genb
126 {
127     public:
128         string Replica;
129         gencz(string Gene, string Template):genb(Gene)
130         {
131             for(int i = 0; i < Template.size(); i++)
132             {
133                 if(Template[i] == 'A') Replica.push_back('T');
134                 else if(Template[i] == 'C') Replica.push_back('G');
135                 else if(Template[i] == 'G') Replica.push_back('C');
136                 else if(Template[i] == 'T') Replica.push_back('A');
137                 else

```

```

138     }
139   }
140 };
141
142 class cell
143 {
144   public:
145     vector<string> DNAs;
146     vector<string> tRNAs;
147     vector<string> rRNAs;
148     vector<string> Polypeptides1;
149     vector<string> Polypeptides2;
150     vector<string> Polypeptides3;
151     vector<char> Monomers1;
152     vector<char> Monomers2;
153     vector<char> Monomers3;
154
155     cell(vector<string> v0,
156         vector<string> v1,
157         vector<string> v2,
158         vector<string> v3,
159         vector<string> v4,
160         vector<string> v5,
161         vector<char> v6,
162         vector<char> v7,
163         vector<char> v8)
164     {
165         DNAs = v0;
166         tRNAs = v1;
167         rRNAs = v2;
168         Polypeptides1 = v3;
169         Polypeptides2 = v4;
170         Polypeptides3 = v5;
171         Monomers1 = v6;
172         Monomers2 = v7;
173         Monomers3 = v8;
174
175         for(int i = 0; i < 10; i++)
176         {
177             gena g1(DNAs[0].substr(0, 9)); tRNAs.push_back(g1.RNA);
178             gena g2(DNAs[0].substr(9, 9)); rRNAs.push_back(g2.RNA);
179             genb g3(DNAs[0].substr(18, 9)); Polypeptides1.push_back(g3.Polypeptide);
180             genb g4(DNAs[0].substr(27, 9)); Polypeptides2.push_back(g4.Polypeptide);
181             genb g5(DNAs[0].substr(36, 9)); Polypeptides3.push_back(g5.Polypeptide);
182             gena g6(DNAs[0].substr(45, 9), 'U', 'V', 'X');
183             Monomers1.push_back(g6.Monomer);
184             genb g7(DNAs[0].substr(54, 9), 'W', 'Y', 'Z');
185             Monomers2.push_back(g7.Monomer1);
186             Monomers3.push_back(g7.Monomer2);
187         }
188         vector<string> DNArePLICAS;
189         genz g0a(DNAs[0].substr(63, 9), DNAs[0]);
190             DNArePLICAS.push_back(g0a.Replica);
191         genz g0b(DNAs[0].substr(63, 9), DNAs[1]);
192             DNArePLICAS.push_back(g0b.Replica);
193         DNAs.insert(DNAs.begin() + 1, DNArePLICAS.begin(), DNArePLICAS.end());
194     }
195 };
196
197 class cp
198 {
199   public:
200     deque<cell> Cells;
201     cp(cell ccell)
202     {
203         Cells.push_back(ccell);
204
205         while(Cells.size() <= Cells.max_size())
206         {
207             ccell = Cells.front();
208
209             cout << "\nCell Number: " << Cells.size() << "\n\n";
210             for(int i = 0; i < ccell.DNAs.size(); i++)
211                 cout << ccell.DNAs[i] << " ";

```

```

212 cout << "\n\n";
213 for(int i = 0; i < ccell.tRNAs.size(); i++)
214     cout << ccell.tRNAs[i] << " ";
215 cout << "\n\n";
216 for(int i = 0; i < ccell.rRNAs.size(); i++)
217     cout << ccell.rRNAs[i] << " ";
218 cout << "\n\n";
219 for(int i = 0; i < ccell.Polypeptides1.size(); i++)
220     cout << ccell.Polypeptides1[i] << " ";
221 cout << "\n\n";
222 for(int i = 0; i < ccell.Polypeptides2.size(); i++)
223     cout << ccell.Polypeptides2[i] << " ";
224 cout << "\n\n";
225 for(int i = 0; i < ccell.Polypeptides3.size(); i++)
226     cout << ccell.Polypeptides3[i] << " ";
227 cout << "\n\n";
228 for(int i = 0; i < ccell.Monomers1.size(); i++)
229     cout << ccell.Monomers1[i] << " ";
230 cout << "\n\n";
231 for(int i = 0; i < ccell.Monomers2.size(); i++)
232     cout << ccell.Monomers2[i] << " ";
233 cout << "\n\n";
234 for(int i = 0; i < ccell.Monomers3.size(); i++)
235     cout << ccell.Monomers3[i] << " ";
236 cout << "\n\n";
237
238 Cells.pop_front();
239
240 vector<string> lDNAs(ccell.DNAs.begin(),
241                   ccell.DNAs.begin() +
242                   ccell.DNAs.size()/2);
243 vector<string> rDNAs(ccell.DNAs.begin() +
244                   ccell.DNAs.size()/2,
245                   ccell.DNAs.end());
246 vector<string> ltRNAs(ccell.tRNAs.begin(),
247                    ccell.tRNAs.begin() +
248                    ccell.tRNAs.size()/2);
249 vector<string> rtRNAs(ccell.tRNAs.begin() +
250                    ccell.tRNAs.size()/2,
251                    ccell.tRNAs.end());
252 vector<string> lrRNAs(ccell.rRNAs.begin(),
253                    ccell.rRNAs.begin() +
254                    ccell.rRNAs.size()/2);
255 vector<string> rrRNAs(ccell.rRNAs.begin() +
356                    ccell.rRNAs.size()/2,
257                    ccell.rRNAs.end());
258 vector<string> lPolypeptides1(ccell.Polypeptides1.begin(),
259                             ccell.Polypeptides1.begin() +
260                             ccell.Polypeptides1.size()/2);
261 vector<string> rPolypeptides1(ccell.Polypeptides1.begin() +
262                             ccell.Polypeptides1.size()/2,
263                             ccell.Polypeptides1.end());
264 vector<string> lPolypeptides2(ccell.Polypeptides2.begin(),
265                             ccell.Polypeptides2.begin() +
266                             ccell.Polypeptides2.size()/2);
267 vector<string> rPolypeptides2(ccell.Polypeptides2.begin() +
268                             ccell.Polypeptides2.size()/2,
269                             ccell.Polypeptides2.end());
270 vector<string> lPolypeptides3(ccell.Polypeptides3.begin(),
271                             ccell.Polypeptides3.begin() +
272                             ccell.Polypeptides3.size()/2);
273 vector<string> rPolypeptides3(ccell.Polypeptides3.begin() +
274                             ccell.Polypeptides3.size()/2,
275                             ccell.Polypeptides3.end());
276 vector<char> lMonomers1(ccell.Monomers1.begin(),
277                       ccell.Monomers1.begin() +
278                       ccell.Monomers1.size()/2);
279 vector<char> rMonomers1(ccell.Monomers1.begin() +
280                       ccell.Monomers1.size()/2,
281                       ccell.Monomers1.end());
282 vector<char> lMonomers2(ccell.Monomers2.begin(),
283                       ccell.Monomers2.begin() +
284                       ccell.Monomers2.size()/2);
285 vector<char> rMonomers2(ccell.Monomers2.begin() +

```

```

286         ccell.Monomers2.size()/2,
287         ccell.Monomers2.end());
288     vector<char> lMonomers3(ccell.Monomers3.begin(),
289         ccell.Monomers3.begin() +
290         ccell.Monomers3.size()/2);
291     vector<char> rMonomers3(ccell.Monomers3.begin() +
292         ccell.Monomers3.size()/2,
293         ccell.Monomers3.end());
294
295     cell lcell(lDNAs,
296         ltrNAs,
297         lrRNAs,
298         lPolypeptides1,
299         lPolypeptides2,
300         lPolypeptides3,
301         lMonomers1,
302         lMonomers2,
303         lMonomers3);
304     cell rcell(rDNAs,
305         rtRNAs,
306         rrRNAs,
307         rPolypeptides1,
308         rPolypeptides2,
309         rPolypeptides3,
310         rMonomers1,
311         rMonomers2,
312         rMonomers3);
313     Cells.push_back(lcell);
314     Cells.push_back(rcell);
315 }
316 }
317 };
318
319 int main()
320 {
321     string s0a, s0b;
322     s0a="CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC";
323     s0b="GCATGCGACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG";
324     vector<string> V0;
325     V0.push_back(s0a);
326     V0.push_back(s0b);
327     vector<string> V1(10, "GCAUGCGAC"), V2(10, "GCAUGCGAC");
328     vector<string> V3(10, "HIK"), V4(10, "LMN"), V5(10, "PQR");
329     vector<char> V6(10, 'X'), V7(10, 'Y'), V8(10, 'Z');
330     cell Cell(V0, V1, V2, V3, V4, V5, V6, V7, V8);
331
332     cp CellProgression(Cell);
333
334     return 0;
335 }

```

Here, C++ program has to set up an experiment for construction of an *in silico* genome multiplication network. This life pattern is roughly equal to the individual cell progression.

The genome replication by genome expression leads to genome multiplication. Progressive genome replication is usually associated with progressive cell propagation producing a cell progression: one cell, two cells, four cells, eight cells, and so on. The whole living world is only one cell progression which arose from one single primordial cell and has 3 or 4 billions years of uninterrupted history. It can be called general cell progression. The present-day biosphere is merely a tiny slice from it, a visible top of iceberg in ocean of time. The ancient part of this gigantic life pattern leaves very scarce traces. The genome multiplication is tightly associated with genome diversification. The mechanisms of genome diversification differ greatly ranging from the spontaneous sequence mutation to the highly regulated sequence transfer. The genome

diversification produces cell progressions each of which is specified by a particular individual genome and can be called individual cell progression. Respectively, the general cell progression can be considered as a growing composition of an increasing number of individual cell progressions.

Here, it is reasonable to begin with an experiment for construction of an *in silico* genome multiplication network with the simplest genome and the simplest life history. Environmental conditions too must be first extremely favourable: energy and all monomers for synthesis of DNA, RNA, and polypeptide molecules are in overflow. Lines from 197 to 317 contain definition of the new object type `cp` for *in silico* genome multiplication network.

Essentially, an individual cell progression is a binary tree. Respectively, the new object type `cp` has a constructor function which fills the special container `Cells` with cells in the so called in-level order so that each mother cell becomes replaced by its two daughter cells as soon it divides. The container `Cells` is an object of the object type `deque` defined in the file `deque` of the Standard C++ Library. The object type `deque` is specifically designed for construction of objects that can hold/contain a pool of other objects and for operation on them. Similar to the object type `vector`, the object type `deque` is defined as a template and can take other object types as parameters. Here, the object type `cell` is used as the template parameter. The object type `deque` is suited very well for construction of container to be filled in in-level order so that a binary tree will be produced. The body of the constructor function the new object type `cp` contains code for control structure specifying how to fill the container `Cells` with cells in in-level order. Note, the execution of the `while` loop is limited by the expression

```
Cells.size() <= Cells.max_size()
```

where `Cells.max_size()` can be replaced by other reasonable variable or number to reduce execution time.

Additionally, the constructor function of the new object type `cp` includes simple code for *in silico* instrumentation.

Lines from 321 to 329 contain code instructing the computer to construct in the memory 9 pools of *in silico* chemicals. Line 330 contains statement instructing the computer to construct in the memory an *in silico* genome expression network `Cell` of the type `cell` taking all 9 pools as arguments for its constructor functions. Line 332 contains statement instructing the computer to construct in the memory an *in silico* genome multiplication network `CellProgression` of the type `cp` taking object `Cell` as argument for its constructor functions.

Build Experiment4-1 and run the executable machine code.

The console window will display pictures documenting how the experiment for *in silico* genome multiplication proceeds. However, pictures flow too quickly. Close the console window, go to the line 205 of the file `Experiment4-1.cpp`, and replace the statement

```
Cells.size() <= Cells.max_size()
```

by the statement

```
Cells.size() <= 7
```

Build Experiment4-1 anew and run the executable machine code. The console window will display the first 7 pictures only:

```
Cell Number: 1

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC GCATGCG
ACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG CGTACGCTGCTTAA
GCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC GCATGCGACGAATTCGGACAC
ATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC
GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC
GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC

GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC
GCAUGCGAC GCAUGCGAC GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA
GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA

HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK

LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN

PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR

X X X X X X X X X X X X X X X X X X X X X X
Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y
Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z Z

Cell Number: 2

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC GCATGCG
ACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG CGTACGCTGCTTAA
GCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC GCATGCGACGAATTCGGACAC
ATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC
GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC
GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC

GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC
GCAUGCGAC GCAUGCGAC GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA
GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA

HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK

LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN

PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR

X X X X X X X X X X X X X X X X X X X X X X
Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y Y
```



GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC  
GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC

GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA  
GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA  
GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA

HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK

LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN

PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR

X X

Y Y

Z Z

Cell Number: 6

CGTACGCTGCTTAAAGCCTGTGATTTTGGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC GCATGCG  
ACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG CGTACGCTGCTTAA  
GCCTGTGATTTTGGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC GCATGCGACGAATTCGGACAC  
ATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC  
GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC  
GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC

GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA  
GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA  
GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA

HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK

LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN

PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR

X X

Y Y

Z Z

Cell Number: 7

CGTACGCTGCTTAAAGCCTGTGATTTTGGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC GCATGCG  
ACGAATTCGGACACATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG CGTACGCTGCTTAA  
GCCTGTGATTTTGGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC GCATGCGACGAATTCGGACAC  
ATAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC  
GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC  
GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC

GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA  
GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA  
GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA

HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK HIK

LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN LMN

PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR PQR

X X

Y Y

Z Z

Press any key . . .

Nikita Tirjatin. Laboratory for *in silico* life construction.

© 2010 Nikita Tirjatin