"The study of methods of optimization of the program code at the cost of CPU time"

Process optimization code

- Optimization - a transformation in which the result of the system remains unchanged, but improve some of its characteristics. The most common goals of optimization when developing the code: reduce the execution time of programs, improve performance, compactification of code, memory consumption, minimizing energy consumption, reducing the number of input / output.

Integrated methods for optimization of programs

Constant folding («свертывание констант»)

One of the common optimizations in the compiler is "constant folding". In the process of performing this action in program code are the constants in the resulting code consists of the calculated values.

До оптимизации	*После оптимизации*

`#include <stdlib.h>`
`int main(int argc, char **argv)`
`{`
`  struct point`
`  {`
`  int x;`
`  int y;`
`  } p;`
`  int a = 32*32;`
`  int b = 32*32*4;`
`  long int c;`
`  //`
`  c = (a + b) * (4*4*sizeof(p) - 2 + 32);`
`  return 0;`
` }
	`#include <stdlib.h>`
	`int main(int argc, char **argv)`
	`{`
	`  struct point`
	`  {`
	`  int x;`
	`  int y;`
	`  } p;`
	`  int a = 1024; // Свёрнуто из 32 * 32`
	`  int b = 4096; // Свёрнуто из 32 * 32 * 4`
	`  long int c;`
	`  // 16 = 4*4, 30 = -2 + 32`
	`  c = (a + b) * (16*sizeof(p) + 30);`
	`  return 0;`
	` }

Integrated methods for optimization of programs

Common Sub-Expression Elimination (CSE) (Устранение общих подвыражений)

This optimization is the following: if you use the calculation of any expression two or more times, it can be calculated once and then to substitute all uses of its expression.

До оптимизации	*После оптимизации*

`int calc(int x, int y)`
`{`
`int a = (x + y) * (x - y) - x * y;`
`int b = x * (x + y) - y * (x - y);`
`return (a * b + x - y) * (a * b + x + y);`
`}`
	`int calc(int x, int y)`
	`{`
	`int tmp1 = x + y;`
	`int tmp2 = x - y;`
	`int a = tmp1 * tmp2 - x * y;`
	`int b = x * tmp1 - y * tmp2;`
	`int tmp3 = a * b;`
	`return (tmp3 + tmp2) * (tmp3 + tmp1);`
	`}`

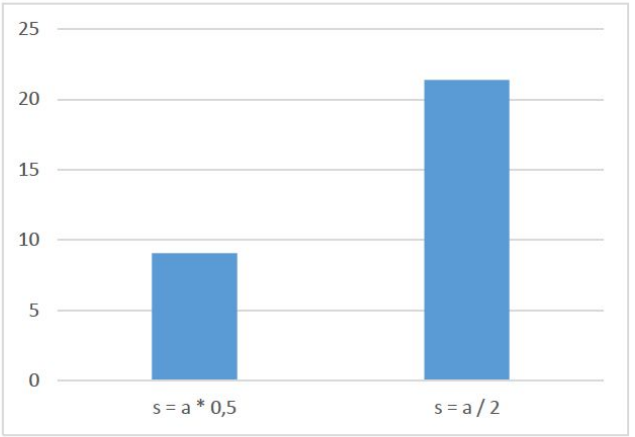
Performing arithmetic operations on variables of different types

Comparison of execution time of commands s = a*0.5 and s = a/2 The aim of the study was to determine which of the basic arithmetic operations on variables of different types are faster. Also need to find out how the performance of an operation depends on the type of the operands.

```
var
  i, j, k, n0, n1: integer;
  a: integer;
  s: real;
begin
  a := 100;
  n0 := Milliseconds;
  for i := 1 to 1500 do
    for j := 1 to 1500 do
      for k := 1 to 1500 do
        s := a *0.5;
  n1 := Milliseconds;
  writeln((n1 - n0)/1000);
end.
```

```
var
  i, j, k, n0, n1: integer;
  a: integer;
  s: real;
begin
  a := 100;
  n0 := Milliseconds;
  for i := 1 to 1500 do
    for j := 1 to 1500 do
      for k := 1 to 1500 do
        s := a / 2;
  n1 := Milliseconds;
  writeln((n1 - n0)/1000);
end.
```

Номер опыта	s = a * 0,5	s = a / 2
1	9,069	21,386
2	9,072	21,389
3	9,088	21,381
4	9,060	21,372
5	9,067	21,391
6	9,071	21,385
7	9,102	21,386
8	9,109	21,407
9	9,058	21,376
10	9,066	21,381
Ср. значение	**9,076**	**21,385**



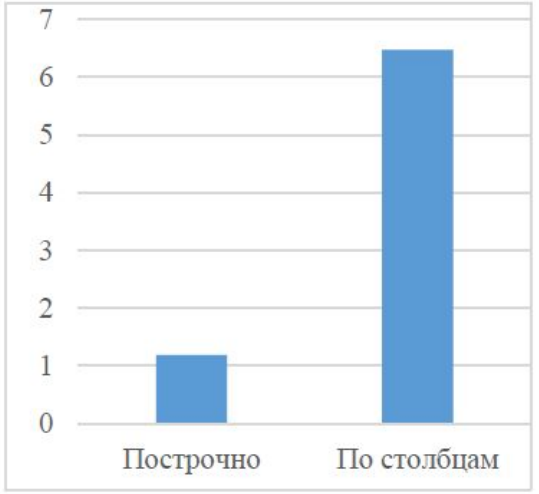
Fill the array

The aim of the study was to test the effect of the method of passage when processing two – dimensional arrays by rows or by columns.Intended result: Since the array dimensions are the same, and the number of elementary operations when working with the cells of the array is also the same, then the difference in the way traverse the array should not occur.

```
type
  TMassiv = array[0..9999,
0..9999] of Integer;
var
  Mas: TMassiv;
  I, J, T: Integer;
  n0, n1: integer;
begin
  n0 := Milliseconds;
  for i := 0 to 9999 do
    for j := 0 to 9999 do
      Mas[I, J] := 1;
  n1 := Milliseconds;
  writeln((n1 - n0)/1000);
end.
```

```
type
  TMassiv = array[0..9999,
0..9999] of Integer;
var
  Mas: TMassiv;
  I, J, T: Integer;
  n0, n1: integer;
begin
  n0 := Milliseconds;
  for j := 0 to 9999 do
    for i := 0 to 9999 do
      Mas[I, J] := 1;
  n1 := Milliseconds;
  writeln((n1 - n0)/1000);
end.
```

Номер опыта	Построчно	По столбцам
1	1,185	6,357
2	1,173	6,58
3	1,216	6,695
4	1,187	6,362
5	1,196	6,36
Ср. Значение	1,1914	6,4708



Branching statements

The aim of the study was to verify the herniation of the choice of branching statements (If-then-else statement and multiple-choice Case) on the speed of the program.

```
var
  a, k, n0, n1, i, j, q: integer;
begin
  n0 := milliseconds;
  for i := 1 to 1000 do
    for j := 1 to 20000 do
    begin
      a := random(1, 10);
      case a of
        1: k := 1;
        2: k := 2;
        3..5: k := 3;
        7: k := 4;
      else k := 0
      end;
    end;
  n1 := milliseconds;
  write((n1 - n0) / 1000);
end.
```

```
var
  a, k, n0, n1, i, j, q: integer;
begin
  n0 := milliseconds;
  for i := 1 to 1000 do
    for j := 1 to 20000 do
    begin
      a := random(1, 10);
      if a = 1 then k := 1
      else
      if a = 2 then k := 2
      else
      if a in [3..5] then k:=3
      else
      if a = 7 then k := 4
      else k := 0;
    end;
  n1 := milliseconds;
  write((n1 - n0) / 1000);
end.
```

Номер опыта	Case	Конструкция с If
1	0,929	11,618
2	0,894	11,674
3	0,935	11,761
4	0,887	11,859
5	0,894	11,787
Ср. Значение	0,9078	11,7398