Шаблони функцій. Шаблони класів.

Шаблони функцій

Шаблон функції дозволяє визначити сімейство функцій. Це сімейство характеризується загальним алгоритмом, який може застосовуватися до даних різних типів.

Об’явлення і визначення шаблону функції починається ключовим словом **template**, за яким слідує у кутових дужках і розділений комами писок параметрів шаблону. Ця частина об’явлення називається заголовком шаблону.

Кожен параметр шаблона складається з службового слова **class**, за яким йде ідентифікатор. У контексті об’явлення шаблону функції службове слово class не несе ніякої особливого смислового навантаження.

За заголовком шаблону розташовується прототип функції.

Шаблон функції служить інструкцією для транслятора. З цієї інструкції транслятор може самостійно побудувати визначення нової функції.

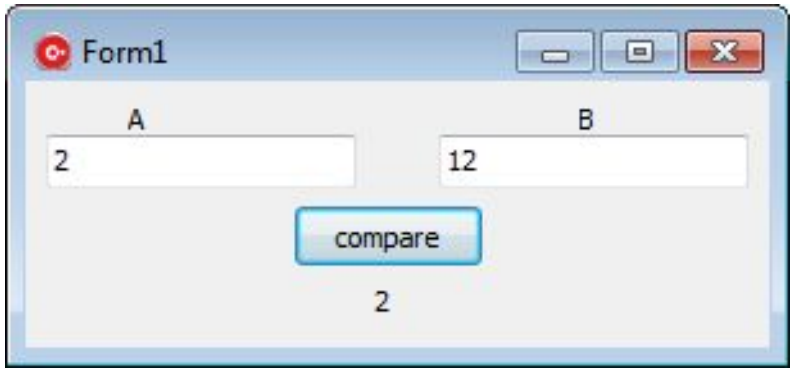
Параметри шаблону в шаблонних параметрах функції позначають місця майбутньої підстановки, яку здійснює транслятор в процесі побудови функції. Область дії параметрів шаблону обмежується шаблоном. Тому в різних шаблонах дозволено використання одних і тих же ідентифікаторів-імен параметрів шаблону.

Шаблони функцій

Приклад:

```
//Десь у заголовочному файлі
template <class Type>

Type minimum (Type a, Type b){
  if(a < b) return a;
  else return b;
}

//Десь у програмі……..

void __fastcall TForm1::Button1Click(TObject *Sender){
  int a = StrToInt(Edit1->Text);
  int b = StrToInt(Edit2->Text);
  Label3->Caption = minimum(a,b);        // виклик функції
}
```

Тип змінних, що підставляються до шаблону може бути **будь-яким**



Шаблони функцій

Якщо у програмі зустрінуться, наприклад, виклики функції з наступними типами аргументів:

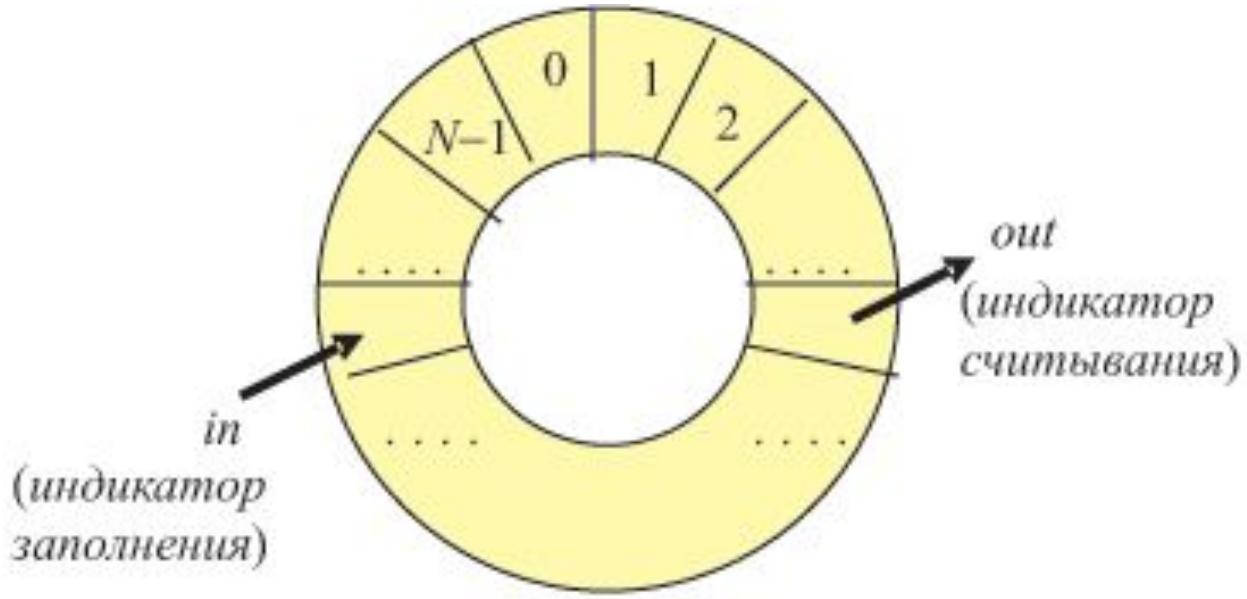
```
minimum(10, 20);       // цілий тип
minimum(10.0, 20.0);       // тип з плаваючою точкою,
```

то компілятор побудує окремі функції:

```
int minimum(int a, int b);
float minimum(float a, float b);
```

У виклику функції типи параметрів повинні збігатися !!!

Шаблони класів. Приклад: Кільцевий Буф



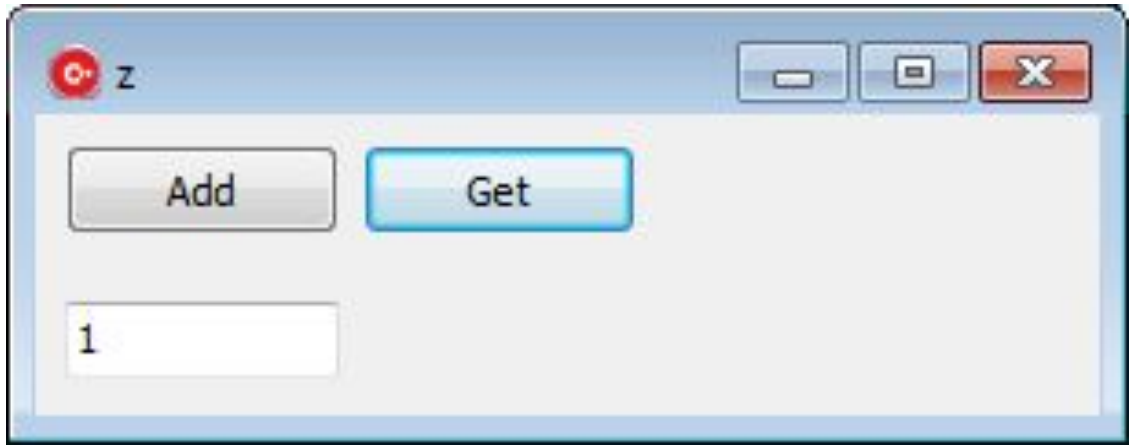
Циклічний буфер або **кільцевий буфер** - це структура даних, яка має фіксований розмір і використовується так ніби кінець буферу і початок замкнені в кільце, тобто при досягненні кінця буфера знов переміщуються в його початок. Така структура дає можливість здійснювати буферизацію потоків даних.

Шаблони класів. Приклад: Кільцевий Буфер.

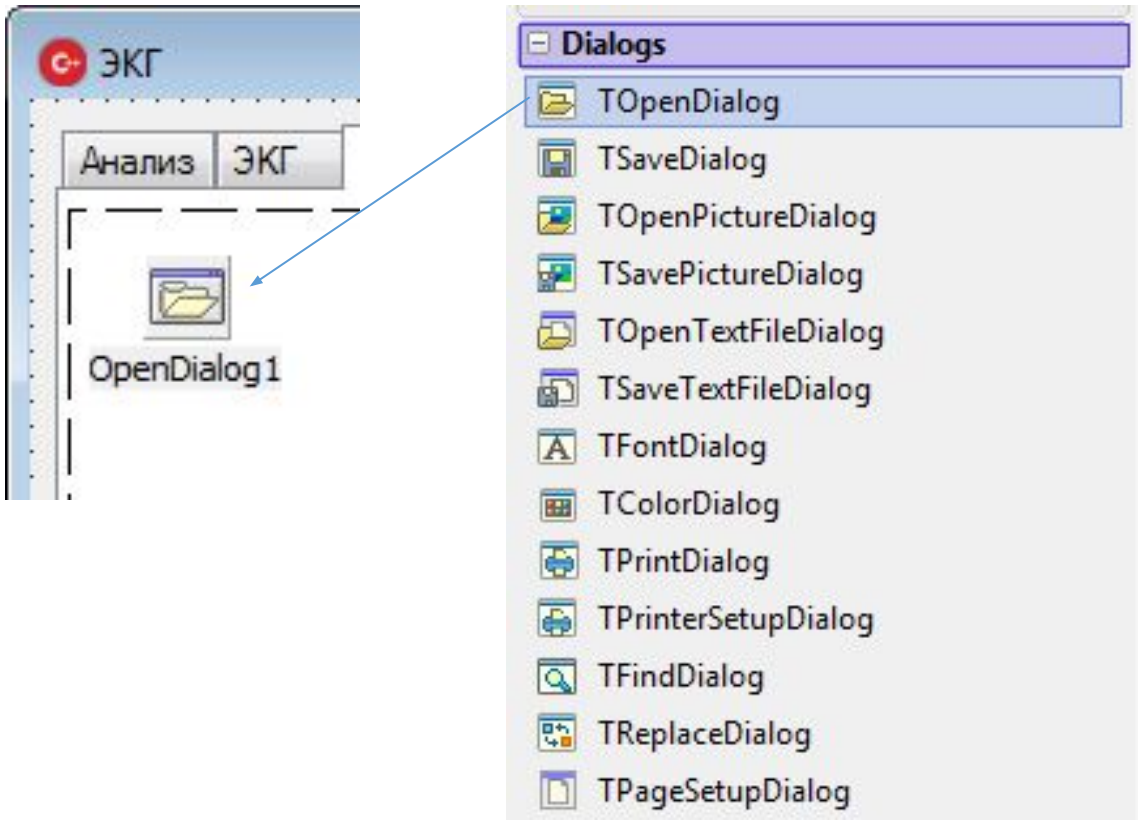
```
template <typename T>
class CircularBuffer{
  private:
      T Buff[16];          // буфер
      char WritePoint; // Точка записи
      char ReadPoint;  // Точка чтения

  public:
      char Count;  // Соличество элементов
      CircularBuffer(){
        WritePoint = 0;
        ReadPoint = 0;
        Count = 0;
      };
      ~CircularBuffer(){}; // деструктор

      void WriteData(T data){
        if(WritePoint == 16) WritePoint = 0;
        Buff[WritePoint++] = data;
```

Шаблони класів. Приклад: Кільцевий Буфер.

```
//Тут данные добавляются в буфер
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if(cb.Count < 16){
        char z = random(255);
        Caption = z;
        cb.WriteData(z);
  }
  Edit1->Text = IntToStr(cb.Count);
}
//Тут данные извлекаются из буфера
void __fastcall TForm1::Button2Click(TObject *Sender){

  if(cb.Count > 0){
        Caption = cb.ReadData();

  }
  Edit1->Text = IntToStr(cb.Count);
}
```



Робота з файлами.



Для вибору або відкриття файлу зручно використовувати компонент OpenDialog. Щоб отримати ім'я файлу:

```
OpenDialog1->Execute();
String fname = OpenDialog1->FileName;
```

Робота з файлами.

Для роботи з файлами необхідно підключити бібліотеку для виконання операцій введення / виводу:

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

Базові функції для роботи з файлами →

Direct input/output:

fread

Read block of data from stream (function)

fwrite

Write block of dat to stream (function)

File positioning:

fgetpos

Get current position in stream (function)

fseek

Reposition stream position indicator (function)

fsetpos

Set position indicator of stream (function)

ftell

Get current position in stream (function)

rewind

Set position of stream to the beginning (function)

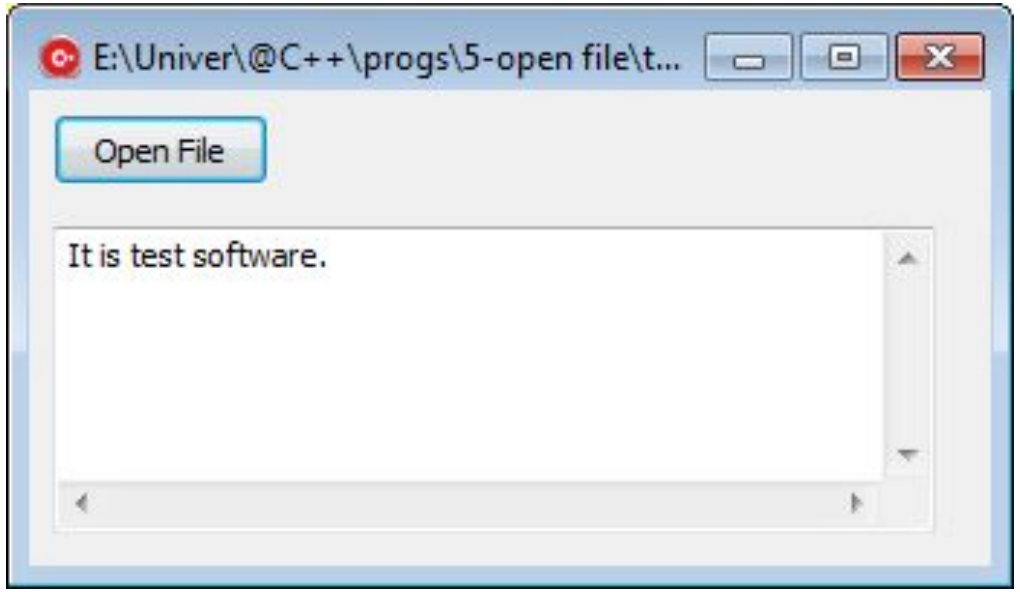
http://www.cplusplus.com/

http://docwiki.embarcadero.com/

Робота з файлами.

```
#include <stdio.h>

void __fastcall TForm1::Button1Click(TObject *Sender)
{
  OpenDialog1->Execute();
  AnsiString option, fname  = OpenDialog1->FileName;
  FILE *f;
  long fSize;
  option = "r";
  f = fopen(fname.c_str() , option.c_str() );
  if (f != NULL){
        Caption = fname;
        // obtain file size:
        fseek (f , 0 , SEEK_END);
        fSize = ftell (f);
        char *MC = new char [fSize];
        fseek (f , 0 , 0);
        fread(MC,fSize,fSize,f);
        String S;
```

```
for(int i = 0; i < fSize; i++){
          S+=MC[i];
        }
        Memo1->Lines->Add(S);
        fclose (f);
  }
}
```

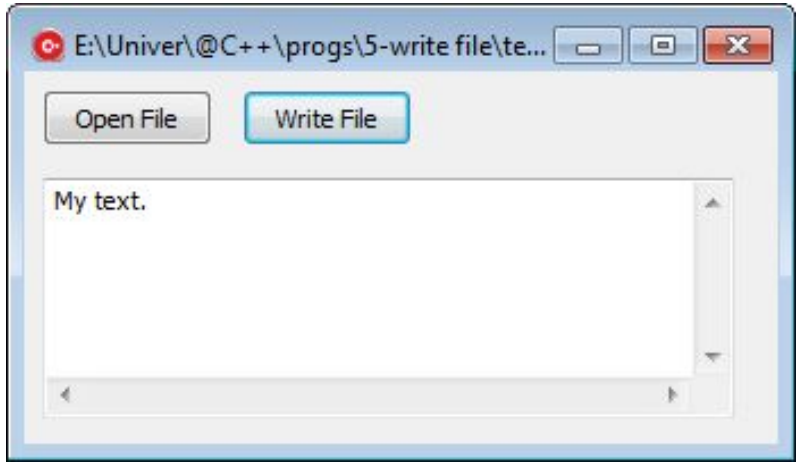


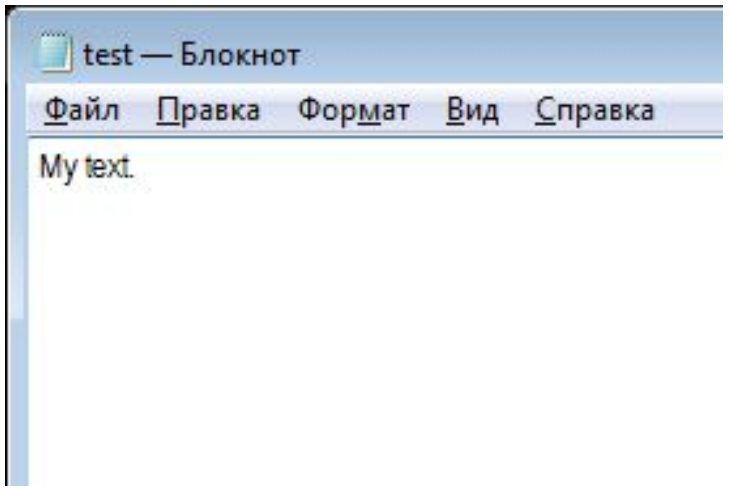
FILE * fopen (const char * filename, const char * mode);

"r"	**read:** Open file for input operations. The file must exist.
"w"	**write:** Create an empty file for output operations. If a file with the same name already exists, its contents are discarded and the file is treated as a new empty file.
"a"	**append:** Open file for output at the end of a file. Output operations always write data at the end of the file, expanding it. Repositioning operations (fseek, fsetpos, rewind) are ignored. The file is created if it does not exist.
"r+"	**read/update:** Open a file for update (both for input and output). The file must exist.
"w+"	**write/update:** Create an empty file and open it for update (both for input and output). If a file with the same name already exists its contents are discarded and the file is treated as a new empty file.
"a+"	**append/update:** Open a file for update (both for input and output) with all output operations writing data at the end of the file. Repositioning operations (fseek, fsetpos, rewind) affects the next input operations, but output operations move the position back to the end of file. The file is created if it does not exist.

Робота з файлами. Запис у файл.

```
#include <stdio.h>
 AnsiString option, fname  ;

void __fastcall TForm1::Button1Click(TObject *Sender){
  OpenDialog1->Execute();                 //вызов диалогового окна
  fname = OpenDialog1->FileName;          //присвоение имени файла
}

void __fastcall TForm1::Button2Click(TObject *Sender){
  FILE *f;                          //указатель на файл
  AnsiString text;                          //текст для записи
  option = "r+";                            //открытие с записью
  f = fopen(fname.c_str() , option.c_str() );    //открываем файл
  if (f != NULL){                           //если все ок продолжаем
        Caption = fname;
        fseek(f , 0 , 0);                   //указатель в начало файла
        text = Memo1->Text;                       //считываем текст из мемо
        fwrite(text.c_str(),1,text.Length(),f);     //пишем в файл
        fclose (f);                         //закрываем файл
   }
}
```





Обробка виняткових ситуацій.

Існує категорія помилок, які не здатні виявити препроцесори, транслятори і програми збірки. До їх числа відносяться так звані помилки часу виконання. Ці помилки виявляються в ході виконання програми.

Помилки часу виконання, що виникають безпосередньо в ході виконання програми, в термінах об'єктно-орієнтованого програмування називаються винятковими ситуаціями. Виняткові ситуації - це події, які переривають нормальний хід виконання програми.

Реакція на виняткову ситуацію називається винятком.

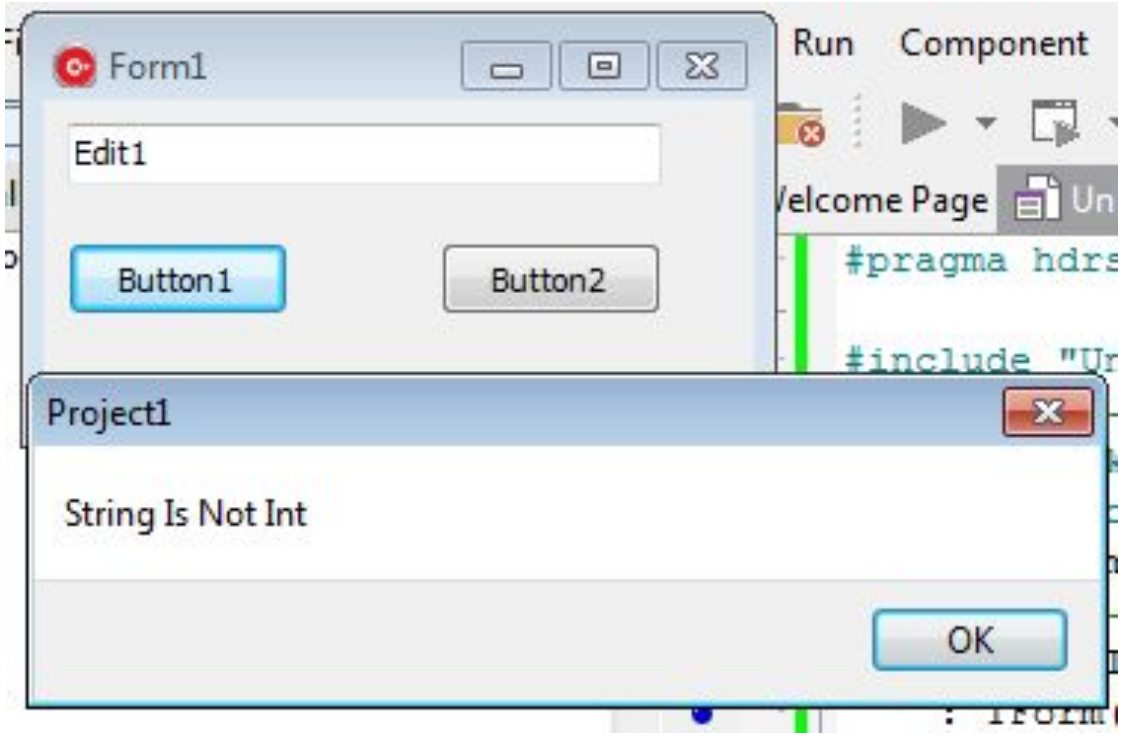
У С ++ є вбудовані засоби для їх збудження і обробки. За допомогою цих засобів активізується механізм, що дозволяє двом непов'язаним фрагментами програми обмінюватися інформацією про виключення.

Генерація виключення забезпечується операцією **throw**. Оператор збудження виключення є повноправним оператором і в принципі може розташовуватися в будь-якому місці програми. Його виконання за межами контрольованого блоку призводить до завершення процесу виконання програми.

Обробка виняткових ситуацій.

Зазвичай генератор виключення використовується в поєднанні з **try-блоком** (блок, що контролюється). Їх взаємодія забезпечується через стек виклику. **Тому точка генерації виключення повинна розташовуватися в тілі функції, безпосередньо або побічно викликається з безлічі операторів даного try-блоку.** Такий блок починається з ключового слова try, за яким йде послідовність інструкцій, укладена в фігурні дужки, а після цього - список обробників, званих catch-пропозиціями. Try-блок групує інструкції програми і асоціює з ними обробники винятків.

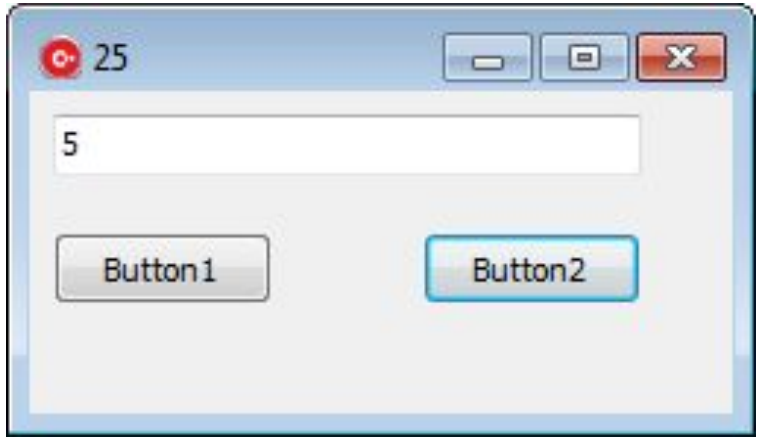
```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int a;

  try {
      a = StrToInt(Edit1->Text);
      Caption = a*a;
  } catch( EConvertError &ex ) {
      a = 0;
      ShowMessage("String Is Not Int");
  }
}
```



Обробка виняткових ситуацій.

Часто треба запобігти помилці вводу. Зручно використовувати функцію TryStrToInt(string, int)

```
void __fastcall TForm1::Button2Click(TObject *Sender)
{

int i;
if (TryStrToInt(Edit1->Text,i)){
      ShowMessage("OK");
      Caption = i*i;
}else{
  ShowMessage("String Is Not Int");
}
}
```



Розв'язання диференційного рівняння. Метод Ейлера

Решение дифференциального уравнения методом Эйлера.

Написать программу, находящую решение дифференциального уравнения $y'=f(x,y)$ методом Эйлера на отрезке $[x_0,x_n]$ с шагом h при начальных условиях $y=y_0$. Исходные данные приведены.

Метод Эйлера и вычисление функции $f(x,y)$ необходимо реализовать в виде отдельных функций. Исходные данные y_0, x_0, x_n, h вводятся с клавиатуры.

Значения численного решения рассчитываются по формуле

$y_{i+1}=y_i+h*f(x_i,y_i)$**Исходные данные:**

Дифференциальное уравнение: y'=cos(x) - y

$y_0 = 0$

$x_0 = 0$

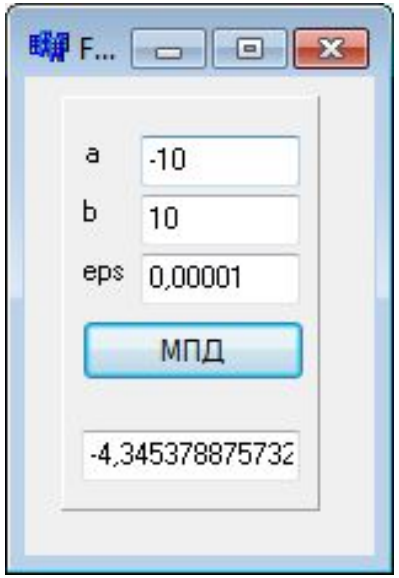
$x_n = 1$

```
float f(float x, float y){
  return cos(x)-y;
}

void __fastcall TForm1::Button3Click(TObject *Sender){
  double h = StrToFloat(Edit5->Text);
  double x0 = StrToFloat(Edit6->Text);
  double y0 = StrToFloat(Edit7->Text);
  double b = StrToFloat(Edit8->Text);
  double y;
  Memo2->Clear();
  for(float i = x0; i < b; i+=h){
      y = y0 + h*f(i, y0) ;
      Series1->AddXY(i,y,"",clRed);
      Series2->AddXY(i,0.5*(sin(i) + cos(i) - exp(-i)),"",clGreen);
      Memo2->Lines->Add(FloatToStr(i)+ "; " +  FloatToStr(y));
      y0 = y;
  }
}
```

Розв'язання рівнянь. Метод половинного ділення.

$$x^3 + sin(x) - x^2 + 100 = 0$$

Таким чином, якщо ми шукаємо нуль, то на кінцях відрізка функція повинна бути протилежних знаків. Розділимо відрізок навпіл і візьмемо ту з половинок, на кінцях якої функція як і раніше приймає значення протилежних знаків. Якщо значення функції в серединній точці виявилося потрібним нулем, то процес завершується.




```
float f(float x){
  return pow(x,3)+sin(x)-pow(x,2)+100;
}

void __fastcall TForm1::Button2Click(TObject *Sender){
 float a = StrToFloat(Edit1->Text);
 float b = StrToFloat(Edit2->Text);
 float eps = StrToFloat(Edit3->Text);
 float c, e;
 while((b-a)/2> eps ){
   c = (a+b)/2;
   if(f(a)*f(c) < 0){
         b = c;
   }else{
         a=c;
   }
 }
 Edit4->Text = FloatToStr(c);
}
```