How to construct life *in silico*

Nikita Tirjatkin

Introduction

Life constructs itself using few simple rules of information processing. On the one hand, there is a set of rules determining how such basic chemical reactions as transcription, translation and catalysis proceed. On the other hand, there is a set of rules determining how these basic chemical reactions couple forming the information processing networks of growing complexity. These rules in fact represent the true algorithms for life construction. They also can be used for *in silico* life reconstruction.

Constructing itself, life arranges basic chemical reactions – transcription, translation, and catalysis – 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	Cell
1	Gene expression network	

To reconstruct life in any medium, these patterns of information processing must be reconstructed. Here, you will learn how to reconstruct these patterns *in silico*.

To establish you own laboratory for *in silico* life construction on your computer, install an Integrated Development Environment (IDE). IDE is needed for developing programs for *in silico* construction experiments. Once developed, the program then conducts the experiment every time it runs. Here, C++ is used as programming language and Microsoft Visual C++ 2008 Express Edition is used as IDE. 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 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, you laboratory is equipped for developing of C++ programs for *in silico* construction experiments.

Objectives

This script is broken if four parts. Each part contains descriptions of related experiments with following objectives:

Part 1	Learn how to construct *in silico* chemicals
Part 2	Learn how to construct *in silico* gene expression networks
Part 3	Learn how to construct *in silico* genome expression networks
Part 4	Learn how to construct *in silico* genome multiplication networks

Generally, a program for an *in silico* 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.

The script is aimed at absolute beginners without any knowledge and experience in programming and biology. The description of every experiment involves the following sections:

Entering source code
Here is the source code to be entered in the editor window.

Understanding source code
In this part, the source code is explained in detail.

Building and executing machine code
This part show pictures emerging in the console window and documenting how the experiment proceeds.

Boxes **Info** and **Note!** provide additional help.

Documentation

Documentation to this script contains files with source code and executable machine code for all experiments:

Part 1	Experiment1-1.cpp Experiment1-2.cpp	Experiment1-1.exe Experiment1-2.exe
Part 2	Experiment2-1.cpp Experiment2-2.cpp Experiment2-3.cpp Experiment2-4.cpp Experiment2-5.cpp	Experiment2-1.exe Experiment2-2.exe Experiment2-3.exe Experiment2-4.exe Experiment2-5.exe
Part 3	Experiment3-1.cpp	Experiment3-1.exe
Part 4	Experiment4-1.cpp Experiment4-2.cpp	Experiment4-1.exe Experiment4-2.exe

This documentation can be found in ZIP-directory at *www.nikita-tirjatkin.de*.

Part 1

Objective: Learn how to construct *in silico* chemicals.

Experiment1-1: Entering source code

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, Experiment1-1) in the field **Name**. The solution name (here, Experiment1-1) 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, Experiment1-1) in the field **Name** and click the button **Add**. In the window **Solution Explorer**, a new file (here, Experiment1-1.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 Experiment1-1.cpp:

```
 1  //Experiment 1-1
 2  /*Nikita Tirjatkin
 3    Laboratory for in silico life construction
 4    Januar 2011*/
 5
 6  #include <string>
 7  #include <iostream>
 8  using namespace std;
 9
10  int main()
11  {
12      char Monomer('A');
13      string Polymer("ABCDDCBAABCD");
14
15      cout << Monomer << "\t\t" << Polymer << "\n\n";
16
17      return 0;
18  }
```

Note! **Line numbers**
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**.

Experiment1-1: Understanding source code

Although extremely simple, the source code in the file Exeriment1-1.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` and has the name `std`.

Lines from 10 to 18 contain the definition of the function `main`. The function `main` is mandatory. Every C++ program must have the function `main`. Even if the C++ program contains other functions, the program execution begins by the function `main`, independently of its location within the source code. The name `main` of the function `main` is usually followed by a pair of parentheses `()` and then by the body of the function enclosed in braces `{}`.

Here, the body of the function `main` begins with two statements instructing the computer to construct two *in silico* objects in the memory.

Info

Object types
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.

Note!

Object types
The declaration statement 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 `Monomer` and an object of the type `string` with the name `Polymer` 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 function `main`).

Info

Object type `char`
`char` is a built-in object type. 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.

Info

Object type `string`
`string` is an object type defined in the file `string` of the Standard C++ Library. The object type `string` is specifically designed for construction of objects that can hold a sequence of characters and for operation on them.

Note!

Object type `string`
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.

Here, the computer is instructed to assign a character `A` to the object `Monomer` and a sequence of characters `ABCDDCBAABCD` to the object `Polymer`. The character to be assigned is always placed between single quotes `''`. The sequence of characters to be assigned is always placed between double quotes `“”`. Here, the assignment occurs implicitly by using constructor function of the object.

In experiments for *in silico* life construction, objects of type `char` and `string` are suited very well to represent *in silico* chemicals.

Info

Monomers and polymers
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 molecule is a chain of amino acids.

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.

Line 15 contains the statement instructing the computer to construct the standard output stream `cout` and to use it to transport images of objects `Monomer` and `Polymer` to the console window of the screen.

Info

Standard output stream `cout`
Standard output stream `cout` is an object of the object type `ostream` defined in the file `iostream` of the Standard C++ Library. To use it, C++ program must contain a directive shown in line 7.

In C++ program for console application, the standard output stream `cout` is the best device to be used as *in silico* instrumentation. 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.

Line 17 contains the statement instructing the computer to end C++ program for console window.

Note!

Whitespaces
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. In line 15, literals `"\t\t"` and `"\n\n"` stand for two tabs and two new lines respectively.

Experiment1-1: Building and executing machine code

After entering the source code to the file Experiment1-1.cpp, go to the menu **Build** and select **Build Experiment1-1**. In the window **Output**, the protocol with various status messages will appear. If all goes well, the protocol ends with the message:

```
Experiment1-1 - 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 objects `Monomer` and `Polymer`:

```
A                ABCDDCBAABCD

Press any key . . .
```

Note!

Console window
After program execution, the console window usually pauses automatically, exhibits the stream `Press any key . . .` , and waits for user action. 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 insert two additional instructions

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

just before the return instruction.

Experiment1-2: Entering source code

Create a new solution/project Experiment1-2 and add a new file Experiment1-2.cpp. Enter the following code within the file Experiment1-2.cpp:

```cpp
//Experiment 1-2
/*Nikita Tirjatkin
  Laboratory for in silico life construction
  Januar 2011*/

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

int main()
{
    string Polymer("ABCD");
    cout << "Picture 1:\n\n" << Polymer << "\n\n";

    Polymer.push_back('D');
    Polymer.push_back('C');
    Polymer.push_back('B');
    Polymer.push_back('A');
    cout << "Picture 2:\n\n" << Polymer << "\n\n";

    char Monomer1('A'), Monomer2('B'), Monomer3('C'), Monomer4('D');
    Polymer.push_back(Monomer1);
    Polymer.push_back(Monomer2);
    Polymer.push_back(Monomer3);
    Polymer.push_back(Monomer4);
    cout << "Picture 3:\n\n" << Polymer << "\n\n";

    return 0;
}
```

Experiment1-2: Understanding source code

Here, the body of the `main`–function begins with the statement instructing the computer to construct in the memory an *in silico* polymer `Polymer` consisting of four *in silico* monomers `A`, `B`, `C`, and `D`.

Lines from 15 to 18 contain instructions to append additional monomers `D`, `C`, `B`, and `A` to the polymer `Polymer` using function `push_back`.

Info

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!

Function
The parameters and arguments must have a clear correspondence.

Info

Function `push_back`
Function `push_back` is defined in the file `string` of the Standard C++ Library and can be used for all objects of the type `string`. It appends a single character to the string content and increases its size by one.

Note!

Function `push_back`
To call function `push_back`, its name must be connected to the name of the object (here, `Polymer`) by a dot sign `.`.

Lines from 21 to 25 contain statements instructing the computer first to construct in the memory four separate *in silico* monomers `A`, `B`, `C`, and `D` and then append them to the polymer `Polymer` using function `push_back`.

Lines 13, 19, and 26 contain code to produce pictures documenting the appearance and the subsequent grows (polymerization) of the object `Polymer`.

Experiment1-2: Building and executing machine code

Build Experiment1-2 and run the executable machine code.

The console window will appear and show pictures documenting the appearance and the subsequent grows (polymerization) of the the object `Polymer`:

```
Picture 1:

ABCD

Picture 2:

ABCDDCBA

Picture 3:

ABCDDCBAABCD

Press any key . . .
```

Part 2

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

Experiment2-1: Entering source code

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:

```
//Experiment2-1
/*Nikita Tirjatkin
  Laboratory for in silico life construction
  Januar 2011*/

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

class gena
{
  public:
    string RNA;
    gena(string Gene)
    {
        for(int i = 0; i < Gene.size(); i++)
        {
                 if(Gene[i] == 'A') RNA.push_back('U');
            else if(Gene[i] == 'C') RNA.push_back('G');
            else if(Gene[i] == 'G') RNA.push_back('C');
            else if(Gene[i] == 'T') RNA.push_back('A');
            else                    RNA.push_back('-');
        }
    }
};

int main()
{
    string Sa, Sb;
    Sa="CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC";
    Sb="GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG";
    cout << "Picture 1:\n\n"
         << Sa << endl
         << Sb << "\n\n";
    cout << "Picture 2:\n\n"
         << Sa << "\n\n\n"
         << Sb << "\n\n";

    gena G1(Sa.substr(0, 9));
    cout << "Picture 3:\n\n"
         << Sa << endl
         << G1.RNA << "\n\n"
         << Sb << "\n\n";
    cout << "Picture 4:\n\n"
         << Sa << endl
         << Sb << "\n\n"
         << G1.RNA << "\n\n";

    return 0;
}
```

Experiment2-1: Understanding source code

Here, C++ program has to set up an experiment for construction of an *in silico* gene expression network (GEN).

Info

Gene expression network
The directed sequence of basic chemical reactions

transcription -> translation -> catalysis

builds the most fundamental unit of information processing – gene expression network (GEN).

Note!

Gene expression network
In some GENs, this sequence can be restricted or extended. So, in many GENs, end products are polypeptides functioning always as substrate molecules and never as catalysts. There are also GENs whose end products are RNAs that never become translated into polypeptides, but function always at the level of RNA as substrate molecules. On the other hand, in many GENs, products of transcription or translation undergo post-transcriptional or post-translational processing respectively.

In order to be able to instruct computer to construct such objects as *in silico* GENs, some new object types must be defined.

Info

Definition of new object type
The form

```
class typename
{
    member object1;
    member object2;
    …
    member function1;
    member function2;
    …
};
```

is typically used to define new object type.

Note!

Definition of new object type
If the keyword `class` is used by the definition of the new object type, member objects and member functions have private access by default. Therefore, the keyword `public` must be used to make them publicly accessible (as shown in line 12).

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

Info

Transcription
During 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!

Transcription
The DNA molecule persists 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`. The function `gena` is not ordinary member function. It is a constructor function.

Info

Constructor and destructor
A constructor function is a special member function which will be automatically called when an object will be constructed. It is typically used to initialize the member objects to appropriate default values or to any desired values. A destructor function is the counterpart to constructor function. It will be automatically called when an object will be destroyed. It cleans up the memory before the object is removed.

Note!

Constructor and destructor
The constructor function must have the same name as the new object type. The destructor function too must have the same name as the new object type but with a tilde (~) in front of it. Both constructor and destructor cannot have any return type. Simple new object type does not need a destructor function because the memory for its object is usually cleaned up automatically.

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`). This control structure combines an iterative control structure with a conditional control structure.

Info

Iterative control structure `for`
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.

Here, the `statement1` of the iterative control structure declares a counter variable `i` of the object type `int` and assigns 0 to it.

Info

Object type `int`
`int` is a built-in object type. An object of the type `int` can hold only whole numbers.

Note!

Object type `int`
Object of the type `int` can have two different ranges, depending on whether it is signed or unsigned. If it is declared as signed (`signed int`), it can hold both negative and positive numbers and has a range of -2,147,483,648 to 2,147,483,647. If it is declared as unsigned (`unsigned int`), it can only hold positive values and has a range of 0 to 4,294,967,295. By default, object of the type `int` is signed.

So, the counter variable `i` is defined as an index that initially refers to the first element of the container (here, `string Gene`) and can be used to move between elements in the container and to access elements. Respectively, the `condition` checks wether `i` is reached the end of this container using function `size` which is defined in the file `string` of the Standard C++ Library and can be used for all objects of the type `string`. It returns a count of the number of characters in the string. The `statement2` uses the conditional control structure while the increment operator `++` in `statement3` advances `i` to the next element of the `string Gene` (`i++` is equal to `i = i + 1`).

Info

Conditional control structure `if`
Its form is

```
if(condition1)
    statement1
else if(condition2)
    statement2
else if(condition3)
    statement3
…
else
    statement
```

The program evaluates conditions to select the statement to be executed.

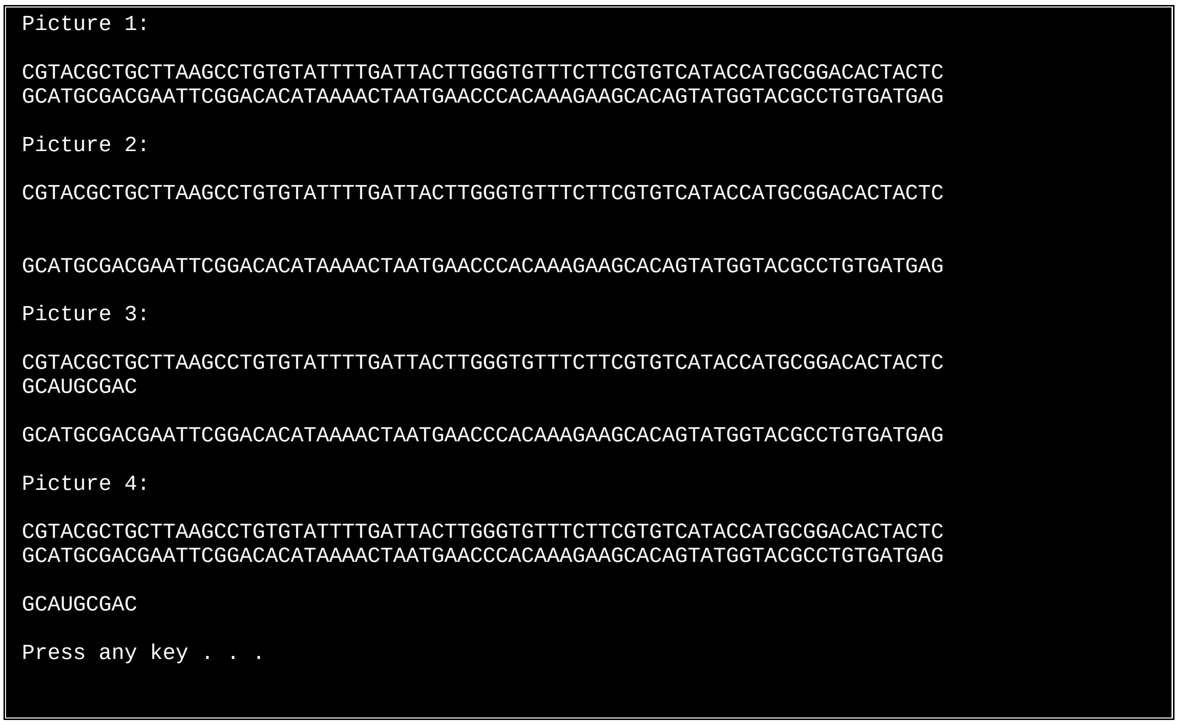
Here, the conditions of the conditional control structure are made dependent from the value of the characters for *in silico* deoxyribonucleotides in `Gene`. The access operator `[]` provides access to the deoxyribonucleotide at the position `i` in `Gene` while the statements uses function `push_back` to append corresponding ribinucleotide to the `RNA`. 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.

Experiment2-1: Building and executing machine code

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, transcription) proceeds:



```
Picture 1:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

Picture 2:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC


GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

Picture 3:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCAUGCGAC

GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

Picture 4:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

GCAUGCGAC

Press any key . . .
```

Experiment2-2: Entering source code

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:

```
//Experiment 2-2
/*Nikita Tirjatkin
  Laboratory for in silico life construction
  Januar 2011*/

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

class gena
{
  public:
    string RNA;
    gena(string Gene)
    {
        for(int i = 0; i < Gene.size(); i++)
        {
                 if(Gene[i] == 'A') RNA.push_back('U');
            else if(Gene[i] == 'C') RNA.push_back('G');
            else if(Gene[i] == 'G') RNA.push_back('C');
            else if(Gene[i] == 'T') RNA.push_back('A');
            else                    RNA.push_back('-');
        }
    }
};

class genb:public gena
{
  public:
    string Polypeptide;
    genb(string Gene):gena(Gene)
    {
        for(int i = 0; i < RNA.size(); i += 3)
        {
                 if(RNA.substr(i, 3) == "GCA" ||
                    RNA.substr(i, 3) == "GCC" ||
                    RNA.substr(i, 3) == "GCG" ||
                    RNA.substr(i, 3) == "GCU") Polypeptide.push_back('A');
            else if(RNA.substr(i, 3) == "UGC" ||
                    RNA.substr(i, 3) == "UGU") Polypeptide.push_back('C');
            else if(RNA.substr(i, 3) == "GAC" ||
                    RNA.substr(i, 3) == "GAU") Polypeptide.push_back('D');
            else if(RNA.substr(i, 3) == "GAA" ||
                    RNA.substr(i, 3) == "GAG") Polypeptide.push_back('E');
            else if(RNA.substr(i, 3) == "UUC" ||
                    RNA.substr(i, 3) == "UUU") Polypeptide.push_back('F');
            else if(RNA.substr(i, 3) == "GGA" ||
                    RNA.substr(i, 3) == "GGC" ||
                    RNA.substr(i, 3) == "GGG" ||
                    RNA.substr(i, 3) == "GGU") Polypeptide.push_back('G');
            else if(RNA.substr(i, 3) == "CAC" ||
                    RNA.substr(i, 3) == "CAU") Polypeptide.push_back('H');
            else if(RNA.substr(i, 3) == "AUA" ||
                    RNA.substr(i, 3) == "AUC" ||
                    RNA.substr(i, 3) == "AUU") Polypeptide.push_back('I');
            else if(RNA.substr(i, 3) == "AAA" ||
                    RNA.substr(i, 3) == "AAG") Polypeptide.push_back('K');
            else if(RNA.substr(i, 3) == "CUA" ||
                    RNA.substr(i, 3) == "CUC" ||
                    RNA.substr(i, 3) == "CUG" ||
                    RNA.substr(i, 3) == "CUU" ||
                    RNA.substr(i, 3) == "UUA" ||
                    RNA.substr(i, 3) == "UUG") Polypeptide.push_back('L');
            else if(RNA.substr(i, 3) == "AUG") Polypeptide.push_back('M');
            else if(RNA.substr(i, 3) == "AAC" ||
                    RNA.substr(i, 3) == "AAU") Polypeptide.push_back('N');
            else if(RNA.substr(i, 3) == "CCA" ||
                    RNA.substr(i, 3) == "CCC" ||
                    RNA.substr(i, 3) == "CCG" ||
                    RNA.substr(i, 3) == "CCU") Polypeptide.push_back('P');
            else if(RNA.substr(i, 3) == "CAA" ||
                    RNA.substr(i, 3) == "CAG") Polypeptide.push_back('Q');
            else if(RNA.substr(i, 3) == "AGA" ||
                    RNA.substr(i, 3) == "AGG" ||
                    RNA.substr(i, 3) == "CGA" ||
                    RNA.substr(i, 3) == "CGC" ||
                    RNA.substr(i, 3) == "CGG" ||
                    RNA.substr(i, 3) == "CGU") Polypeptide.push_back('R');
            else if(RNA.substr(i, 3) == "AGC" ||
                    RNA.substr(i, 3) == "AGU" ||
                    RNA.substr(i, 3) == "UCA" ||
                    RNA.substr(i, 3) == "UCC" ||
                    RNA.substr(i, 3) == "UCG" ||
                    RNA.substr(i, 3) == "UCU") Polypeptide.push_back('S');
            else if(RNA.substr(i, 3) == "ACA" ||
                    RNA.substr(i, 3) == "ACC" ||
                    RNA.substr(i, 3) == "ACG" ||
                    RNA.substr(i, 3) == "ACU") Polypeptide.push_back('T');
            else if(RNA.substr(i, 3) == "GUA" ||
                    RNA.substr(i, 3) == "GUC" ||
                    RNA.substr(i, 3) == "GUG" ||
                    RNA.substr(i, 3) == "GUU") Polypeptide.push_back('V');
            else if(RNA.substr(i, 3) == "UGG") Polypeptide.push_back('W');
            else if(RNA.substr(i, 3) == "UAC" ||
                    RNA.substr(i, 3) == "UAU") Polypeptide.push_back('Y');
            else                               Polypeptide.push_back('-');
        }
    }
};

int main()
{
    string Sa, Sb;
    Sa="CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC";
    Sb="GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG";
    cout << "Picture 1:\n\n"
         << Sa << endl
         << Sb << "\n\n";
    cout << "Picture 2:\n\n"
         << Sa << "\n\n\n"
         << Sb << "\n\n";

    genb G3(Sa.substr(18, 9));
    cout << "Picture 3:\n\n"
         << Sa << endl
         << "                  " << G3.RNA << "\n\n"
         << Sb << "\n\n";
    cout << "Picture 4:\n\n"
         << Sa << endl
         << Sb << "\n\n"
         << "                  " << G3.RNA << "\n\n";
    cout << "Picture 5:\n\n"
         << Sa << endl
         << Sb << "\n\n"
         << "                  " << G3.RNA << endl
         << "                       " << G3.Polypeptide << "\n\n";
    cout << "Picture 6:\n\n"
         << Sa << endl
         << Sb << "\n\n"
         << "                  " << G3.RNA << "\n\n"
         << "                       " << G3.Polypeptide << "\n\n";

    return 0;
}
```

Experiment2-2: Understanding source code

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

Info **Translation**
During 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! **Translation**
The RNA molecule persists translation without to be exhausted.

A new object type `genb` for *in silico* GENs involving transcription and translation can be defined as derived from the object type `gena` as shown in lines from 27 to 99.

Info **Derived object types**
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.

Note! **Derived object types**
To declare an object type as derived from another object type, a colon **:** is used.

Here, the derived object type `genb` aquires 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!

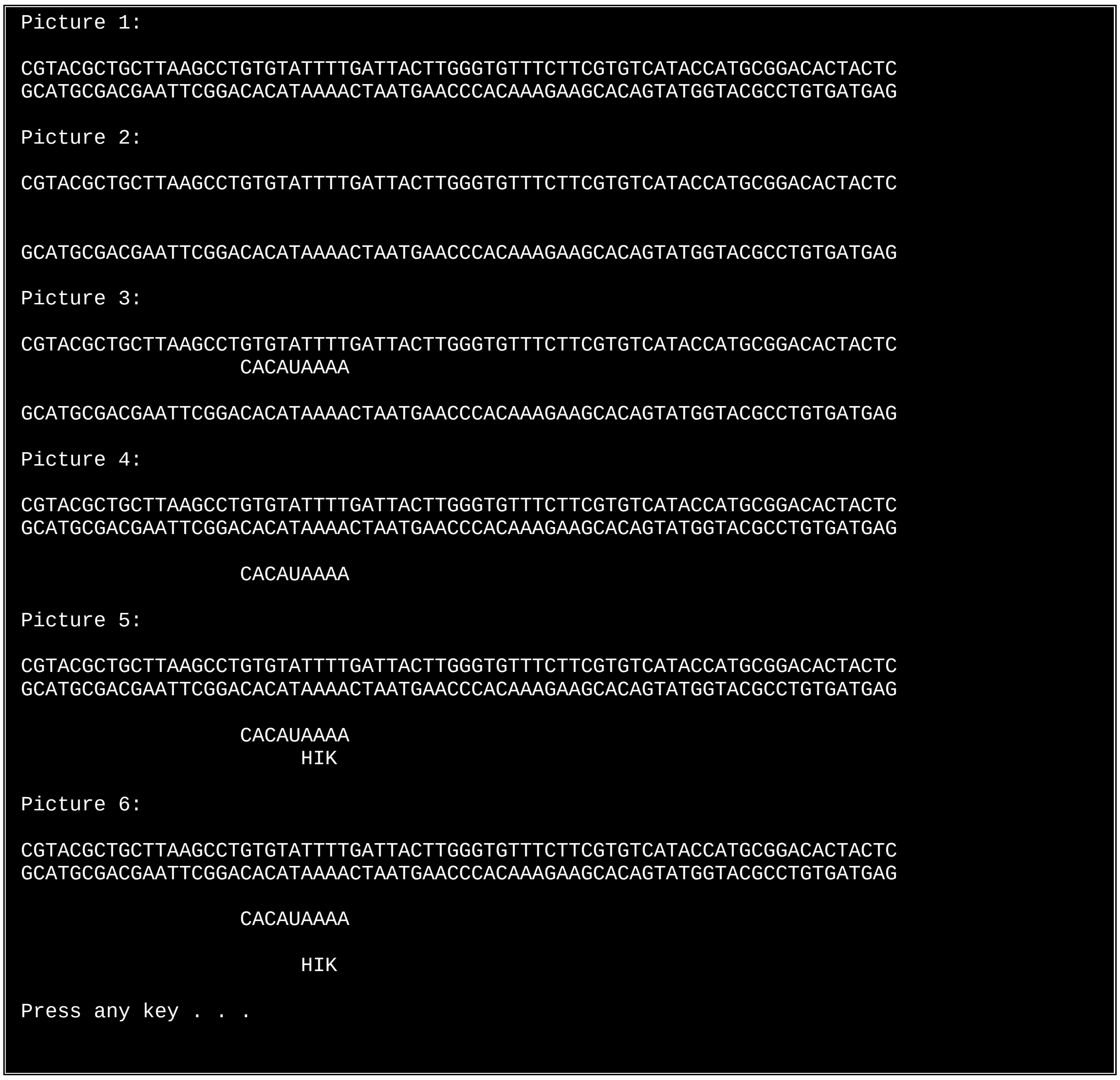
Logical operator OR
In conditional control structure, each condition makes many comparisons at once. They are connected by logical operator OR. The sign `||` is used as operator OR.

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.

Experiment2-2: Building and executing machine code

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, transcription + translation) proceeds:



```
Picture 1:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

Picture 2:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC


GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

Picture 3:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
                  CACAUAAAA

GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

Picture 4:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

                  CACAUAAAA

Picture 5:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

                  CACAUAAAA
                       HIK

Picture 6:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

                  CACAUAAAA

                       HIK

Press any key . . .
```

Experiment2-3: Entering source code

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:

```
//Experiment 2-3
/*Nikita Tirjatkin
  Laboratory for in silico life construction
  Januar 2011*/

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

class gena
{
  public:
    string RNA;
    gena(string Gene)
    {
        for(int i = 0; i < Gene.size(); i++)
        {
                 if(Gene[i] == 'A') RNA.push_back('U');
            else if(Gene[i] == 'C') RNA.push_back('G');
            else if(Gene[i] == 'G') RNA.push_back('C');
            else if(Gene[i] == 'T') RNA.push_back('A');
            else                    RNA.push_back('-');
        }
    }
};

class genb:public gena
{
  public:
    string Polypeptide;
    genb(string Gene):gena(Gene)
    {
        for(int i = 0; i < RNA.size(); i += 3)
        {
                 if(RNA.substr(i, 3) == "GCA" ||
                    RNA.substr(i, 3) == "GCC" ||
                    RNA.substr(i, 3) == "GCG" ||
                    RNA.substr(i, 3) == "GCU") Polypeptide.push_back('A');
            else if(RNA.substr(i, 3) == "UGC" ||
                    RNA.substr(i, 3) == "UGU") Polypeptide.push_back('C');
            else if(RNA.substr(i, 3) == "GAC" ||
                    RNA.substr(i, 3) == "GAU") Polypeptide.push_back('D');
            else if(RNA.substr(i, 3) == "GAA" ||
                    RNA.substr(i, 3) == "GAG") Polypeptide.push_back('E');
            else if(RNA.substr(i, 3) == "UUC" ||
                    RNA.substr(i, 3) == "UUU") Polypeptide.push_back('F');
            else if(RNA.substr(i, 3) == "GGA" ||
                    RNA.substr(i, 3) == "GGC" ||
                    RNA.substr(i, 3) == "GGG" ||
                    RNA.substr(i, 3) == "GGU") Polypeptide.push_back('G');
            else if(RNA.substr(i, 3) == "CAC" ||
                    RNA.substr(i, 3) == "CAU") Polypeptide.push_back('H');
            else if(RNA.substr(i, 3) == "AUA" ||
                    RNA.substr(i, 3) == "AUC" ||
                    RNA.substr(i, 3) == "AUU") Polypeptide.push_back('I');
            else if(RNA.substr(i, 3) == "AAA" ||
                    RNA.substr(i, 3) == "AAG") Polypeptide.push_back('K');
            else if(RNA.substr(i, 3) == "CUA" ||
                    RNA.substr(i, 3) == "CUC" ||
                    RNA.substr(i, 3) == "CUG" ||
                    RNA.substr(i, 3) == "CUU" ||
                    RNA.substr(i, 3) == "UUA" ||
                    RNA.substr(i, 3) == "UUG") Polypeptide.push_back('L');
            else if(RNA.substr(i, 3) == "AUG") Polypeptide.push_back('M');
            else if(RNA.substr(i, 3) == "AAC" ||
                    RNA.substr(i, 3) == "AAU") Polypeptide.push_back('N');
            else if(RNA.substr(i, 3) == "CCA" ||
                    RNA.substr(i, 3) == "CCC" ||
                    RNA.substr(i, 3) == "CCG" ||
                    RNA.substr(i, 3) == "CCU") Polypeptide.push_back('P');
            else if(RNA.substr(i, 3) == "CAA" ||
                    RNA.substr(i, 3) == "CAG") Polypeptide.push_back('Q');
            else if(RNA.substr(i, 3) == "AGA" ||
                    RNA.substr(i, 3) == "AGG" ||
                    RNA.substr(i, 3) == "CGA" ||
                    RNA.substr(i, 3) == "CGC" ||
                    RNA.substr(i, 3) == "CGG" ||
                    RNA.substr(i, 3) == "CGU") Polypeptide.push_back('R');
            else if(RNA.substr(i, 3) == "AGC" ||
                    RNA.substr(i, 3) == "AGU" ||
                    RNA.substr(i, 3) == "UCA" ||
                    RNA.substr(i, 3) == "UCC" ||
                    RNA.substr(i, 3) == "UCG" ||
                    RNA.substr(i, 3) == "UCU") Polypeptide.push_back('S');
            else if(RNA.substr(i, 3) == "ACA" ||
                    RNA.substr(i, 3) == "ACC" ||
                    RNA.substr(i, 3) == "ACG" ||
                    RNA.substr(i, 3) == "ACU") Polypeptide.push_back('T');
            else if(RNA.substr(i, 3) == "GUA" ||
                    RNA.substr(i, 3) == "GUC" ||
                    RNA.substr(i, 3) == "GUG" ||
                    RNA.substr(i, 3) == "GUU") Polypeptide.push_back('V');
            else if(RNA.substr(i, 3) == "UGG") Polypeptide.push_back('W');
            else if(RNA.substr(i, 3) == "UAC" ||
                    RNA.substr(i, 3) == "UAU") Polypeptide.push_back('Y');
            else                               Polypeptide.push_back('-');
        }
    }
};

class genca:public genb
{
  public:
    char Monomer;
    genca(string Gene, char A, char B, char C):genb(Gene)
    {
        Monomer = C;
    }
};

int main()
{
    string Sa, Sb;
    Sa="CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC";
    Sb="GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG";
    cout << "Picture 1:\n\n"
         << Sa << endl
         << Sb << "\n\n";
    cout << "Picture 2:" << "\n\n"
         << Sa << "\n\n\n"
         << Sb << "\n\n";

    genca G6(Sa.substr(45, 9), 'U', 'V', 'X');
    cout << "Picture 3:\n\n"
         << Sa << endl
         << "                                             " << G6.RNA << "\n\n"
         << Sb << "\n\n";
    cout << "Picture 4:\n\n"
         << Sa << endl
         << Sb << "\n\n"
         << "                                             " << G6.RNA << "\n\n";
    cout << "Picture 5:\n\n"
         << Sa << endl
         << Sb << "\n\n"
         << "                                             " << G6.RNA << endl
         << "                                                  " << G6.Polypeptide
         << "\n\n";
    cout << "Picture 6:\n\n"
         << Sa << endl
         << Sb << "\n\n"
         << "                                             " << G6.RNA << "\n\n"
         << "                                                  " << G6.Polypeptide
         << "\n\n";
    cout << "Picture 7:\n\n"
         << Sa << endl
         << Sb << "\n\n"
         << "                                             " << G6.RNA << "\n\n"
         << "                                                  " << G6.Polypeptide
         << endl
         << "                                                  U V" << "\n\n";
    cout << "Picture 8:\n\n"
         << Sa << endl
         << Sb << "\n\n"
         << "                                             " << G6.RNA << "\n\n"
         << "                                                  " << G6.Polypeptide
         << endl
         << "                                                   " << G6.Monomer
         << "\n\n";
    cout << "Picture 9:\n\n"
         << Sa << endl
         << Sb << "\n\n"
         << "                                             " << G6.RNA << "\n\n"
         << "                                                  " << G6.Polypeptide
         << "\n\n"
         << "                                                   " << G6.Monomer
         << "\n\n";

        return 0;
}
```

Experiment2-3: Understanding source code

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

Info	**Catalysis** During catalysis, the catalyst serves as a template for the reaction that otherwise could occur too slowly for live. 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.
Note!	**Catalysis** The catalyst persists catalysis without to be exhausted.

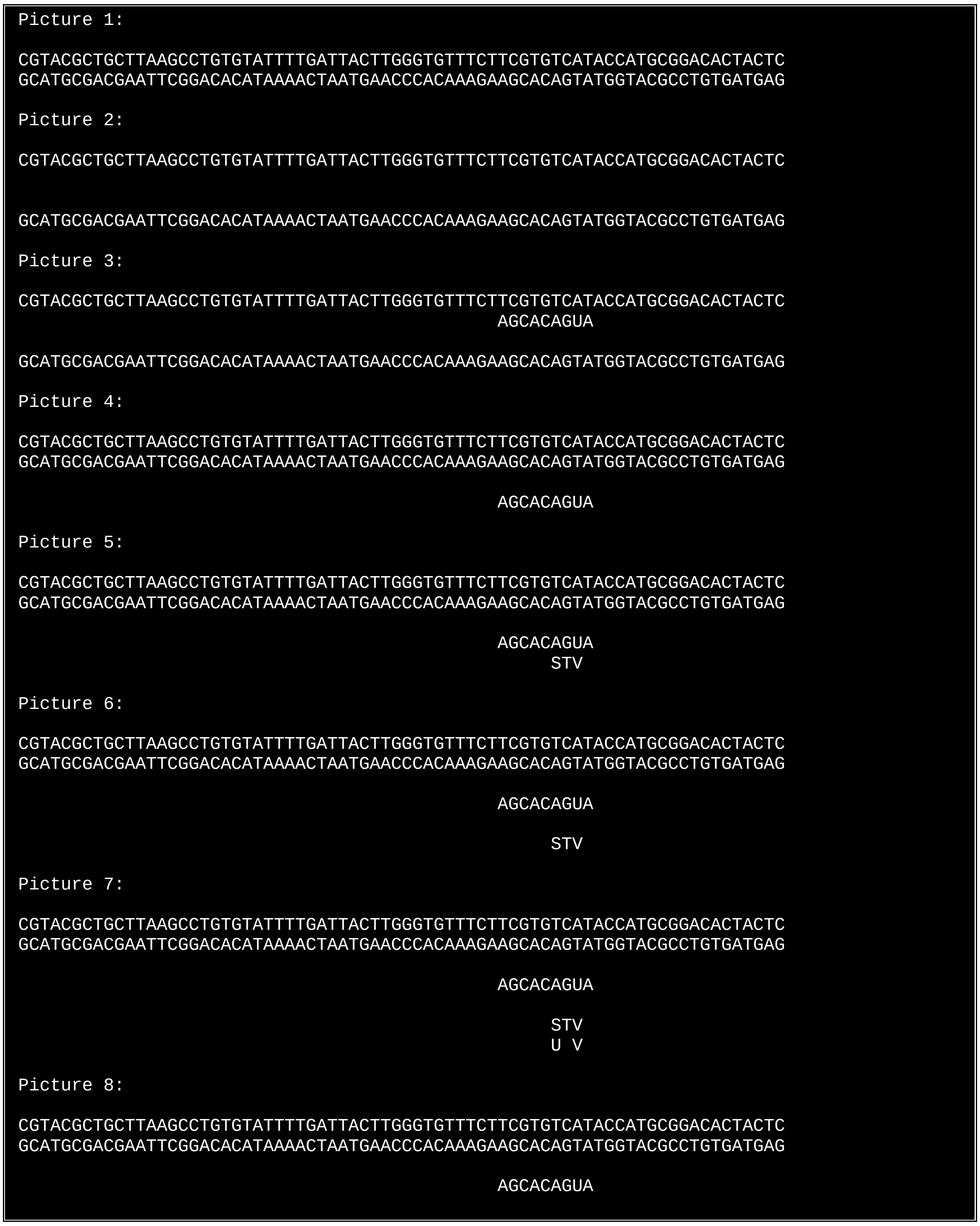
A new object type for *in silico* GENs involving transcription, translation, and catalysis can be defined as derived from the object type `genb`. Lines from 101 to 109 contain definition of the object type `genca` for *in silico* GENs involving catalysis of the reaction A + B = C, where A, B, and C are monomers. Since the object type `genb` itself derives from the object type `gena`, the object type `genca` aquires 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.

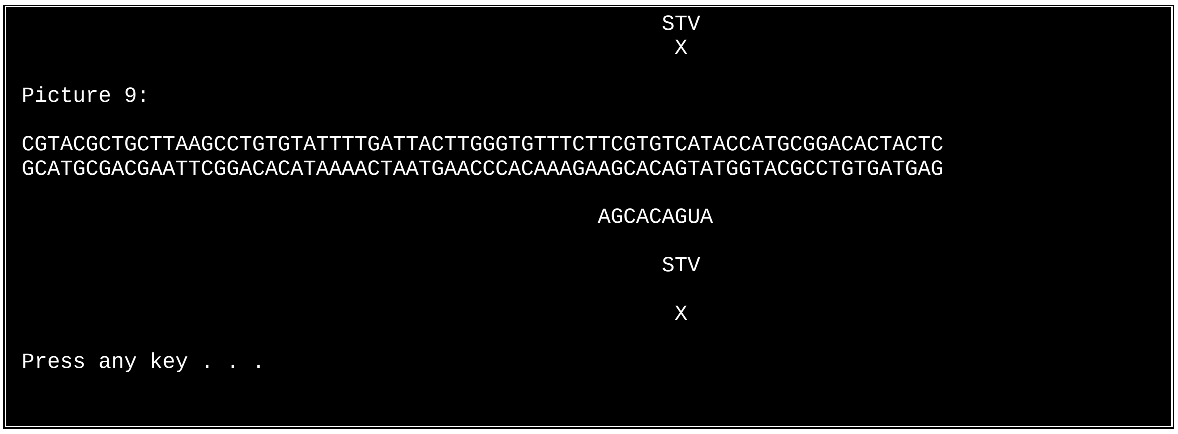
Experiment2-3: Building and executing machine code

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, transcription + translation + catalysis, where the reaction of type A + B = C is catalysed) proceeds:



```
Picture 1:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

Picture 2:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC


GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

Picture 3:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
                                             AGCACAGUA

GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

Picture 4:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

                                             AGCACAGUA

Picture 5:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

                                             AGCACAGUA
                                                  STV

Picture 6:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

                                             AGCACAGUA

                                                  STV

Picture 7:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

                                             AGCACAGUA

                                                  STV
                                                  U V

Picture 8:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

                                             AGCACAGUA
```



```
                                                  STV
                                                   X

Picture 9:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

                                             AGCACAGUA

                                                  STV

                                                   X

Press any key . . .
```

Experiment2-4: Entering source code

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:

```cpp
//Experiment 2-4
/*Nikita Tirjatkin
  Laboratory for in silico life construction
  Januar 2011*/

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

class gena
{
  public:
    string RNA;
    gena(string Gene)
    {
        for(int i = 0; i < Gene.size(); i++)
        {
                 if(Gene[i] == 'A') RNA.push_back('U');
            else if(Gene[i] == 'C') RNA.push_back('G');
            else if(Gene[i] == 'G') RNA.push_back('C');
            else if(Gene[i] == 'T') RNA.push_back('A');
            else                    RNA.push_back('-');
        }
    }
};

class genb:public gena
{
  public:
    string Polypeptide;
    genb(string Gene):gena(Gene)
    {
        for(int i = 0; i < RNA.size(); i += 3)
        {
                 if(RNA.substr(i, 3) == "GCA" ||
                    RNA.substr(i, 3) == "GCC" ||
                    RNA.substr(i, 3) == "GCG" ||
                    RNA.substr(i, 3) == "GCU") Polypeptide.push_back('A');
            else if(RNA.substr(i, 3) == "UGC" ||
                    RNA.substr(i, 3) == "UGU") Polypeptide.push_back('C');
            else if(RNA.substr(i, 3) == "GAC" ||
                    RNA.substr(i, 3) == "GAU") Polypeptide.push_back('D');
            else if(RNA.substr(i, 3) == "GAA" ||
                    RNA.substr(i, 3) == "GAG") Polypeptide.push_back('E');
            else if(RNA.substr(i, 3) == "UUC" ||
                    RNA.substr(i, 3) == "UUU") Polypeptide.push_back('F');
            else if(RNA.substr(i, 3) == "GGA" ||
                    RNA.substr(i, 3) == "GGC" ||
                    RNA.substr(i, 3) == "GGG" ||
                    RNA.substr(i, 3) == "GGU") Polypeptide.push_back('G');
            else if(RNA.substr(i, 3) == "CAC" ||
                    RNA.substr(i, 3) == "CAU") Polypeptide.push_back('H');
            else if(RNA.substr(i, 3) == "AUA" ||
                    RNA.substr(i, 3) == "AUC" ||
                    RNA.substr(i, 3) == "AUU") Polypeptide.push_back('I');
            else if(RNA.substr(i, 3) == "AAA" ||
                    RNA.substr(i, 3) == "AAG") Polypeptide.push_back('K');
            else if(RNA.substr(i, 3) == "CUA" ||
                    RNA.substr(i, 3) == "CUC" ||
                    RNA.substr(i, 3) == "CUG" ||
                    RNA.substr(i, 3) == "CUU" ||
                    RNA.substr(i, 3) == "UUA" ||
                    RNA.substr(i, 3) == "UUG") Polypeptide.push_back('L');
            else if(RNA.substr(i, 3) == "AUG") Polypeptide.push_back('M');
            else if(RNA.substr(i, 3) == "AAC" ||
                    RNA.substr(i, 3) == "AAU") Polypeptide.push_back('N');
            else if(RNA.substr(i, 3) == "CCA" ||
                    RNA.substr(i, 3) == "CCC" ||
                    RNA.substr(i, 3) == "CCG" ||
                    RNA.substr(i, 3) == "CCU") Polypeptide.push_back('P');
            else if(RNA.substr(i, 3) == "CAA" ||
                    RNA.substr(i, 3) == "CAG") Polypeptide.push_back('Q');
            else if(RNA.substr(i, 3) == "AGA" ||
                    RNA.substr(i, 3) == "AGG" ||
                    RNA.substr(i, 3) == "CGA" ||
                    RNA.substr(i, 3) == "CGC" ||
                    RNA.substr(i, 3) == "CGG" ||
                    RNA.substr(i, 3) == "CGU") Polypeptide.push_back('R');
            else if(RNA.substr(i, 3) == "AGC" ||
                    RNA.substr(i, 3) == "AGU" ||
                    RNA.substr(i, 3) == "UCA" ||
                    RNA.substr(i, 3) == "UCC" ||
                    RNA.substr(i, 3) == "UCG" ||
                    RNA.substr(i, 3) == "UCU") Polypeptide.push_back('S');
            else if(RNA.substr(i, 3) == "ACA" ||
                    RNA.substr(i, 3) == "ACC" ||
                    RNA.substr(i, 3) == "ACG" ||
                    RNA.substr(i, 3) == "ACU") Polypeptide.push_back('T');
            else if(RNA.substr(i, 3) == "GUA" ||
                    RNA.substr(i, 3) == "GUC" ||
                    RNA.substr(i, 3) == "GUG" ||
                    RNA.substr(i, 3) == "GUU") Polypeptide.push_back('V');
            else if(RNA.substr(i, 3) == "UGG") Polypeptide.push_back('W');
            else if(RNA.substr(i, 3) == "UAC" ||
                    RNA.substr(i, 3) == "UAU") Polypeptide.push_back('Y');
            else                               Polypeptide.push_back('-');
        }
    }
};

class gencb:public genb
{
  public:
    char Monomer1;
    char Monomer2;
    gencb(string Gene, char A, char B, char C):genb(Gene)
    {
        Monomer1 = B;
        Monomer2 = C;
    }
};

int main()
{
    string Sa, Sb;
    Sa="CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC";
    Sb="GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG";
    cout << "Picture 1:\n\n"
         << Sa << endl
         << Sb << "\n\n";
    cout << "Picture 2:\n\n"
         << Sa << "\n\n\n"
         << Sb << "\n\n";

    gencb G7(Sa.substr(54, 9), 'W', 'Y', 'Z');
    cout << "Picture 3:\n\n"
         << Sa << endl
         << "                                                      " << G7.RNA
         << "\n\n" << Sb << "\n\n";
    cout << "Picture 4:\n\n"
         << Sa << endl
         << Sb << "\n\n"
         << "                                                      " << G7.RNA
         << "\n\n";
    cout << "Picture 5:\n\n"
         << Sa << endl
         << Sb << "\n\n"
         << "                                                      " << G7.RNA
         << endl
         << "                                                           "
         << G7.Polypeptide << "\n\n";
    cout << "Picture 6:\n\n"
         << Sa << endl
         << Sb << "\n\n"
         << "                                                       " << G7.RNA
         << "\n\n"
         << "                                                            "
         << G7.Polypeptide << "\n\n";
    cout << "Picture 7:\n\n"
         << Sa << endl
         << Sb << "\n\n"
         << "                                                       " << G7.RNA
         << "\n\n"
         << "                                                            "
         << G7.Polypeptide << endl
         << "                                                             W"
         << "\n\n";
    cout << "Picture 8:\n\n"
         << Sa << endl
         << Sb << "\n\n"
         << "                                                       " << G7.RNA
         << "\n\n"
         << "                                                            "
         << G7.Polypeptide << endl
         << "                                                            "
         << G7.Monomer1 << " " << G7.Monomer2 << "\n\n";
    cout << "Picture 9:\n\n"
         << Sa << endl
         << Sb << "\n\n"
         << "                                                       " << G7.RNA
         << "\n\n"
         << "                                                            "
         << G7.Polypeptide << "\n\n"
         << "                                                            "
         << G7.Monomer1 << " " << G7.Monomer2 << "\n\n";

       return 0;
}
```

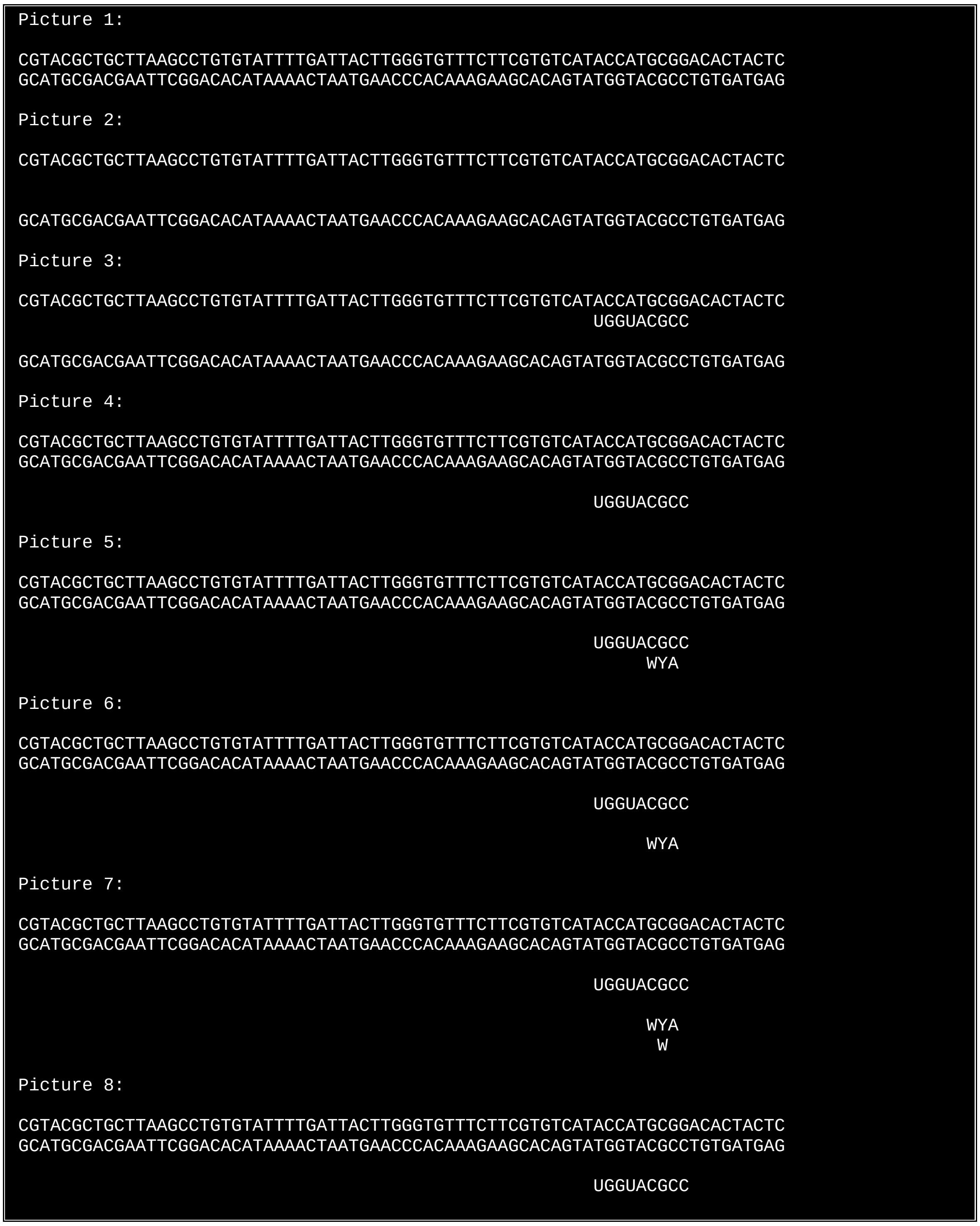
Experiment2-4: Understanding source code

Here, C++ program has to set up an experiment for construction of an *in silico* gene expression network involving transcription, 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.

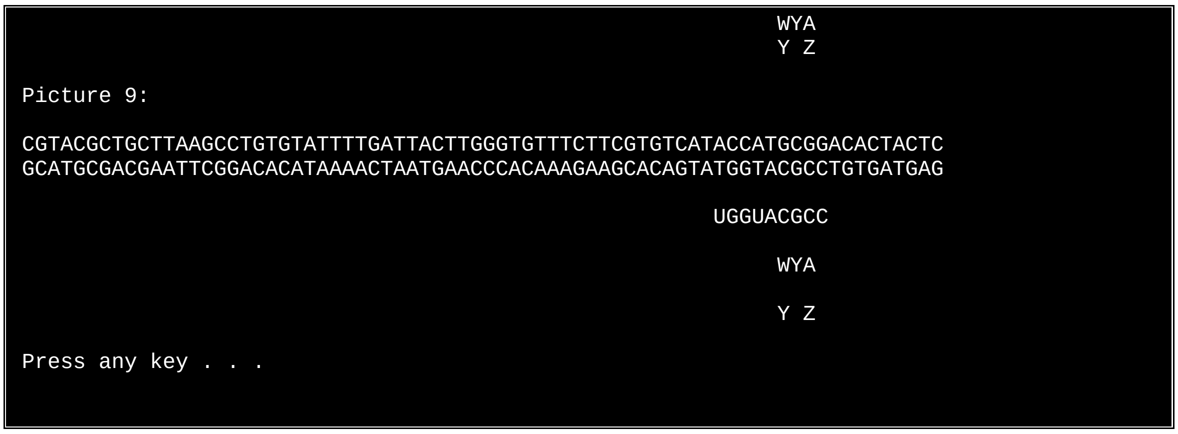
Experiment2-4: Building and executing machine code

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, transcription + translation + catalysis, where the reaction of type A = B + C is catalysed) proceeds:



```
Picture 1:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

Picture 2:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC


GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

Picture 3:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
                                                      UGGUACGCC

GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

Picture 4:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

                                                      UGGUACGCC

Picture 5:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

                                                      UGGUACGCC
                                                           WYA

Picture 6:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

                                                      UGGUACGCC

                                                           WYA

Picture 7:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

                                                      UGGUACGCC

                                                           WYA
                                                            W

Picture 8:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

                                                      UGGUACGCC
```



```
                                                           WYA
                                                           Y Z

Picture 9:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

                                                      UGGUACGCC

                                                           WYA

                                                           Y Z

Press any key . . .
```

Experiment2-5: Entering source code

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:

```
//Experiment 2-5
/*Nikita Tirjatkin
  Laboratory for in silico life construction
  Januar 2011*/

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

class gena
{
  public:
    string RNA;
    gena(string Gene)
    {
        for(int i = 0; i < Gene.size(); i++)
        {
                 if(Gene[i] == 'A') RNA.push_back('U');
            else if(Gene[i] == 'C') RNA.push_back('G');
            else if(Gene[i] == 'G') RNA.push_back('C');
            else if(Gene[i] == 'T') RNA.push_back('A');
            else                    RNA.push_back('-');
        }
    }
};

class genb:public gena
{
  public:
    string Polypeptide;
    genb(string Gene):gena(Gene)
    {
        for(int i = 0; i < RNA.size(); i += 3)
        {
                 if(RNA.substr(i, 3) == "GCA" ||
                    RNA.substr(i, 3) == "GCC" ||
                    RNA.substr(i, 3) == "GCG" ||
                    RNA.substr(i, 3) == "GCU") Polypeptide.push_back('A');
            else if(RNA.substr(i, 3) == "UGC" ||
                    RNA.substr(i, 3) == "UGU") Polypeptide.push_back('C');
            else if(RNA.substr(i, 3) == "GAC" ||
                    RNA.substr(i, 3) == "GAU") Polypeptide.push_back('D');
            else if(RNA.substr(i, 3) == "GAA" ||
                    RNA.substr(i, 3) == "GAG") Polypeptide.push_back('E');
            else if(RNA.substr(i, 3) == "UUC" ||
                    RNA.substr(i, 3) == "UUU") Polypeptide.push_back('F');
            else if(RNA.substr(i, 3) == "GGA" ||
                    RNA.substr(i, 3) == "GGC" ||
                    RNA.substr(i, 3) == "GGG" ||
                    RNA.substr(i, 3) == "GGU") Polypeptide.push_back('G');
            else if(RNA.substr(i, 3) == "CAC" ||
                    RNA.substr(i, 3) == "CAU") Polypeptide.push_back('H');
            else if(RNA.substr(i, 3) == "AUA" ||
                    RNA.substr(i, 3) == "AUC" ||
                    RNA.substr(i, 3) == "AUU") Polypeptide.push_back('I');
            else if(RNA.substr(i, 3) == "AAA" ||
                    RNA.substr(i, 3) == "AAG") Polypeptide.push_back('K');
            else if(RNA.substr(i, 3) == "CUA" ||
                    RNA.substr(i, 3) == "CUC" ||
                    RNA.substr(i, 3) == "CUG" ||
                    RNA.substr(i, 3) == "CUU" ||
                    RNA.substr(i, 3) == "UUA" ||
                    RNA.substr(i, 3) == "UUG") Polypeptide.push_back('L');
            else if(RNA.substr(i, 3) == "AUG") Polypeptide.push_back('M');
            else if(RNA.substr(i, 3) == "AAC" ||
                    RNA.substr(i, 3) == "AAU") Polypeptide.push_back('N');
            else if(RNA.substr(i, 3) == "CCA" ||
                    RNA.substr(i, 3) == "CCC" ||
                    RNA.substr(i, 3) == "CCG" ||
                    RNA.substr(i, 3) == "CCU") Polypeptide.push_back('P');
            else if(RNA.substr(i, 3) == "CAA" ||
                    RNA.substr(i, 3) == "CAG") Polypeptide.push_back('Q');
            else if(RNA.substr(i, 3) == "AGA" ||
                    RNA.substr(i, 3) == "AGG" ||
                    RNA.substr(i, 3) == "CGA" ||
                    RNA.substr(i, 3) == "CGC" ||
                    RNA.substr(i, 3) == "CGG" ||
                    RNA.substr(i, 3) == "CGU") Polypeptide.push_back('R');
            else if(RNA.substr(i, 3) == "AGC" ||
                    RNA.substr(i, 3) == "AGU" ||
                    RNA.substr(i, 3) == "UCA" ||
                    RNA.substr(i, 3) == "UCC" ||
                    RNA.substr(i, 3) == "UCG" ||
                    RNA.substr(i, 3) == "UCU") Polypeptide.push_back('S');
            else if(RNA.substr(i, 3) == "ACA" ||
                    RNA.substr(i, 3) == "ACC" ||
                    RNA.substr(i, 3) == "ACG" ||
                    RNA.substr(i, 3) == "ACU") Polypeptide.push_back('T');
            else if(RNA.substr(i, 3) == "GUA" ||
                    RNA.substr(i, 3) == "GUC" ||
                    RNA.substr(i, 3) == "GUG" ||
                    RNA.substr(i, 3) == "GUU") Polypeptide.push_back('V');
            else if(RNA.substr(i, 3) == "UGG") Polypeptide.push_back('W');
            else if(RNA.substr(i, 3) == "UAC" ||
                    RNA.substr(i, 3) == "UAU") Polypeptide.push_back('Y');
            else                               Polypeptide.push_back('-');
        }
    }
};

class gencz:public genb
{
  public:
    string Replica;
    gencz(string Gene, string Template):genb(Gene)
    {
        for(int i = 0; i < Template.size(); i++)
        {
                 if(Template[i] == 'A') Replica.push_back('T');
            else if(Template[i] == 'C') Replica.push_back('G');
            else if(Template[i] == 'G') Replica.push_back('C');
            else if(Template[i] == 'T') Replica.push_back('A');
            else                        Replica.push_back('-');
        }
    }
};

int main()
{
    string Sa, Sb;
    Sa="CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC";
    Sb="GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG";
    cout << "Picture 1:\n\n"
         << Sa << endl
         << Sb << "\n\n";
    cout << "Picture 2:\n\n"
         << Sa << "\n\n\n"
         << Sb << "\n\n";

    gencz G8a(Sa.substr(63, 9), Sa);
    gencz G8b(Sa.substr(63, 9), Sb);
    cout << "Picture 3:\n\n"
         << Sa << endl
         << "                                                               "
         << G8a.RNA << "\n\n"
         << Sb << "\n\n";
    cout << "Picture 4:\n\n"
         << Sa << endl
         << Sb << "\n\n"
         << "                                                               "
         << G8a.RNA << "\n\n";
    cout << "Picture 5:\n\n"
         << Sa << endl
         << Sb << "\n\n"
         << "                                                                "
         << G8a.RNA << endl
         << "                                                                      "
         << G8a.Polypeptide << "\n\n";
    cout << "Picture 6:\n\n"
         << Sa << endl
         << Sb << "\n\n"
         << "                                                                "
         << G8a.RNA << "\n\n"
         << "                                                                      "
         << G8a.Polypeptide << "\n\n";
    cout << "Picture 7:\n\n"
         << Sa << endl
         << G8a.Polypeptide << "\n\n"
         << "                                                                      "
         << G8b.Polypeptide << endl
         << Sb << "\n\n";
    cout << "Picture 8:\n\n"
         << Sa << endl
         << G8a.Replica << "\n\n"
         << G8b.Replica << endl
         << Sb << "\n\n";

    return 0;
}
```

Experiment2-5: Understanding source code

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

Info

Replication
During 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 replication is complete, each original DNA molecule is replaced by its two identical copies.

Note!

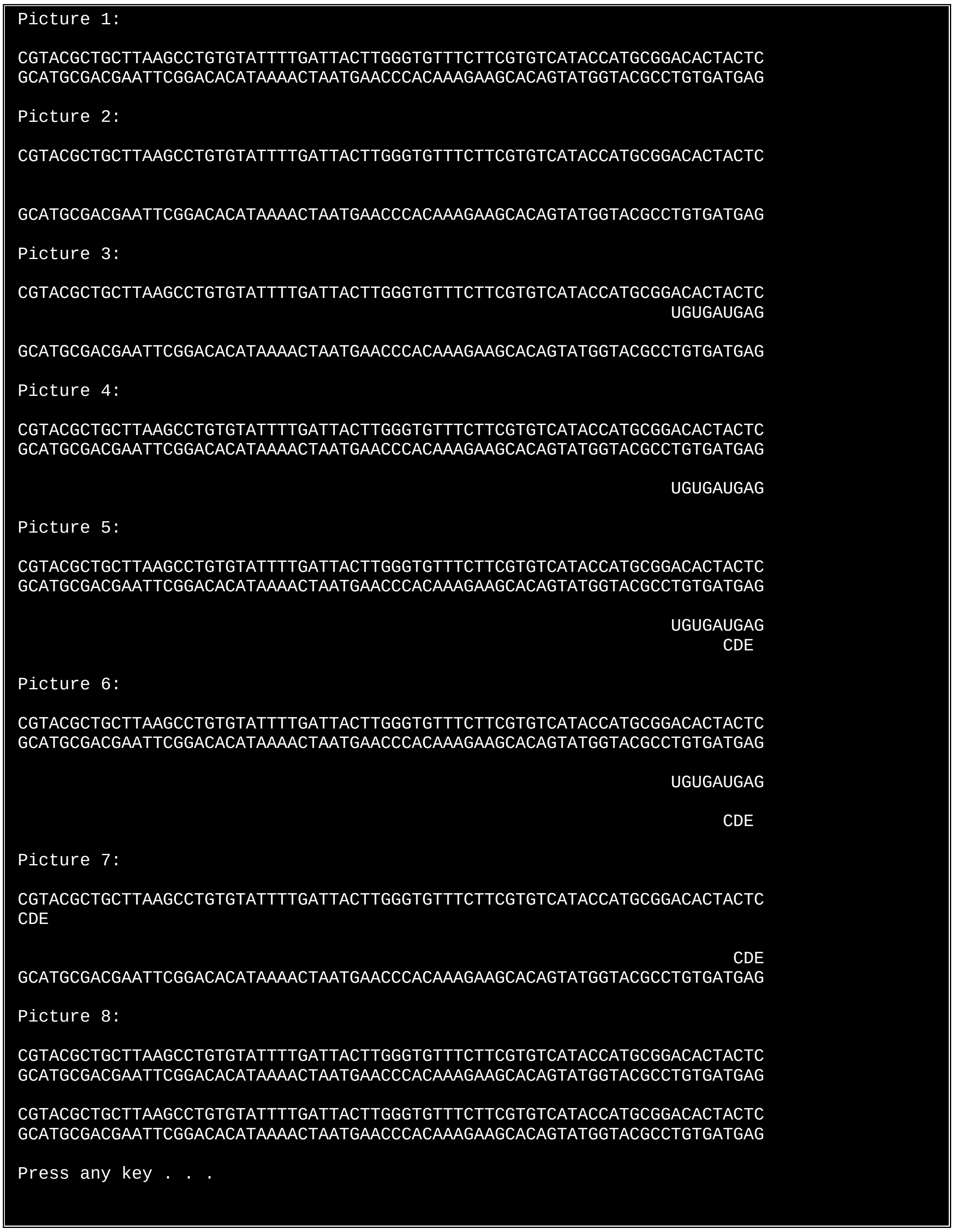
Replication
The DNA molecule strands persist replication without to be exhausted.

Here, C++ program has to set up an experiment for construction of an *in silico* gene expression network involving transcription, translation, and catalysis, where the 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 template (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.

Experiment2-5: Building and executing machine code

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, transcription + translation + catalysis, where the replication is catalysed) proceeds:



```
Picture 1:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

Picture 2:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC


GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

Picture 3:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
                                                               UGUGAUGAG

GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

Picture 4:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

                                                               UGUGAUGAG

Picture 5:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

                                                               UGUGAUGAG
                                                                    CDE

Picture 6:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

                                                               UGUGAUGAG

                                                                    CDE

Picture 7:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
CDE

                                                                     CDE
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

Picture 8:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

Press any key . . .
```

Part 3

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

Experiment3-1: Entering source code

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:

```
//Experiment 3-1
/*Nikita Tirjatkin
  Laboratory for in silico life construction
  Januar 2011*/

#include <string>
#include <vector>
#include <iostream>
using namespace std;

class gena
{
  public:
    string RNA;
    gena(string Gene)
    {
        for(int i = 0; i < Gene.size(); i++)
        {
                 if(Gene[i] == 'A') RNA.push_back('U');
            else if(Gene[i] == 'C') RNA.push_back('G');
            else if(Gene[i] == 'G') RNA.push_back('C');
            else if(Gene[i] == 'T') RNA.push_back('A');
            else                    RNA.push_back('-');
        }
    }
};

class genb:public gena
{
  public:
    string Polypeptide;
    genb(string Gene):gena(Gene)
    {
        for(int i = 0; i < RNA.size(); i += 3)
        {
                 if(RNA.substr(i, 3) == "GCA" ||
                    RNA.substr(i, 3) == "GCC" ||
                    RNA.substr(i, 3) == "GCG" ||
                    RNA.substr(i, 3) == "GCU") Polypeptide.push_back('A');
            else if(RNA.substr(i, 3) == "UGC" ||
                    RNA.substr(i, 3) == "UGU") Polypeptide.push_back('C');
            else if(RNA.substr(i, 3) == "GAC" ||
                    RNA.substr(i, 3) == "GAU") Polypeptide.push_back('D');
            else if(RNA.substr(i, 3) == "GAA" ||
                    RNA.substr(i, 3) == "GAG") Polypeptide.push_back('E');
            else if(RNA.substr(i, 3) == "UUC" ||
                    RNA.substr(i, 3) == "UUU") Polypeptide.push_back('F');
            else if(RNA.substr(i, 3) == "GGA" ||
                    RNA.substr(i, 3) == "GGC" ||
                    RNA.substr(i, 3) == "GGG" ||
                    RNA.substr(i, 3) == "GGU") Polypeptide.push_back('G');
            else if(RNA.substr(i, 3) == "CAC" ||
                    RNA.substr(i, 3) == "CAU") Polypeptide.push_back('H');
            else if(RNA.substr(i, 3) == "AUA" ||
                    RNA.substr(i, 3) == "AUC" ||
                    RNA.substr(i, 3) == "AUU") Polypeptide.push_back('I');
            else if(RNA.substr(i, 3) == "AAA" ||
                    RNA.substr(i, 3) == "AAG") Polypeptide.push_back('K');
            else if(RNA.substr(i, 3) == "CUA" ||
                    RNA.substr(i, 3) == "CUC" ||
                    RNA.substr(i, 3) == "CUG" ||
                    RNA.substr(i, 3) == "CUU" ||
                    RNA.substr(i, 3) == "UUA" ||
                    RNA.substr(i, 3) == "UUG") Polypeptide.push_back('L');
            else if(RNA.substr(i, 3) == "AUG") Polypeptide.push_back('M');
            else if(RNA.substr(i, 3) == "AAC" ||
                    RNA.substr(i, 3) == "AAU") Polypeptide.push_back('N');
            else if(RNA.substr(i, 3) == "CCA" ||
                    RNA.substr(i, 3) == "CCC" ||
                    RNA.substr(i, 3) == "CCG" ||
                    RNA.substr(i, 3) == "CCU") Polypeptide.push_back('P');
            else if(RNA.substr(i, 3) == "CAA" ||
                    RNA.substr(i, 3) == "CAG") Polypeptide.push_back('Q');
            else if(RNA.substr(i, 3) == "AGA" ||
                    RNA.substr(i, 3) == "AGG" ||
                    RNA.substr(i, 3) == "CGA" ||
                    RNA.substr(i, 3) == "CGC" ||
                    RNA.substr(i, 3) == "CGG" ||
                    RNA.substr(i, 3) == "CGU") Polypeptide.push_back('R');
            else if(RNA.substr(i, 3) == "AGC" ||
                    RNA.substr(i, 3) == "AGU" ||
                    RNA.substr(i, 3) == "UCA" ||
                    RNA.substr(i, 3) == "UCC" ||
                    RNA.substr(i, 3) == "UCG" ||
                    RNA.substr(i, 3) == "UCU") Polypeptide.push_back('S');
            else if(RNA.substr(i, 3) == "ACA" ||
                    RNA.substr(i, 3) == "ACC" ||
                    RNA.substr(i, 3) == "ACG" ||
                    RNA.substr(i, 3) == "ACU") Polypeptide.push_back('T');
            else if(RNA.substr(i, 3) == "GUA" ||
                    RNA.substr(i, 3) == "GUC" ||
                    RNA.substr(i, 3) == "GUG" ||
                    RNA.substr(i, 3) == "GUU") Polypeptide.push_back('V');
            else if(RNA.substr(i, 3) == "UGG") Polypeptide.push_back('W');
            else if(RNA.substr(i, 3) == "UAC" ||
                    RNA.substr(i, 3) == "UAU") Polypeptide.push_back('Y');
            else                               Polypeptide.push_back('-');
        }
    }
};

class genca:public genb
{
  public:
    char Monomer;
    genca(string Gene, char A, char B, char C):genb(Gene)
    {
        Monomer = C;
    }
};

class gencb:public genb
{
  public:
    char Monomer1;
    char Monomer2;
    gencb(string Gene, char A, char B, char C):genb(Gene)
    {
        Monomer1 = B;
        Monomer2 = C;
    }
};

class gencz:public genb
{
  public:
    string Replica;
    gencz(string Gene, string Template):genb(Gene)
    {
        for(int i = 0; i < Template.size(); i++)
        {
                 if(Template[i] == 'A') Replica.push_back('T');
            else if(Template[i] == 'C') Replica.push_back('G');
            else if(Template[i] == 'G') Replica.push_back('C');
            else if(Template[i] == 'T') Replica.push_back('A');
            else                        Replica.push_back('-');
        }
    }
};

```

```
class cell
{
  public:
    vector<string> DNAs;
    vector<string> tRNAs;
    vector<string> rRNAs;
    vector<string> Polypeptides1;
    vector<string> Polypeptides2;
    vector<string> Polypeptides3;
    vector<char> Monomers1;
    vector<char> Monomers2;
    vector<char> Monomers3;

    cell(vector<string> V0,
         vector<string> V1,
         vector<string> V2,
         vector<string> V3,
         vector<string> V4,
         vector<string> V5,
         vector<char> V6,
         vector<char> V7,
         vector<char> V8)
    {
        DNAs = V0;
        tRNAs = V1;
        rRNAs = V2;
        Polypeptides1 = V3;
        Polypeptides2 = V4;
        Polypeptides3 = V5;
        Monomers1 = V6;
        Monomers2 = V7;
        Monomers3 = V8;

        for(int i = 0; i < 10; i++)
        {
            gena G1(DNAs[0].substr(0, 9)); tRNAs.push_back(G1.RNA);
            gena G2(DNAs[0].substr(9, 9)); rRNAs.push_back(G2.RNA);
            genb G3(DNAs[0].substr(18, 9));Polypeptides1.push_back(G3.Polypeptide);
            genb G4(DNAs[0].substr(27, 9));Polypeptides2.push_back(G4.Polypeptide);
            genb G5(DNAs[0].substr(36, 9));Polypeptides3.push_back(G5.Polypeptide);
            genca G6(DNAs[0].substr(45, 9), 'U', 'V', 'X');
                Monomers1.push_back(G6.Monomer);
            gencb G7(DNAs[0].substr(54, 9), 'W', 'Y', 'Z');
                Monomers2.push_back(G7.Monomer1);
                Monomers3.push_back(G7.Monomer2);
        }
        vector<string> DNAreplicas;
        gencz G8a(DNAs[0].substr(63, 9), DNAs[0]);
            DNAreplicas.push_back(G8a.Replica);
        gencz G8b(DNAs[0].substr(63, 9), DNAs[1]);
            DNAreplicas.push_back(G8b.Replica);
        DNAs.insert(DNAs.begin() + 1, DNAreplicas.begin(), DNAreplicas.end());
    }
};

int main()
{
    string Sa, Sb;
    Sa="CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC";
    Sb="GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG";
    vector<string> V0;
    V0.push_back(Sa);
    V0.push_back(Sb);
    vector<string> V1(10, "GCAUGCGAC"), V2(10, "GCAUGCGAC");
    vector<string> V3(10, "HIK"), V4(10, "LMN"), V5(10, "PQR");
    vector<char> V6(10, 'X'), V7(10, 'Y'), V8(10, 'Z');
    cout << "Picture 1:" << "\n\n";
    for(int i = 0; i < V0.size(); i++)
        cout << V0[i] << endl;
        cout << "\n";
    for(int i = 0; i < V1.size(); i++)
        cout << V1[i] << " ";
    cout << "\n\n";
    for(int i = 0; i < V2.size(); i++)
```

```
        cout << V2[i] << " ";
    cout << "\n\n";
    for(int i = 0; i < V3.size(); i++)
        cout << V3[i] << " ";
    cout << "\n\n";
    for(int i = 0; i < V4.size(); i++)
        cout << V4[i] << " ";
    cout << "\n\n";
    for(int i = 0; i < V5.size(); i++)
        cout << V5[i] << " ";
    cout << "\n\n";
    for(int i = 0; i < V6.size(); i++)
        cout << V6[i] << " ";
    cout << "\n\n";
    for(int i = 0; i < V7.size(); i++)
        cout << V7[i] << " ";
    cout << "\n\n";
    for(int i = 0; i < V8.size(); i++)
        cout << V8[i] << " ";
    cout << "\n\n";

    cell Cell(V0, V1, V2, V3, V4, V5, V6, V7, V8);
    cout << "Picture 2:" << "\n\n";
    for(int i = 0; i < Cell.DNAs.size(); i++)
        cout << Cell.DNAs[i] << endl;
    cout << "\n";
    for(int i = 0; i < Cell.tRNAs.size(); i++)
        cout << Cell.tRNAs[i] << " ";
    cout << "\n\n";
    for(int i = 0; i < Cell.rRNAs.size(); i++)
        cout << Cell.rRNAs[i] << " ";
    cout << "\n\n";
    for(int i = 0; i < Cell.Polypeptides1.size(); i++)
        cout << Cell.Polypeptides1[i] << " ";
    cout << "\n\n";
    for(int i = 0; i < Cell.Polypeptides2.size(); i++)
        cout << Cell.Polypeptides2[i] << " ";
    cout << "\n\n";
    for(int i = 0; i < Cell.Polypeptides3.size(); i++)
        cout << Cell.Polypeptides3[i] << " ";
    cout << "\n\n";
    for(int i = 0; i < Cell.Monomers1.size(); i++)
        cout << Cell.Monomers1[i] << " ";
    cout << "\n\n";
    for(int i = 0; i < Cell.Monomers2.size(); i++)
        cout << Cell.Monomers2[i] << " ";
    cout << "\n\n";
    for(int i = 0; i < Cell.Monomers3.size(); i++)
        cout << Cell.Monomers3[i] << " ";
    cout << "\n\n";

    return 0;
}
```

Experiment3-1: Understanding source code

Here, C++ program has to set up an experiment for construction of an *in silico* genome expression network (GENome).

Info	**Genome expression network** Since the genes are usually associated in a genome, corresponding GENs are organised in more complicated unit of information processing – genome expression network (GENome). This life pattern is roughly equal to the cell. Whereas gene and genome are notions that refer to how information is stored, GEN and GENome refer to how the gene and genome work. The GENome can be considered as a highly regular composition of interacting GENs. During information processing in particular GEN, it is just the job of other GENs to provide necessary elements for gene expression machinery. Collectively, GENs in GENome work to replicate the complete genome so that 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. In particular cell, the GENome is suited to specific subset of sources of mass, impulse (momentum), and energy to produce their usable forms essential for the cell life.
Note!	**Genome expression network** In GENome, the information processing involves two tightly coupled reactions – genome expression and genome replication.

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`.

Info	**Object type `vector`** `vector` is an object type defined in the file `vector` of the Standard C++ Library. It is specifically designed for construction of objects that can hold/contain a pool of other objects and for operation on them. In comparison to other standard sequence containers, `vector` container is most efficient to add element at the end of the container or to delete last element from the container.
Note!	**Object type `vector`** In Standard C++ Library, the object type `vector` is defined as a template. A template takes other object types as parameters. 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).

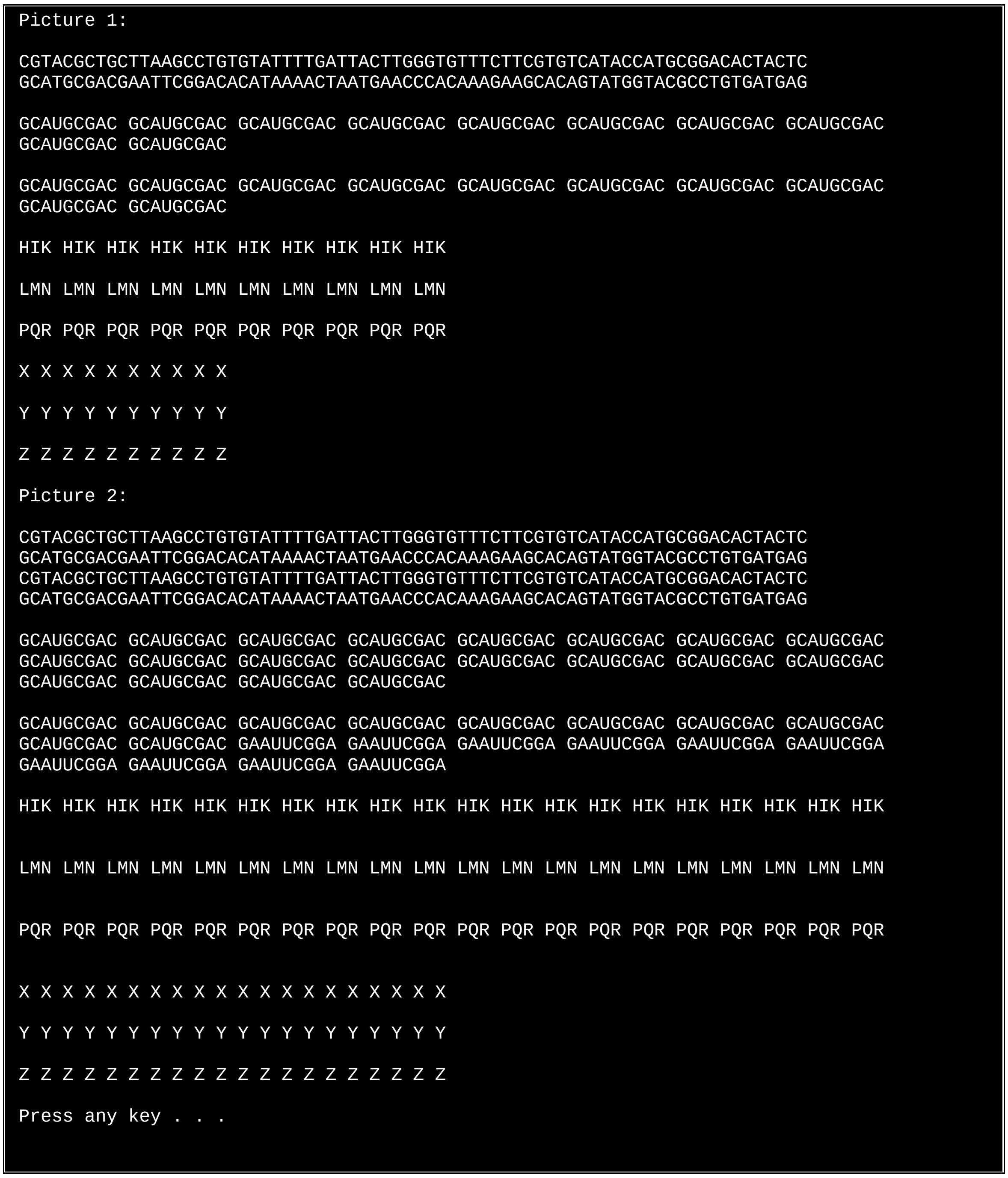
Expectedly, the constructor function of the new object type `cell` has 9 parameters. The body of the constructor function contains code for control structure specifying how genes must be expressed to double contents of 9 pools.

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.

Experiment3-1: Building and executing machine code

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
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC
GCAUGCGAC GCAUGCGAC

GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC GCAUGCGAC
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
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG
CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

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

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

Press any key . . .
```

Part 4

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

Experiment4-1: Entering source code

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:

```
//Experiment 4-1
/*Nikita Tirjatkin
  Laboratory for in silico life construction
  Januar 2011*/

#include <string>
#include <vector>
#include <deque>
#include <iostream>
using namespace std;

class gena
{
  public:
    string RNA;
    gena(string Gene)
    {
        for(int i = 0; i < Gene.size(); i++)
        {
                 if(Gene[i] == 'A') RNA.push_back('U');
            else if(Gene[i] == 'C') RNA.push_back('G');
            else if(Gene[i] == 'G') RNA.push_back('C');
            else if(Gene[i] == 'T') RNA.push_back('A');
            else                    RNA.push_back('-');
        }
    }
};

class genb:public gena
{
  public:
    string Polypeptide;
    genb(string Gene):gena(Gene)
    {
        for(int i = 0; i < RNA.size(); i += 3)
        {
                 if(RNA.substr(i, 3) == "GCA" ||
                    RNA.substr(i, 3) == "GCC" ||
                    RNA.substr(i, 3) == "GCG" ||
                    RNA.substr(i, 3) == "GCU") Polypeptide.push_back('A');
            else if(RNA.substr(i, 3) == "UGC" ||
                    RNA.substr(i, 3) == "UGU") Polypeptide.push_back('C');
            else if(RNA.substr(i, 3) == "GAC" ||
                    RNA.substr(i, 3) == "GAU") Polypeptide.push_back('D');
            else if(RNA.substr(i, 3) == "GAA" ||
                    RNA.substr(i, 3) == "GAG") Polypeptide.push_back('E');
            else if(RNA.substr(i, 3) == "UUC" ||
                    RNA.substr(i, 3) == "UUU") Polypeptide.push_back('F');
            else if(RNA.substr(i, 3) == "GGA" ||
                    RNA.substr(i, 3) == "GGC" ||
                    RNA.substr(i, 3) == "GGG" ||
                    RNA.substr(i, 3) == "GGU") Polypeptide.push_back('G');
            else if(RNA.substr(i, 3) == "CAC" ||
                    RNA.substr(i, 3) == "CAU") Polypeptide.push_back('H');
            else if(RNA.substr(i, 3) == "AUA" ||
                    RNA.substr(i, 3) == "AUC" ||
                    RNA.substr(i, 3) == "AUU") Polypeptide.push_back('I');
            else if(RNA.substr(i, 3) == "AAA" ||
                    RNA.substr(i, 3) == "AAG") Polypeptide.push_back('K');
            else if(RNA.substr(i, 3) == "CUA" ||
                    RNA.substr(i, 3) == "CUC" ||
                    RNA.substr(i, 3) == "CUG" ||
                    RNA.substr(i, 3) == "CUU" ||
                    RNA.substr(i, 3) == "UUA" ||
                    RNA.substr(i, 3) == "UUG") Polypeptide.push_back('L');
            else if(RNA.substr(i, 3) == "AUG") Polypeptide.push_back('M');
            else if(RNA.substr(i, 3) == "AAC" ||
                    RNA.substr(i, 3) == "AAU") Polypeptide.push_back('N');
            else if(RNA.substr(i, 3) == "CCA" ||
                    RNA.substr(i, 3) == "CCC" ||
                    RNA.substr(i, 3) == "CCG" ||
                    RNA.substr(i, 3) == "CCU") Polypeptide.push_back('P');
            else if(RNA.substr(i, 3) == "CAA" ||
                    RNA.substr(i, 3) == "CAG") Polypeptide.push_back('Q');
            else if(RNA.substr(i, 3) == "AGA" ||
                    RNA.substr(i, 3) == "AGG" ||
                    RNA.substr(i, 3) == "CGA" ||
                    RNA.substr(i, 3) == "CGC" ||
                    RNA.substr(i, 3) == "CGG" ||
                    RNA.substr(i, 3) == "CGU") Polypeptide.push_back('R');
            else if(RNA.substr(i, 3) == "AGC" ||
                    RNA.substr(i, 3) == "AGU" ||
                    RNA.substr(i, 3) == "UCA" ||
                    RNA.substr(i, 3) == "UCC" ||
                    RNA.substr(i, 3) == "UCG" ||
                    RNA.substr(i, 3) == "UCU") Polypeptide.push_back('S');
            else if(RNA.substr(i, 3) == "ACA" ||
                    RNA.substr(i, 3) == "ACC" ||
                    RNA.substr(i, 3) == "ACG" ||
                    RNA.substr(i, 3) == "ACU") Polypeptide.push_back('T');
            else if(RNA.substr(i, 3) == "GUA" ||
                    RNA.substr(i, 3) == "GUC" ||
                    RNA.substr(i, 3) == "GUG" ||
                    RNA.substr(i, 3) == "GUU") Polypeptide.push_back('V');
            else if(RNA.substr(i, 3) == "UGG") Polypeptide.push_back('W');
            else if(RNA.substr(i, 3) == "UAC" ||
                    RNA.substr(i, 3) == "UAU") Polypeptide.push_back('Y');
            else                               Polypeptide.push_back('-');
        }
    }
};

class genca:public genb
{
  public:
    char Monomer;
    genca(string Gene, char A, char B, char C):genb(Gene)
    {
        Monomer = C;
    }
};

class gencb:public genb
{
  public:
    char Monomer1;
    char Monomer2;
    gencb(string Gene, char A, char B, char C):genb(Gene)
    {
        Monomer1 = B;
        Monomer2 = C;
    }
};

class gencz:public genb
{
  public:
    string Replica;
    gencz(string Gene, string Template):genb(Gene)
    {
        for(int i = 0; i < Template.size(); i++)
        {
                 if(Template[i] == 'A') Replica.push_back('T');
            else if(Template[i] == 'C') Replica.push_back('G');
            else if(Template[i] == 'G') Replica.push_back('C');
            else if(Template[i] == 'T') Replica.push_back('A');
            else                        Replica.push_back('-');
        }
    }
};
```

```

class cell
{
  public:
    vector<string> DNAs;
    vector<string> tRNAs;
    vector<string> rRNAs;
    vector<string> Polypeptides1;
    vector<string> Polypeptides2;
    vector<string> Polypeptides3;
    vector<char> Monomers1;
    vector<char> Monomers2;
    vector<char> Monomers3;

    cell(vector<string> V0,
         vector<string> V1,
         vector<string> V2,
         vector<string> V3,
         vector<string> V4,
         vector<string> V5,
         vector<char> V6,
         vector<char> V7,
         vector<char> V8)
    {
        DNAs = V0;
        tRNAs = V1;
        rRNAs = V2;
        Polypeptides1 = V3;
        Polypeptides2 = V4;
        Polypeptides3 = V5;
        Monomers1 = V6;
        Monomers2 = V7;
        Monomers3 = V8;

        for(int i = 0; i < 10; i++)
        {
            gena G1(DNAs[0].substr(0, 9)); tRNAs.push_back(G1.RNA);
            gena G2(DNAs[0].substr(9, 9)); rRNAs.push_back(G2.RNA);
            genb G3(DNAs[0].substr(18, 9));Polypeptides1.push_back(G3.Polypeptide);
            genb G4(DNAs[0].substr(27, 9));Polypeptides2.push_back(G4.Polypeptide);
            genb G5(DNAs[0].substr(36, 9));Polypeptides3.push_back(G5.Polypeptide);
            genca G6(DNAs[0].substr(45, 9), 'U', 'V', 'X');
                Monomers1.push_back(G6.Monomer);
            gencb G7(DNAs[0].substr(54, 9), 'W', 'Y', 'Z');
                Monomers2.push_back(G7.Monomer1);
                Monomers3.push_back(G7.Monomer2);
        }
        vector<string> DNAreplicas;
        gencz G8a(DNAs[0].substr(63, 9), DNAs[0]);
            DNAreplicas.push_back(G8a.Replica);
        gencz G8b(DNAs[0].substr(63, 9), DNAs[1]);
            DNAreplicas.push_back(G8b.Replica);
        DNAs.insert(DNAs.begin() + 1, DNAreplicas.begin(), DNAreplicas.end());
    }
};

class cp
{
  public:
    deque<cell> Cells;
    cp(cell Cell)
    {
        Cells.push_back(Cell);

        while(Cells.size() <= Cells.max_size())
        {
            Cell = Cells.front();

            cout << "\nCell " << Cells.size() << "\n\n";
            for(int i = 0; i < Cell.DNAs.size(); i++)
                cout << Cell.DNAs[i] << " ";
            cout << "\n\n";
            for(int i = 0; i < Cell.tRNAs.size(); i++)
                cout << Cell.tRNAs[i] << " ";
            cout << "\n\n";
            for(int i = 0; i < Cell.rRNAs.size(); i++)
                cout << Cell.rRNAs[i] << " ";
            cout << "\n\n";
            for(int i = 0; i < Cell.Polypeptides1.size(); i++)
                cout << Cell.Polypeptides1[i] << " ";
            cout << "\n\n";
            for(int i = 0; i < Cell.Polypeptides2.size(); i++)
                cout << Cell.Polypeptides2[i] << " ";
            cout << "\n\n";
            for(int i = 0; i < Cell.Polypeptides3.size(); i++)
                cout << Cell.Polypeptides3[i] << " ";
            cout << "\n\n";
            for(int i = 0; i < Cell.Monomers1.size(); i++)
                cout << Cell.Monomers1[i] << " ";
            cout << "\n\n";
            for(int i = 0; i < Cell.Monomers2.size(); i++)
                cout << Cell.Monomers2[i] << " ";
            cout << "\n\n";
            for(int i = 0; i < Cell.Monomers3.size(); i++)
                cout << Cell.Monomers3[i] << " ";
            cout << "\n\n";

            Cells.pop_front();

            vector<string> lDNAs(Cell.DNAs.begin(),
                                 Cell.DNAs.begin() +
                                 Cell.DNAs.size()/2);
            vector<string> rDNAs(Cell.DNAs.begin() +
                                 Cell.DNAs.size()/2,
                                 Cell.DNAs.end());
            vector<string> ltRNAs(Cell.tRNAs.begin(),
                                  Cell.tRNAs.begin() +
                                  Cell.tRNAs.size()/2);
            vector<string> rtRNAs(Cell.tRNAs.begin() +
                                  Cell.tRNAs.size()/2,
                                  Cell.tRNAs.end());
            vector<string> lrRNAs(Cell.rRNAs.begin(),
                                  Cell.rRNAs.begin() +
                                  Cell.rRNAs.size()/2);
            vector<string> rrRNAs(Cell.rRNAs.begin() +
                                  Cell.rRNAs.size()/2,
                                  Cell.rRNAs.end());
            vector<string> lPolypeptides1(Cell.Polypeptides1.begin(),
                                          Cell.Polypeptides1.begin() +
                                          Cell.Polypeptides1.size()/2);
            vector<string> rPolypeptides1(Cell.Polypeptides1.begin() +
                                          Cell.Polypeptides1.size()/2,
                                          Cell.Polypeptides1.end());
            vector<string> lPolypeptides2(Cell.Polypeptides2.begin(),
                                          Cell.Polypeptides2.begin() +
                                          Cell.Polypeptides2.size()/2);
            vector<string> rPolypeptides2(Cell.Polypeptides2.begin() +
                                          Cell.Polypeptides2.size()/2,
                                          Cell.Polypeptides2.end());
            vector<string> lPolypeptides3(Cell.Polypeptides3.begin(),
                                          Cell.Polypeptides3.begin() +
                                          Cell.Polypeptides3.size()/2);
            vector<string> rPolypeptides3(Cell.Polypeptides3.begin() +
                                          Cell.Polypeptides3.size()/2,
                                          Cell.Polypeptides3.end());
            vector<char> lMonomers1(Cell.Monomers1.begin(),
                                    Cell.Monomers1.begin() +
                                    Cell.Monomers1.size()/2);
            vector<char> rMonomers1(Cell.Monomers1.begin() +
                                    Cell.Monomers1.size()/2,
                                    Cell.Monomers1.end());
            vector<char> lMonomers2(Cell.Monomers2.begin(),
                                    Cell.Monomers2.begin() +
                                    Cell.Monomers2.size()/2);
            vector<char> rMonomers2(Cell.Monomers2.begin() +
                                    Cell.Monomers2.size()/2,
                                    Cell.Monomers2.end());
            vector<char> lMonomers3(Cell.Monomers3.begin(),
                                    Cell.Monomers3.begin() +
                                    Cell.Monomers3.size()/2);
            vector<char> rMonomers3(Cell.Monomers3.begin() +
                                    Cell.Monomers3.size()/2,
                                    Cell.Monomers3.end());

            cell lCell(lDNAs,
                       ltRNAs,
                       lrRNAs,
                       lPolypeptides1,
                       lPolypeptides2,
                       lPolypeptides3,
                       lMonomers1,
                       lMonomers2,
                       lMonomers3);
            cell rCell(rDNAs,
                       rtRNAs,
                       rrRNAs,
                       rPolypeptides1,
                       rPolypeptides2,
                       rPolypeptides3,
                       rMonomers1,
                       rMonomers2,
                       rMonomers3);
            Cells.push_back(lCell);
            Cells.push_back(rCell);
        }
    }
};

int main()
{
    string Sa, Sb;
    Sa="CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC";
    Sb="GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG";
    vector<string> V0;
    V0.push_back(Sa);
    V0.push_back(Sb);
    vector<string> V1(10, "GCAUGCGAC"), V2(10, "GCAUGCGAC");
    vector<string> V3(10, "HIK"), V4(10, "LMN"), V5(10, "PQR");
    vector<char> V6(10, 'X'), V7(10, 'Y'), V8(10, 'Z');
    cell PrimordialCell(V0, V1, V2, V3, V4, V5, V6, V7, V8);

    cp CellProgression(PrimordialCell);

    return 0;
}
```

Experiment4-1: Understanding source code

Here, C++ program has to set up an experiment for construction of an *in silico* genome multiplication network.

Info

Genome multiplication network
Progressive genome replication by genome expression leads to much more complicated unit of information processing – genome multiplication network. 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 entire living world is the 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 producing 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`.

Info

Object type `deque`
`deque` is an object type defined in the file `deque` of the Standard C++ Library. It is specifically designed for construction of objects that can hold/contain a pool of other objects and for operation on them. The `deque` container is a double ended queue. Elements can be efficiently added and deleted from any of its ends. The `deque` container is suited very well to be filled in in-level order so that a binary tree will be produced.

Note!

Object type `deque`
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 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.

Info

Iterative control structure `while`
Its form is

```
while(condition)
    statement
```

The program evaluates `condition`. If `condition` is false, `statement` will be skipped and the program will proceed to the statement after the control structure. If `condition` is true, `statement` will be executed and the program will loop back to evaluate `condition` again. This cycle will be repeated a certain number of times until `condition` will become false.

Note!

Iterative control structure `while`
Here, 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.

Experiment4-1: Building and executing machine code

Build Experiment4-1 and run the executable machine code.

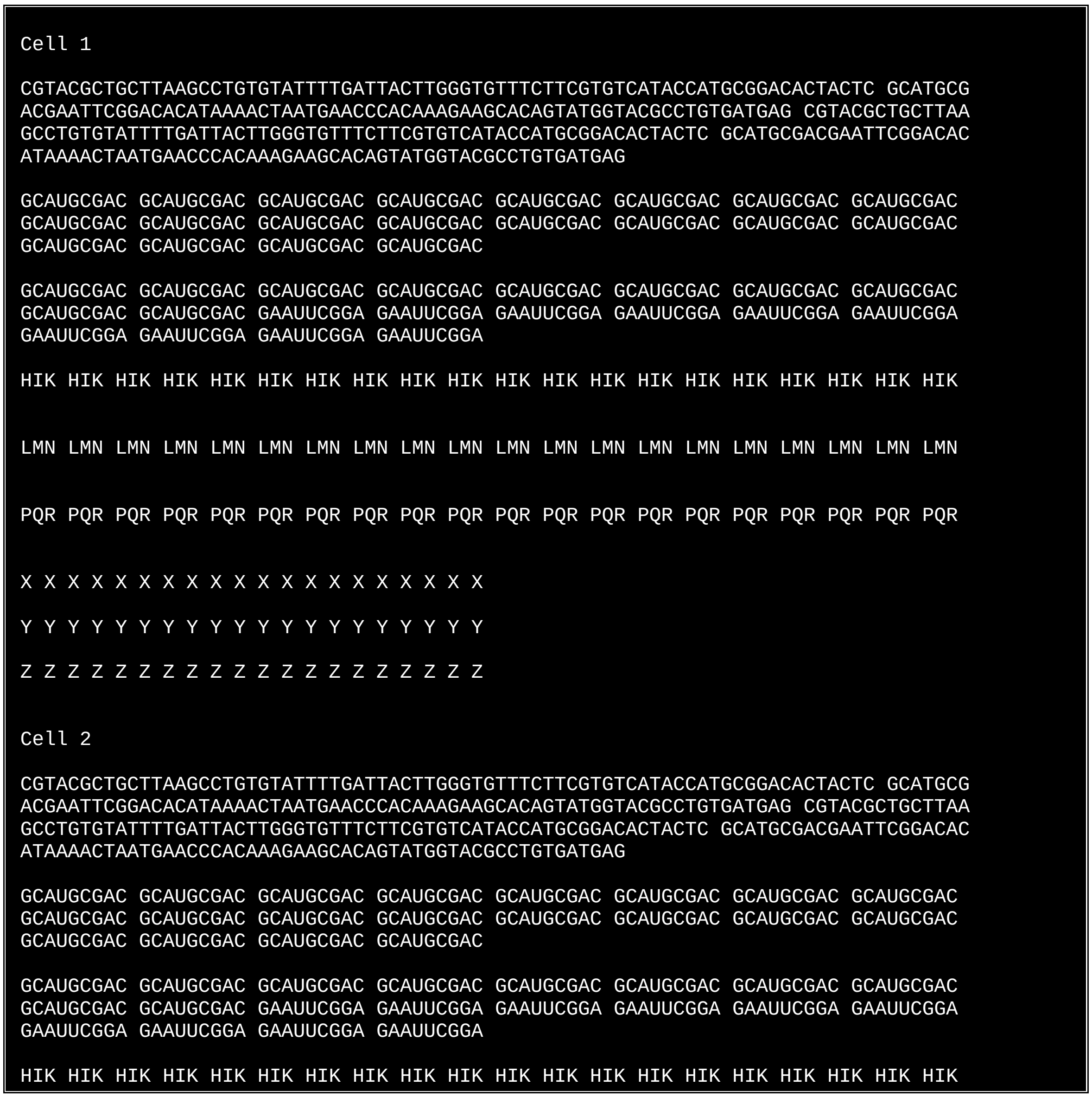
The console window will display pictures documenting how the experiment for *in silico* genome multiplication proceeds. However, pictures flow too quickly. Close the console window, go to the line 205 of the file Experiment4-1.cpp, and replace the statement

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

by the statement

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

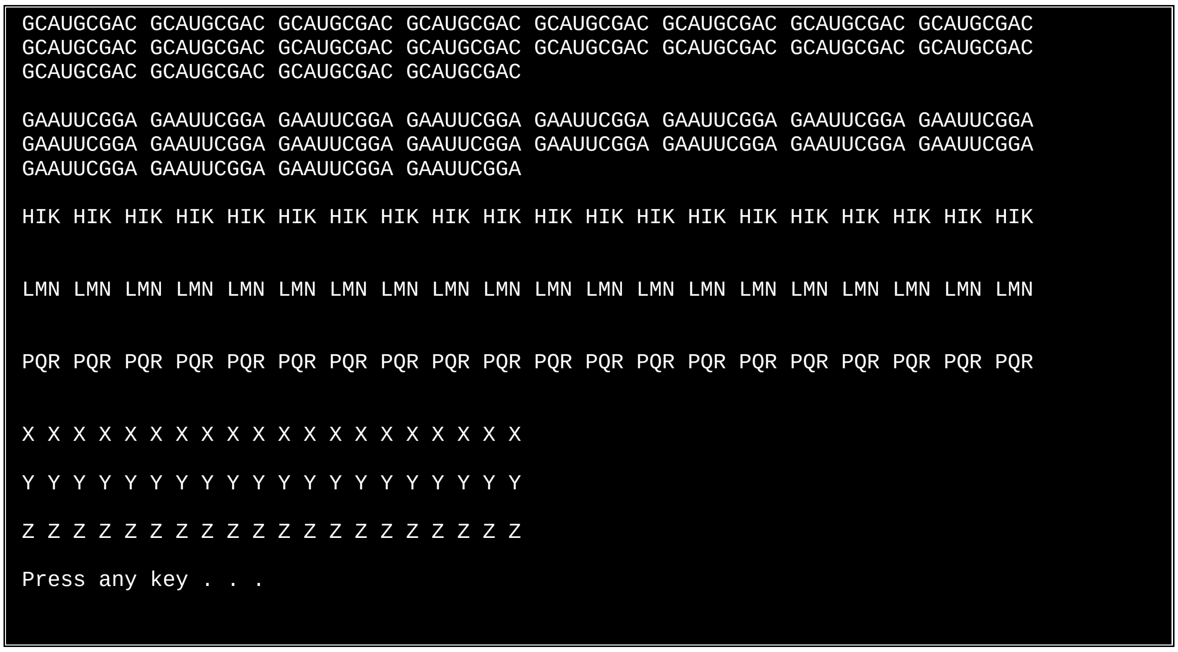
Build Experiment4-1 anew and run the executable machine code. The console window will display only the first 7 cells (1 -> 2 -> 4) from the cell progression:



```
Cell 1

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC GCATGCG
ACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG CGTACGCTGCTTAA
GCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC GCATGCGACGAATTCGGACAC
ATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

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

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


Cell 2

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC GCATGCG
ACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG CGTACGCTGCTTAA
GCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC GCATGCGACGAATTCGGACAC
ATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

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

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


Cell 3

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC GCATGCG
ACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG CGTACGCTGCTTAA
GCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC GCATGCGACGAATTCGGACAC
ATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

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 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


Cell 4

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC GCATGCG
ACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG CGTACGCTGCTTAA
GCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC GCATGCGACGAATTCGGACAC
ATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

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

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


Cell 5

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC GCATGCG
ACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG CGTACGCTGCTTAA
GCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC GCATGCGACGAATTCGGACAC
ATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

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 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


Cell 6

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC GCATGCG
ACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG CGTACGCTGCTTAA
GCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC GCATGCGACGAATTCGGACAC
ATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

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 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


Cell 7

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC GCATGCG
ACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG CGTACGCTGCTTAA
GCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC GCATGCGACGAATTCGGACAC
ATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG
```



```
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 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

Press any key . . .
```

Experiment4-2: Entering source code

Create a new solution/project Experiment4-2 and add a new file Experiment4-2.cpp. Enter the following code within the file Experiment4-2.cpp:

```cpp
//Experiment 4-2
/*Nikita Tirjatkin
  Laboratory for in silico life construction
  Januar 2011*/

#include <string>
#include <vector>
#include <list>
#include <deque>
#include <iostream>
using namespace std;

class gena
{
  public:
    string RNA;
    gena(string Gene)
    {
        for(int i = 0; i < Gene.size(); i++)
        {
                 if(Gene[i] == 'A') RNA.push_back('U');
            else if(Gene[i] == 'C') RNA.push_back('G');
            else if(Gene[i] == 'G') RNA.push_back('C');
            else if(Gene[i] == 'T') RNA.push_back('A');
            else                    RNA.push_back('-');
        }
    }
};

class genb:public gena
{
  public:
    string Polypeptide;
    genb(string Gene):gena(Gene)
    {
        for(int i = 0; i < RNA.size(); i += 3)
        {
                 if(RNA.substr(i, 3) == "GCA" ||
                    RNA.substr(i, 3) == "GCC" ||
                    RNA.substr(i, 3) == "GCG" ||
                    RNA.substr(i, 3) == "GCU") Polypeptide.push_back('A');
            else if(RNA.substr(i, 3) == "UGC" ||
                    RNA.substr(i, 3) == "UGU") Polypeptide.push_back('C');
            else if(RNA.substr(i, 3) == "GAC" ||
                    RNA.substr(i, 3) == "GAU") Polypeptide.push_back('D');
            else if(RNA.substr(i, 3) == "GAA" ||
                    RNA.substr(i, 3) == "GAG") Polypeptide.push_back('E');
            else if(RNA.substr(i, 3) == "UUC" ||
                    RNA.substr(i, 3) == "UUU") Polypeptide.push_back('F');
            else if(RNA.substr(i, 3) == "GGA" ||
                    RNA.substr(i, 3) == "GGC" ||
                    RNA.substr(i, 3) == "GGG" ||
                    RNA.substr(i, 3) == "GGU") Polypeptide.push_back('G');
            else if(RNA.substr(i, 3) == "CAC" ||
                    RNA.substr(i, 3) == "CAU") Polypeptide.push_back('H');
            else if(RNA.substr(i, 3) == "AUA" ||
                    RNA.substr(i, 3) == "AUC" ||
                    RNA.substr(i, 3) == "AUU") Polypeptide.push_back('I');
            else if(RNA.substr(i, 3) == "AAA" ||
                    RNA.substr(i, 3) == "AAG") Polypeptide.push_back('K');
            else if(RNA.substr(i, 3) == "CUA" ||
                    RNA.substr(i, 3) == "CUC" ||
                    RNA.substr(i, 3) == "CUG" ||
                    RNA.substr(i, 3) == "CUU" ||
                    RNA.substr(i, 3) == "UUA" ||
                    RNA.substr(i, 3) == "UUG") Polypeptide.push_back('L');
```

```cpp
                else if(RNA.substr(i, 3) == "AUG") Polypeptide.push_back('M');
                else if(RNA.substr(i, 3) == "AAC" ||
                        RNA.substr(i, 3) == "AAU") Polypeptide.push_back('N');
                else if(RNA.substr(i, 3) == "CCA" ||
                        RNA.substr(i, 3) == "CCC" ||
                        RNA.substr(i, 3) == "CCG" ||
                        RNA.substr(i, 3) == "CCU") Polypeptide.push_back('P');
                else if(RNA.substr(i, 3) == "CAA" ||
                        RNA.substr(i, 3) == "CAG") Polypeptide.push_back('Q');
                else if(RNA.substr(i, 3) == "AGA" ||
                        RNA.substr(i, 3) == "AGG" ||
                        RNA.substr(i, 3) == "CGA" ||
                        RNA.substr(i, 3) == "CGC" ||
                        RNA.substr(i, 3) == "CGG" ||
                        RNA.substr(i, 3) == "CGU") Polypeptide.push_back('R');
                else if(RNA.substr(i, 3) == "AGC" ||
                        RNA.substr(i, 3) == "AGU" ||
                        RNA.substr(i, 3) == "UCA" ||
                        RNA.substr(i, 3) == "UCC" ||
                        RNA.substr(i, 3) == "UCG" ||
                        RNA.substr(i, 3) == "UCU") Polypeptide.push_back('S');
                else if(RNA.substr(i, 3) == "ACA" ||
                        RNA.substr(i, 3) == "ACC" ||
                        RNA.substr(i, 3) == "ACG" ||
                        RNA.substr(i, 3) == "ACU") Polypeptide.push_back('T');
                else if(RNA.substr(i, 3) == "GUA" ||
                        RNA.substr(i, 3) == "GUC" ||
                        RNA.substr(i, 3) == "GUG" ||
                        RNA.substr(i, 3) == "GUU") Polypeptide.push_back('V');
                else if(RNA.substr(i, 3) == "UGG") Polypeptide.push_back('W');
                else if(RNA.substr(i, 3) == "UAC" ||
                        RNA.substr(i, 3) == "UAU") Polypeptide.push_back('Y');
                else                               Polypeptide.push_back('-');
            }
        }
};

class genca:public genb
{
  public:
    char Monomer;
    genca(string Gene, char A, char B, char C):genb(Gene)
    {
        Monomer = C;
    }
};

class gencb:public genb
{
  public:
    char Monomer1;
    char Monomer2;
    gencb(string Gene, char A, char B, char C):genb(Gene)
    {
        Monomer1 = B;
        Monomer2 = C;
    }
};

class gencz:public genb
{
  public:
    string Replica;
    gencz(string Gene, string Template):genb(Gene)
    {
        for(int i = 0; i < Template.size(); i++)
        {
                 if(Template[i] == 'A') Replica.push_back('T');
            else if(Template[i] == 'C') Replica.push_back('G');
            else if(Template[i] == 'G') Replica.push_back('C');
            else if(Template[i] == 'T') Replica.push_back('A');
            else                        Replica.push_back('-');
        }
    }
```

```
};

class component
{
public:
    virtual ~component(){}
    virtual void displayImage()const = 0;
};

class monomer:public component
{
public:
    char Monomer;
    monomer();
    monomer(char M):Monomer(M){};
    virtual ~monomer(){}
    void displayImage()const{cout << Monomer << " ";}
};

class polymer:public component
{
public:
    string Polymer;
    polymer();
    polymer(string P):Polymer(P){};
    virtual ~polymer(){}
    void displayImage()const{cout << Polymer << " ";}
};

class composite:public component
{
public:
    list<component *> pComponents;
    virtual ~composite()
    {
        list<component *>::iterator i;
        for(i = pComponents.begin(); i != pComponents.end(); ++i)
        {
            delete *i;
            *i = 0;
        }
    }
    void displayImage()const
    {
        list<component *>::const_iterator i;
        for(i = pComponents.begin(); i != pComponents.end(); ++i)
            (*i)->displayImage();
        cout << "\n\n";
    }
    void addComponent(component * pC){pComponents.push_back(pC);}
    void eraseComponent(list<component *>::iterator P){pComponents.erase(P);}
};

class compositor
{
public:
    composite * composeComposite1(string S)
    {
        gena G1(S.substr(0, 9)); polymer * tRNA = new polymer(G1.RNA);
        gena G2(S.substr(9, 9)); polymer * rRNA = new polymer(G2.RNA);
        genb G3(S.substr(18, 9)); polymer * Polypeptide1 =
                                          new polymer(G3.Polypeptide);
        genb G4(S.substr(27, 9)); polymer * Polypeptide2 =
                                          new polymer(G4.Polypeptide);
        genb G5(S.substr(36, 9)); polymer * Polypeptide3 =
                                          new polymer(G5.Polypeptide);
        genca G6(S.substr(45, 9), 'U', 'V', 'X'); monomer * Monomer1 =
                                          new monomer(G6.Monomer);
        gencb G7(S.substr(54, 9), 'W', 'Y', 'Z'); monomer * Monomer2 =
                                          new monomer(G7.Monomer1);
                                          monomer * Monomer3 =
                                          new monomer(G7.Monomer2);
        composite * tRNAs = new composite;
        composite * rRNAs = new composite;
        composite * Polypeptides1 = new composite;
        composite * Polypeptides2 = new composite;
        composite * Polypeptides3 = new composite;
        composite * Monomers1 = new composite;
        composite * Monomers2 = new composite;
        composite * Monomers3 = new composite;
        for(unsigned int i = 0; i < 10; ++i)
        {
            tRNAs->addComponent(tRNA);
            rRNAs->addComponent(rRNA);
            Polypeptides1->addComponent(Polypeptide1);
            Polypeptides2->addComponent(Polypeptide2);
            Polypeptides3->addComponent(Polypeptide3);
            Monomers1->addComponent(Monomer1);
            Monomers2->addComponent(Monomer2);
            Monomers3->addComponent(Monomer3);
        }
        composite * Ca = new composite;
        Ca->addComponent(tRNAs);
        Ca->addComponent(rRNAs);
        Ca->addComponent(Polypeptides1);
        Ca->addComponent(Polypeptides2);
        Ca->addComponent(Polypeptides3);
        Ca->addComponent(Monomers1);
        Ca->addComponent(Monomers2);
        Ca->addComponent(Monomers3);
        composite * C = new composite;
        C->addComponent(Ca);
        return C;
    }
    composite * composeComposite2(string S)
    {
        genb G1(S.substr(3, 27)); polymer * Polypeptide4 =
                                          new polymer(G1.Polypeptide);
        genb G2(S.substr(6, 27)); polymer * Polypeptide5 =
                                          new polymer(G2.Polypeptide);
        genb G3(S.substr(9, 27)); polymer * Polypeptide6 =
                                          new polymer(G3.Polypeptide);
        composite * Polypeptides4 = new composite;
        composite * Polypeptides5 = new composite;
        composite * Polypeptides6 = new composite;
        for(unsigned int i = 0; i < 20; ++i)
        {
            Polypeptides4->addComponent(Polypeptide4);
            Polypeptides5->addComponent(Polypeptide5);
            Polypeptides6->addComponent(Polypeptide6);
        }
        composite * Cb = new composite;
        Cb->addComponent(Polypeptides4);
        Cb->addComponent(Polypeptides5);
        Cb->addComponent(Polypeptides6);
        return Cb;
    }
};

class cell
{
public:
    vector<string> DNA;
    composite * pMatrix;
    composite * plMatrix, * prMatrix;
    static int Number;
    cell(vector<string> V, composite * pM):DNA(V), pMatrix(pM)
    {
        Number++;
        cout << "Cell " << Number << "\n\n" << "Picture 1:\n\n";
        for(int i = 0; i < DNA.size(); ++i)
            cout << DNA[i] << endl;
        cout << endl;
        pMatrix->displayImage();

        composite * pMatrix2;
        compositor Compositor2;
        pMatrix2 = Compositor2.composeComposite2(DNA[0]);
        pMatrix->addComponent(pMatrix2);
        list<component *>::iterator Position = pMatrix->pComponents.begin();
        pMatrix->eraseComponent(Position);
        cout << "Picture 2:\n\n";
        for(int i = 0; i < DNA.size(); ++i)
            cout << DNA[i] << endl;
        cout << endl;
        pMatrix->displayImage();

        compositor Compositor3;
        plMatrix = Compositor3.composeComposite1(DNA[0]);
        prMatrix = Compositor3.composeComposite1(DNA[0]);
        Position = pMatrix->pComponents.begin();
        pMatrix->eraseComponent(Position);
        vector<string> DNAreplicas;
        gencz G8a(DNA[0].substr(63, 9), DNA[0]);
        DNAreplicas.push_back(G8a.Replica);
        gencz G8b(DNA[0].substr(63, 9), DNA[1]);
        DNAreplicas.push_back(G8b.Replica);
        DNA.insert(DNA.begin() + 1, DNAreplicas.begin(), DNAreplicas.end());
        cout << "Picture 3:\n\n";
        for(int i = 0; i < DNA.size(); ++i)
            cout << DNA[i] << endl;
        cout << endl;
        pMatrix->displayImage();
        plMatrix->displayImage();
        prMatrix->displayImage();
    }
};

int cell::Number = 0;

class cp
{
  public:
    deque<cell> Cells;
    cp(cell Cell)
    {
        Cells.push_back(Cell);

        while(Cell.Number < 7)//Cells.max_size())
        {
            Cell = Cells.front();
            Cells.pop_front();

            vector<string> lDNA(Cell.DNA.begin(), Cell.DNA.begin() + Cell.DNA.size()/2);
            vector<string> rDNA(Cell.DNA.begin() + Cell.DNA.size()/2, Cell.DNA.end());
            cell lCell(lDNA, Cell.plMatrix);
            cell rCell(rDNA, Cell.prMatrix);
            Cells.push_back(lCell);
            Cells.push_back(rCell);
        }
    }
};

int main()
{

    vector<string> Strands;
    string S("CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC");
    Strands.push_back(S);
    gencz Ga0(Strands[0].substr(63, 9), S);
    Strands.push_back(Ga0.Replica);

    composite * pMatrix1;
    compositor Compositor1;
    pMatrix1 = Compositor1.composeComposite1(S);
    cell PrimordialCell(Strands, pMatrix1);
    cp CellProgression(PrimordialCell);

    return 0;
}
```

Experiment4-2: Understanding source code

Here, C++ program has to set up an experiment for construction of an *in silico* genome multiplication network too. However, the code contains some improvements.

Lines from 143 to 268 contain definitions of the new object types `component`, `monomer`, `polymer`, `composite`, and `compositor`. These object types are participants of the so called Composite Design Pattern.

Info

Composite Design Pattern
Composite Design Pattern is a structural design pattern. It is used to represent part-whole hierarchies ignoring differences between primitives and their containers. All objects in the composition can be then treated uniformly. Primitive objects can be composed into more complex objects which in turn can be composed and so on.

The new object type `component` declares the interface for objects in the composition. It is designed as an abstract base object type.

Info

Abstract base object type
Object type that contains at least one pure virtual member function is an abstract base object type. Objects cannot be created of it. Declaration of the virtual member function is preceded by the keyword `virtual`. Declaration of pure virtual member function is appended by `= 0;`. This means that member function remains without implementation at all in the abstract base object type but can be redefined in derived object types. Here, the destructor function `~component` is declared as virtual while the function `displayImage` is declared as pure virtual.

New object types `monomer` and `polymer` define behaviour of primitive components in the composition. They derive from the abstract base type `component` and redefine the pure virtual function `displayImage` respectively.

The new object type `composite` defines behaviour for components that can contain both primitive and composite components. It too derives from the abstract base type `component` and redefines the pure virtual function `displayImage`. Additionally, it contains member function `addComponent` and `eraseComponent`. Its member object `pComponents` is of the object type `list`.

Info

Object type `list`
`list` is an object type defined in the file `list` of the Standard C++ Library. It is specifically designed for construction of objects that can hold/contain a pool of other objects and for operation on them. Elements can be efficiently inserted or deleted anywhere in the `list` container.

Note!

Object type `list`
In Standard C++ Library, the object type `list` is defined as a template. A template takes other object types as parameters.

Here, the object type `list` uses object type `component *` as template parameter. The object type `component *` is not ordinary object type. It is a pointer.

Info	**Pointer** A pointer is an object type whose object directly refers to (points to) another object stored elsewhere in the memory using its address. Pointer declaration contains specification of the object type the pointer is going to point to (here, `component`) and an asterisk sign `*`. The pointer can be used to directly access object pointed by this pointer. To do this, the pointer identifier/name must be preceded by the asterisk sign `*` (dereference operator).
Note!	**Pointer** The asterisk sign (`*`) is used both by the declaration of the pointer and as the dereference operator.

Thus, the member object and member functions of the new object type `composite` have to do with the pointer to the abstract base object type `component`.

Info	**Pointer to base object type** A pointer to base object type is compatible with a pointer to derived object type. This simple feature becomes especially useful when combined with virtual member functions. Virtual member functions of derived object types with the same name as one in the base object type can be appropriately called from the pointer to the base object type. Although an abstract base object type cannot instantiate its objects, pointer to it can be created and has all advantages of this polymorphic behaviour.

Instead of using whole number as counter variable, iterators are used in iterative control structure `for`.

Info	**Iterator** An iterator is designed for each container type such as `vector`, `deque`, `list`, etc. to iterate through its elements in an efficient way. The iterator of the object type `iterator` can iterate over elements in the container in read/write mode while the iterator of the object type `const_iterator` can iterate over elements in read-only mode. Function `begin` returns an iterator referring to the first element in the container. Function `end` returns an iterator referring to the past-the-end element in the container.
Note!	**Iterator** Operator `*` returns the element of the actual position. If the elements have member objects and member functions, operator `->` can be used to access those members directly from the iterator. In line 187, both operators are used to call member function `displayImage`.

The new object type `compositor` manipulates components in the composition. Its member functions `composeComposite1` and `composeComposite2` contain declarations of pointers to objects of object types `monomer`, `polymer`, and `composite` used to allocate dynamic memory necessary to contain one single object of the corresponding object type.

Info	**Dynamic memory allocation** In order to allocate memory dynamically, the operator `new` is used. If `T` is an object type, the expression `new T` yields the address to the (unnamed) newly allocated object of the object type `T`. This address can be assigned to the previously declared pointer `p` to the object of the object type `T`. By dynamic memory allocation, the necessary amount of memory will be determined during runtime. The object stays around until the program either ends or executes `delete p`. The expression `delete p` releases the dynamically allocated memory.
Note!	**Dynamic memory allocation** A call to `delete` must be made for every call to `new` to avoid a memory leak. After calling `delete` the object pointed to is invalid and should no longer be used. Therefore, `0` is usually assigned to the pointer `p` after execution of the expression `delete p`.

The new object type `compositor` and its member functions `composeComposite1` and `composeComposite2` significantly lighten the code for the new object types `cell` and `cp`. They encapsulate how the complex parts of the cell content get composed since they contain all the code to compose these parts. This code is easy to change and extend. This gives a lot of flexibility in what gets composed, how it gets composed, and when.

Consequently, the number of member objects in the new object type `cell` becomes significantly reduced. Respectively, its constructor function now has not more than two parameters. At the same time, the parameter list and the body of the constructor function become independent from the number of genes to be expressed.

To improve documenting, the new object type `cell` obtains an additional member object with the name `Number`. This member object is declared as static using keyword `static`.

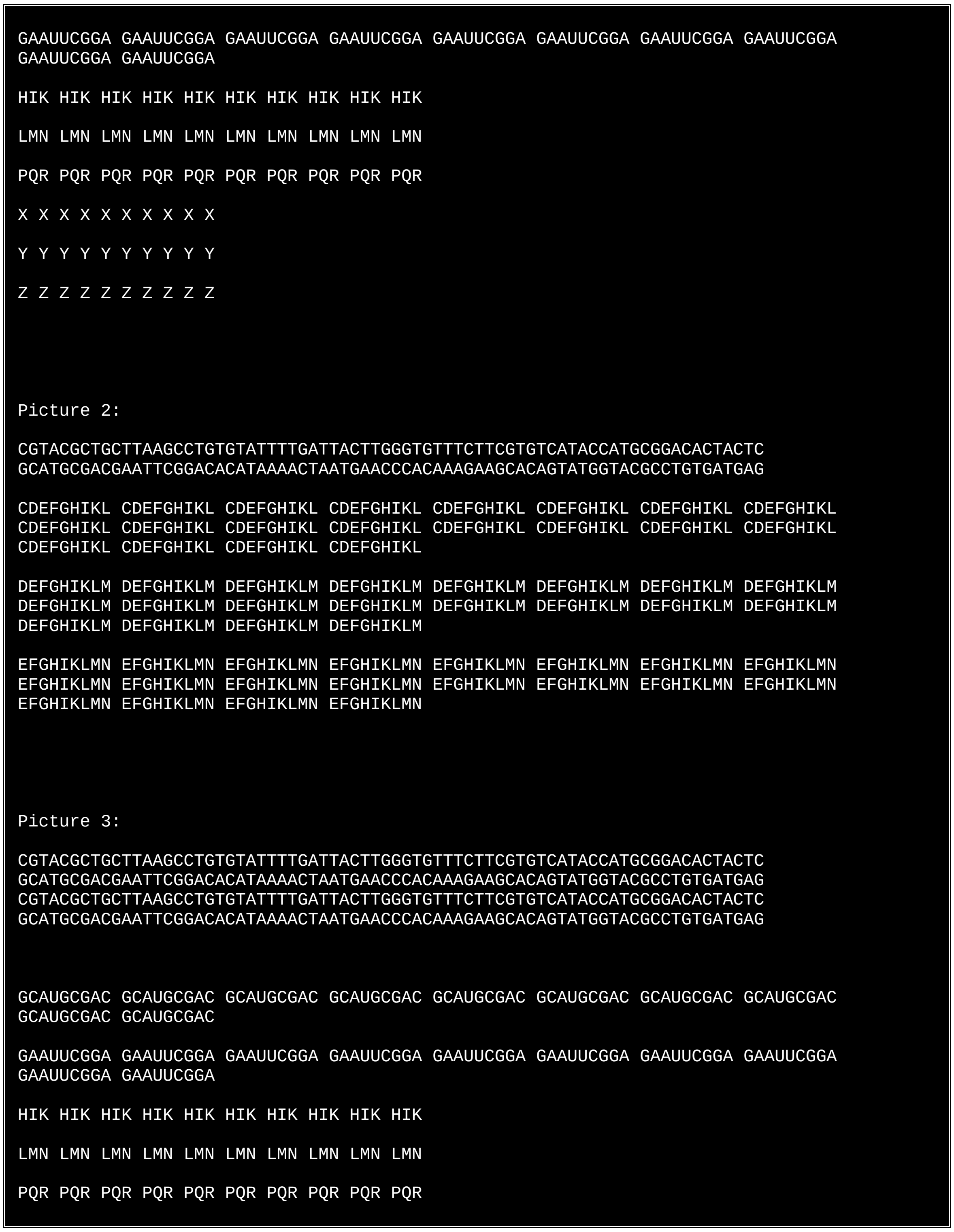
Info	**Static member object** If member object is declared to be static within any object type definition, there is a single instance of this member object for the entire object type regardless of how many objects of the corresponding object type will be constructed. All constructed objects of this object type will share this static member object.
Note!	**Static member object** Because the static member object is not part of the individual objects, it must be explicitly initialized outside the corresponding object type definition as shown in line 319.

Here, the static member object `Number` is used to count number of cells by their multiplication within the cell progression `cp`.

Experiment4-2: Building and executing machine code

Build Experiment4-2 and run the executable machine code.

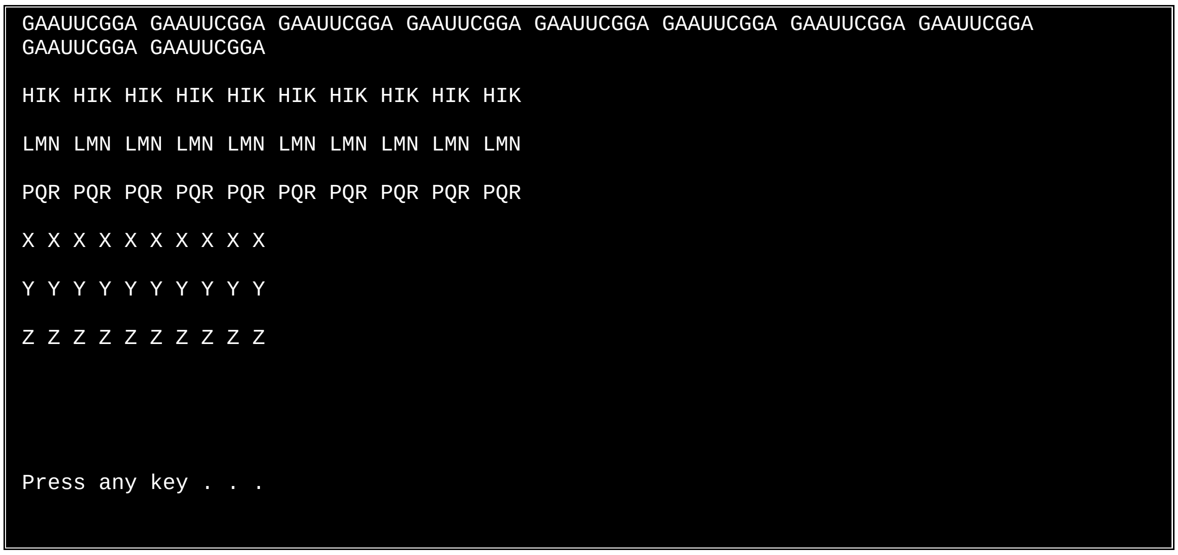
The console window will display pictures documenting how the new experiment for *in silico* genome multiplication proceeds (here, the short snippet):



```
GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA GAAUUCGGA
GAAUUCGGA GAAUUCGGA

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
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

CDEFGHIKL CDEFGHIKL CDEFGHIKL CDEFGHIKL CDEFGHIKL CDEFGHIKL CDEFGHIKL CDEFGHIKL
CDEFGHIKL CDEFGHIKL CDEFGHIKL CDEFGHIKL CDEFGHIKL CDEFGHIKL CDEFGHIKL CDEFGHIKL
CDEFGHIKL CDEFGHIKL CDEFGHIKL CDEFGHIKL

DEFGHIKLM DEFGHIKLM DEFGHIKLM DEFGHIKLM DEFGHIKLM DEFGHIKLM DEFGHIKLM DEFGHIKLM
DEFGHIKLM DEFGHIKLM DEFGHIKLM DEFGHIKLM DEFGHIKLM DEFGHIKLM DEFGHIKLM DEFGHIKLM
DEFGHIKLM DEFGHIKLM DEFGHIKLM DEFGHIKLM

EFGHIKLMN EFGHIKLMN EFGHIKLMN EFGHIKLMN EFGHIKLMN EFGHIKLMN EFGHIKLMN EFGHIKLMN
EFGHIKLMN EFGHIKLMN EFGHIKLMN EFGHIKLMN EFGHIKLMN EFGHIKLMN EFGHIKLMN EFGHIKLMN
EFGHIKLMN EFGHIKLMN EFGHIKLMN EFGHIKLMN




Picture 3:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG
CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG



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

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




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

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




Cell 6

Picture 1:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

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

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
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

CDEFGHIKL CDEFGHIKL CDEFGHIKL CDEFGHIKL CDEFGHIKL CDEFGHIKL CDEFGHIKL CDEFGHIKL
CDEFGHIKL CDEFGHIKL CDEFGHIKL CDEFGHIKL CDEFGHIKL CDEFGHIKL CDEFGHIKL CDEFGHIKL
CDEFGHIKL CDEFGHIKL CDEFGHIKL CDEFGHIKL

DEFGHIKLM DEFGHIKLM DEFGHIKLM DEFGHIKLM DEFGHIKLM DEFGHIKLM DEFGHIKLM DEFGHIKLM
DEFGHIKLM DEFGHIKLM DEFGHIKLM DEFGHIKLM DEFGHIKLM DEFGHIKLM DEFGHIKLM DEFGHIKLM
DEFGHIKLM DEFGHIKLM DEFGHIKLM DEFGHIKLM
```

```
EFGHIKLMN EFGHIKLMN EFGHIKLMN EFGHIKLMN EFGHIKLMN EFGHIKLMN EFGHIKLMN EFGHIKLMN
EFGHIKLMN EFGHIKLMN EFGHIKLMN EFGHIKLMN EFGHIKLMN EFGHIKLMN EFGHIKLMN EFGHIKLMN
EFGHIKLMN EFGHIKLMN EFGHIKLMN EFGHIKLMN




Picture 3:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG
CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG



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

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




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

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




Cell 7

Picture 1:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

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

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
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG

CDEFGHIKL CDEFGHIKL CDEFGHIKL CDEFGHIKL CDEFGHIKL CDEFGHIKL CDEFGHIKL CDEFGHIKL
CDEFGHIKL CDEFGHIKL CDEFGHIKL CDEFGHIKL CDEFGHIKL CDEFGHIKL CDEFGHIKL CDEFGHIKL
CDEFGHIKL CDEFGHIKL CDEFGHIKL CDEFGHIKL

DEFGHIKLM DEFGHIKLM DEFGHIKLM DEFGHIKLM DEFGHIKLM DEFGHIKLM DEFGHIKLM DEFGHIKLM
DEFGHIKLM DEFGHIKLM DEFGHIKLM DEFGHIKLM DEFGHIKLM DEFGHIKLM DEFGHIKLM DEFGHIKLM
DEFGHIKLM DEFGHIKLM DEFGHIKLM DEFGHIKLM

EFGHIKLMN EFGHIKLMN EFGHIKLMN EFGHIKLMN EFGHIKLMN EFGHIKLMN EFGHIKLMN EFGHIKLMN
EFGHIKLMN EFGHIKLMN EFGHIKLMN EFGHIKLMN EFGHIKLMN EFGHIKLMN EFGHIKLMN EFGHIKLMN
EFGHIKLMN EFGHIKLMN EFGHIKLMN EFGHIKLMN




Picture 3:

CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG
CGTACGCTGCTTAAGCCTGTGTATTTTGATTACTTGGGTGTTTCTTCGTGTCATACCATGCGGACACTACTC
GCATGCGACGAATTCGGACACATAAAACTAATGAACCCACAAAGAAGCACAGTATGGTACGCCTGTGATGAG



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

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




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

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




Press any key . . .
```

How to construct life *in silico*

© 2011 Nikita Tirjatkin