

СУ “Св. Климент Охридски”
Факултет по Математика и Информатика

Програмиране на C/C++

*Материали за упражненията по „Структури
от данни и обектно-ориентирано
програмиране“ и „Структури от данни и
обектно-ориентирано програмиране - II“ за
специалност Компютърни науки*

Съставил Емилия Живкова

zivkova@fmi.uni-sofia.bg

2012 г.

Предговор

Авторът е използвал програмната система Moodle при разработката на следните курсове за студенти слушали дисциплината “Увод в програмирането” (с използване на Java), на специалност “Компютърни науки” 1 курс, седмичен хорариум 4+4 (2 часа семинарни и 2 часа лабораторни), лектор доц. д-р Д. Биров, зимен семестър на учебната 2010/2011 г.:

1. Курс „Обектно-ориентирано програмиране и структури от данни и програмиране - 2011, 4 и 5“. Упражнения по дисциплината „Структури от данни и обектно-ориентирано програмиране“ (с използване на С) на специалност “Компютърни науки” 1 курс, 4 и 5 групи, седмичен хорариум 4+4 (2 часа семинарни и 2 часа лабораторни), лектор доц. д-р Ст. Бъчваров, летен семестър на учебната 2010/2011 г.

2. Курс „Обектно-ориентирано програмиране и структури от данни и програмиране II - 2012, 4, 5 и 6“. Упражнения по дисциплината „Структури от данни и обектно-ориентирано програмиране - II“ (с използване на С++) на специалност “Компютърни науки” 2 курс, 4, 5 и 6 групи, седмичен хорариум 4+2 (2 часа семинарни), лектор доц. д-р Ст. Бъчваров, зимен семестър на учебната 2011/2012 г.

Материалите за курсовете са организирани, съгласно тематичните планове на дисциплините и включват:

- Текст за семинарното занятие (файл в PDF формат) за всяка седмица, съдържащ примерни програми, илюстриращи съответната тема, както и задачи по темата, които се обсъждат и решават
- Пълна или частична реализация на С/С++ (файлове в TXT формат) на разглежданите задачи по темата
 - Програмни шаблони на С/С++ (файлове в TXT формат) за самостоятелна работа
 - Задания (файлове в DOC формат) - задачи по темата, като файловете с реализациите на С/С++ се изпращат на асистента

При съставянето на този текст са използвани разработените от автора материали. В него са включени примерни програми, задачи, решавани по време на упражненията или предоставяни на студентите за самостоятелна работа, както и техните реализации на езиците С и С++.

E. Живкова

Съдържание

	I. Програмиране на C	
Тема 1.	Основни езикови средства. Константи, променливи и типове данни. Масиви и низове. Коментари, структура на програмата и етапи на нейната обработка. Форматиран вход и изход. Функции scanf() и printf()	1
Тема 2.	Операции и изрази. Аритметични операции. Операции за отношение и логически операции. Операция за присвояване. Изрази. Преобразуване на типовете. Операции за условен израз, за последователно изпълнение, за преобразуване на типовете и за размер на обект. Управляващи структури. Съставен и условен оператор. Оператори за цикъл. Многовариантен избор. Оператори за прекъсване на цикъл и за завършване на итерация на цикъл. Влагане на оператори	12
Тема 3.	Препроцесор. Принципи на работа на препроцесора. Директиви на препроцесора. Макроопределения. Макроопределения с аргументи. Включване на файлове. Условна компилация	22
Тема 4.	Функции. Дефиниция на функция. Параметри на функция. Връщани резултати. Обръщение към функция. Прототип на функция. Модули. Област на действие на променливите. Класове памет. Инициализиране на променливите	28
Тема 5.	Указатели. Основни операции. Адресна аритметика. Масиви и указатели. Предаване на резултати чрез формални параметри. Формални параметри и масиви. Указатели към функции. Функции с променлив брой параметри	40
Тема 6.	Структури. Основни операции. Вложени структури. Функции и структури. Структури и масиви. Обединения. Дефиниране имена на типове. Изброен тип	47
Тема 7.	Структури от данни. Родови (универсални, с общо предназначение) единици	51
Тема 8.	Файлове. Текстови и двоични файлове. Етапи за работа с файлове. Функции за буфериран вход и изход	68
Тема 9.	Списък, стек, опшка, дек	78
	II. Програмиране на C++	
Тема 10.	Обекти и класове. Предефиниране на операциите. Рационално число	106
Тема 11.	Обекти и масиви от обекти в свободната памет. Вектор	116
Тема 12.	Наследяване и полиморфизъм. Абстрактни класове. Хетерогенен сортиран линеен едносвързан списък. Матрица ...	129
Тема 13.	Функции шаблони. Класове шаблони. Итератор. Линеен списък. Сортиран линеен списък. Полином на една променлива с реални коефициенти	154

Тема 14.	Обобщено програмиране. Приоритетна опашка. Асоциативен масив. Множество	176
Тема 15.	Двоично дърво	202
	Литература	216

Основни езикови средства. Константи, променливи и типове данни. Масиви и низове. Коментари, структура на програмата и етапи на нейната обработка. Форматиран вход и изход. Функции `scanf()` и `printf()`

Програма Hello.c

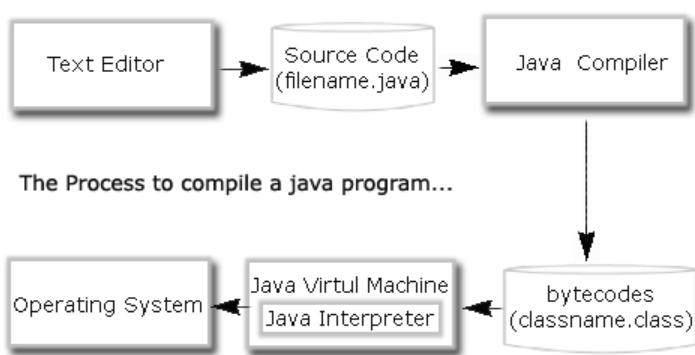
```
/*
 * Hello.c
 *
 * Created on: 27.02.2011
 *
 */
#include <stdio.h>

/* function main begins program execution */
int main(void) {
    printf("Welcome to C!\n");
    return 0; /* indicate that program ended successfully */
}
```

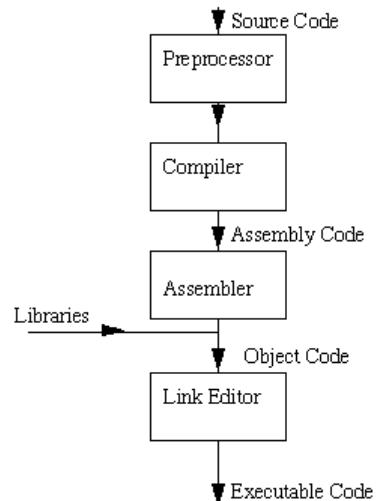
Изпълнение на програмата:

Welcome to C!

Обработка на програма на Java



Обработка на програма на C



	Java	C
Структура на програма	HelloWorld.java: <pre>public class HelloWorld { public static void main(String[] args) { System.out.println("Hello, World!"); } }</pre>	HelloWorld.c: <pre>#include <stdio.h> #include <stdlib.h> int main(void) { printf("Hello, World!\n"); return EXIT_SUCCESS; }</pre>
Транслиране	<code>javac HelloWorld.java</code>	<code>gcc ... HelloWorld.o ... HelloWorld.c</code> <code>gcc ... HelloWorld.exe ... HelloWorld.o</code>
Изпълнение	<code>java HelloWorld.class</code> <i>Hello, World!</i>	<code>Hello.exe</code> <i>Hello, World!</i>

Програма HelloWorld.c

```
/*
=====
Name      : HelloWorld.c
Version   : 03.2011
Description : C program Ansi-style structure and the stages of developing
              C program
=====
*/
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    puts("Hello, World!");
    return EXIT_SUCCESS;
}
```

Създаване и обработка на програмата в среда Wascana:

```
**** Build of configuration Debug for project HelloWorld ****

**** Internal Builder is used for build ****
gcc -O0 -g3 -Wall -c -fmessage-length=0 -osrc\HelloWorld.o ..\src\HelloWorld.c
gcc -oHelloWorld.exe src\HelloWorld.o
Build complete for project HelloWorld
Time consumed: 1052 ms.
```

За програмите в текста са използвани свободно разпространявани програмни средства, достъпни в Internet. Следващата таблица съдържа адреси на някои от страниците и кратко описание на тяхното съдържание.

Адрес	Съдържание
http://code.google.com/a/eclipselabs.org/p/wascana/	Среда за програмиране на C/C++ Eclipse за Windows, с използване на MinGW GNU
http://www.mingw.org/	Среда MinGW за Windows приложения
http://gcc.gnu.org/	GNU колекция от транслатори (GCC)

*Изпълнение на програмата **HelloWorld.exe** в Eclipse:*

Hello, World!

Програма CmdLineArgs.c

```
/*
=====
Name      : CmdLineArgs.c
Version   : 03.2011
Description : Program with command line arguments
=====
*/
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    printf("The program name is %s\n", argv[0]);
    if (argc == 1)
        printf("No program line arguments!\n");
    else {
        int i;
        for (i = 1; i < argc; i++)
            printf("Command line argument %d is %s\n", i, argv[i]);
    }
}
```

```

    }

    return EXIT_SUCCESS;
}

```

Изпълнение на програмата в Eclipse:

```

The program name is C:\Documents and Settings\Butterfly\My
Documents\C&CPPProjects\CmdLineArgs\Debug\CmdLineArgs.exe
No program line arguments!

```

*Изпълнение на програмата **CmdLineArgs.exe**, копирана на E: в директория **ExecutableFiles**, от Windows:*

След изпълнение на Start => Run => cmd:

```
C:\Documents and Settings\Butterfly>E:
```

```
E:>cd ExecutableFiles
```

```
E:\ExecutableFiles>CmdLineArgs
The program name is CmdLineArgs
No program line arguments!
```

```
E:\ExecutableFiles>CmdLineArgs arg1 arg2 arg3
The program name is CmdLineArgs
Command line argument 1 is arg1
Command line argument 2 is arg2
Command line argument 3 is arg3
```

```
E:\ExecutableFiles>exit
```

Програма Constants.c

```

/*
=====
Name      : Constants.c
Version   : 03.2011
Description : C constants
=====
*/
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    printf("    Integer constants:\n");
    printf("Decimal constant %d\n", 12);
    printf("Size of Decimal constant %d is %d\n", 12, sizeof(12));
    printf("Octal representation of decimal constant %d is %#o\n", 12, 12);
    printf("Hexadecimal representation of decimal constant %d is %#x\n", 12,
12);
    printf("Hexadecimal representation of octal constant %#o is %#x\n", 12, 12);
    printf("Hexadecimal representation of decimal constant %d is %#x\n", 014,
014);

    printf("    Floating-point constants:\n");
    printf("Constant %f\n", 1234.567890);
    printf("Size of constant %f is %d\n", 1234.567890, sizeof(1234.567890));
    printf("Constant %.1f\n", 1234.567890);
    printf("Constant %e\n", 1234.567890);
    printf("Constant %.3e\n", 1234.567890);
    printf("Constant %g\n", 1234.567890);
}

```

```

printf("      Character constants:\n");
printf("Constant '%c'\n", 'A');
printf("Size of constant '%c' is %d\n", 'A', sizeof('A'));
printf("Integer representation of constant '%c' is %d\n", 'A', 'A');
printf("Hexadecimal representation of constant '%c' is '\\x%x'\n", '\x41',
      '\x41');
printf("Octal representation of constant '%c' is '\\%o'\n", '\101', '\101');

printf("      String constants:\n");
printf("Constant %s\n", "Hello!");
printf("Size of constant %s is %d\n", "Hello!", sizeof("Hello!"));
printf("Size of empty string %s is %d\n", "", sizeof(""));

return EXIT_SUCCESS;
}

```

Изпълнение на програмата:

```

Integer constants:
Decimal constant 12
Size of Decimal constant 12 is 4
Octal representation of decimal constant 12 is 014
Hexadecimal representation of decimal constant 12 is 0xc
Hexadecimal representation of octal constant 014 is 0xc
Hexadecimal representation of decimal constant 12 is 0xc

      Floating-point constants:
Constant 1234.567890
Size of constant 1234.567890 is 8
Constant 1234.6
Constant 1.234568e+003
Constant 1.235e+003
Constant 1234.57

      Character constants:
Constant 'A'
Size of constant 'A' is 4
Integer representation of constant 'A' is 65
Hexadecimal representation of constant 'A' is '\x41'
Octal representation of constant 'A' is '\101'

      String constants:
Constant Hello!
Size of constant Hello! is 7
Size of empty string is 1

```

Типове	Java	C
Символен тип	char // 16-bit unicode	char /* 8 bits */
Целочислени типове	byte // 8 bits short // 16 bits int // 32 bits long // 64 bits	(unsigned) char (unsigned) short (unsigned) int (unsigned) long
Типове с плаваща точка	float // 32 bits double // 64 bits	float double long double
Логически тип	boolean	Няма. Използва се целочислен тип
Тип указател към произволен тип	Няма	void*
Константи	final int MAX = 10;	#define MAX 10 const int MAX = 10; enum {MAX = 10};

Програма Types.c

```
/*
=====
Name      : Types.c
Version   : 03.2011
Description : C types:
1. char, signed char
2. unsigned char
3. short, signed short, short int, signed short int
4. unsigned short, unsigned short int
5. int, signed, signed int
6. unsigned int, unsigned
7. long, signed long, long int, signed long int
8. unsigned long, unsigned long int
9. float
10.double
11.long double
=====
*/
#include <stdio.h>
#include <stdlib.h>
#include <float.h>
#include <limits.h>

int main(void) {
    //From limits.h
    printf("      Type unsigned char:\n");
    printf("Size of type unsigned char is %d\n", sizeof(unsigned char));
    printf("Max value for unsigned char is %d\n", UCHAR_MAX);

    printf("      Type char, signed char:\n");
    printf("Size of type char is %d\n", sizeof(char));
    printf("Min value for char is %d\n", SCHAR_MIN);
    printf("Max value for char is %d\n", SCHAR_MAX);

    printf("      Type short, signed short, short int, signed short int int:\n");
    printf("Size of type short is %d\n", sizeof(short));
    printf("Min value for short is %d\n", SHRT_MIN);
    printf("Max value for short is %d\n", SHRT_MAX);

    printf("      Type unsigned short, unsigned short int:\n");
    printf("Size of type unsigned short is %d\n", sizeof(unsigned short));
    printf("Max value for unsigned short is %d\n", USHRT_MAX);

    printf("      Type int, signed, signed int:\n");
    printf("Size of type int is %d\n", sizeof(int));
    printf("Min value for int is %d\n", INT_MIN);
    printf("Max value for int is %d\n", INT_MAX);

    printf("      Type unsigned int, unsigned:\n");
    printf("Size of type unsigned int is %d\n", sizeof(unsigned int));
    printf("Max value for unsigned int is %u\n", UINT_MAX);

    printf("      Type long, signed long, long int, signed long int:\n");
    printf("Size of type long is %d\n", sizeof(long));
    printf("Min value for long is %ld\n", LONG_MIN);
    printf("Max value for long is %ld\n", LONG_MAX);

    printf("      Type unsigned long, unsigned long int:\n");
    printf("Size of type unsigned long is %d\n", sizeof(unsigned long));
    printf("Max value for unsigned long is %lu\n", ULONG_MAX);

    //From float.h (mEexp)
    printf("      Type float:\n");
    printf("Size of type float is %d\n", sizeof(float));
    printf("Length of binary m is %d\n", FLT_MANT_DIG);
}
```

```

printf("Max length of decimal m is %d\n", FLT_DIG);
printf("Min value of binary exp is %d\n", FLT_MIN_EXP);
printf("Min value of decimal exp is %d\n", FLT_MIN_10_EXP);
printf("Max value of binary exp is %d\n", FLT_MAX_EXP);
printf("Max value of decimal exp is %d\n", FLT_MAX_10_EXP);
printf("Min value for float is %E\n", FLT_MIN);
printf("Max value for float is %E\n", FLT_MAX);

printf("      Type double:\n");
printf("Size of type double is %d\n", sizeof(double));
printf("Length of binary m is %d\n", DBL_MANT_DIG);
printf("Max length of decimal m is %d\n", DBL_DIG);
printf("Min value of binary exp is %d\n", DBL_MIN_EXP);
printf("Min value of decimal exp is %d\n", DBL_MIN_10_EXP);
printf("Max value of binary exp is %d\n", DBL_MAX_EXP);
printf("Max value of decimal exp is %d\n", DBL_MAX_10_EXP);
printf("Min value for double is %E\n", DBL_MIN);
printf("Max value for double is %E\n", DBL_MAX);

printf("      Type long double:\n");
printf("Size of type long double is %d\n", sizeof(long double));
printf("Length of binary m is %d\n", LDBL_MANT_DIG);
printf("Max length of decimal m is %d\n", LDBL_DIG);
printf("Min value of binary exp is %d\n", LDBL_MIN_EXP);
printf("Min value of decimal exp is %d\n", LDBL_MIN_10_EXP);
printf("Max value of binary exp is %d\n", LDBL_MAX_EXP);
printf("Max value of decimal exp is %d\n", LDBL_MAX_10_EXP);
//printf("Min value for long double is %???\n", LDBL_MIN);
//printf("Max value for long double is %???\n", LDBL_MAX);

return EXIT_SUCCESS;
}

```

Изпълнение на програмата:

```

Type unsigned char:
Size of type unsigned char is 1
Max value for unsigned char is 255

Type char, signed char:
Size of type char is 1
Min value for char is -128
Max value for char is 127

Type short, signed short, short int, signed short int int:
Size of type short is 2
Min value for short is -32768
Max value for short is 32767

Type unsigned short, unsigned short int:
Size of type unsigned short is 2
Max value for unsigned short is 65535

Type int, signed, signed int:
Size of type int is 4
Min value for int is -2147483648
Max value for int is 2147483647

Type unsigned int, unsigned:
Size of type unsigned int is 4
Max value for unsigned int is 4294967295

Type long, signed long, long int, signed long int:
Size of type long is 4
Min value for long is -2147483648
Max value for long is 2147483647

Type unsigned long, unsigned long int:
Size of type unsigned long is 4
Max value for unsigned long is 4294967295

Type float:
Size of type float is 4
Length of binary m is 24

```

```

Max length of decimal m is 6
Min value of binary exp is -125
Min value of decimal exp is -37
Max value of binary exp is 128
Max value of decimal exp is 38
Min value for float is 1.175494E-038
Max value for float is 3.402823E+038

Type double:
Size of type double is 8
Length of binary m is 53
Max length of decimal m is 15
Min value of binary exp is -1021
Min value of decimal exp is -307
Max value of binary exp is 1024
Max value of decimal exp is 308
Min value for double is 2.225074E-308
Max value for double is 1.797693E+308

Type long double:
Size of type long double is 12
Length of binary m is 64
Max length of decimal m is 18
Min value of binary exp is -16381
Min value of decimal exp is -4931
Max value of binary exp is 16384
Max value of decimal exp is 4932

```

Типове	Java	C
Масиви	int [] a = new int [10];	int a[10];
Проверка за коректност на индекс	float [][] b = new float [10][20]; По време на изпълнение	float b[10][20]; Не се прави
Тип указател	Няма явен	int *p;
Тип структура	class Point { float x, y; }	struct Point { float x, y; }
Низове	String s1 = "Hello"; String s2 = new String("hello");	char *s1 = "Hello"; char s2[6]; strcpy(s2, "hello");

Програма ArraysAndStrings.c

```

/*
=====
Name      : ArraysAndStrings.c
Version   : 03.2011
Description : C arrays and strings
=====
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void) {
    printf("      Array of integers: int array[5]\n");
    int array[5];
    printf("Sizeof(int): %d\n", sizeof(int));
    printf("Sizeof(array): %d\n", sizeof(array));
    printf("Elements:\n");
    printf("array[0] = %d - undefined\n", array[0]);
    printf("array[1] = %d - undefined\n", array[1]);
    printf("array[2] = %d - undefined\n", array[2]);
    printf("array[3] = %d - undefined\n", array[3]);
}

```

```

printf("array[4] = %d - undefined\n", array[4]);
printf("Index 7 is out of bounds, but array[7] = %d\n", array[7]);

printf("      Array of integers: int array2[3] = {-101}\n");
int array2[3] = { -101 };
printf("Sizeof(int): %d\n", sizeof(int));
printf("Sizeof(array2): %d\n", sizeof(array2));
printf("Elements:\n");
printf("array2[0] = %d\n", array2[0]);
printf("array2[1] = %d\n", array2[1]);
printf("array2[2] = %d\n", array2[2]);

printf("      Array of integers: int array3[2][2] = {{1,2},{3,4}}\n");
int array3[2][2] = { { 1, 2 }, { 3, 4 } };
printf("Sizeof(int): %d\n", sizeof(int));
printf("Sizeof(array3): %d\n", sizeof(array3));
printf("Elements:\n");
printf("array3[0][0] = %d\n", array3[0][0]);
printf("array3[0][1] = %d\n", array3[0][1]);
printf("array3[1][0] = %d\n", array3[1][0]);
printf("array3[1][1] = %d\n", array3[1][1]);

printf("      C string: char s[10] = \"str\"\n");
char s[15] = "str";
printf("String: %s\n", s);
printf("Length(s): %d\n", strlen(s));
printf("Sizeof(s): %d\n", sizeof(s));
printf("Length(\"str\"): %d\n", strlen("str"));
printf("Sizeof(\"str\"): %d\n", sizeof("str"));
printf("Charactes:\n");
printf("s[0] = %c\n", s[0]);
printf("s[1] = %c\n", s[1]);
printf("s[2] = %c\n", s[2]);
printf("s[3] = '\\%o'\n", s[4]);

printf("      C string: char s2[] = \"str2\"\n");
char s2[] = "str2";
printf("String: %s\n", s2);
printf("Length(s2): %d\n", strlen(s2));
printf("Sizeof(s2): %d\n", sizeof(s2));
printf("Length(\"str2\"): %d\n", strlen("str2"));
printf("Sizeof(\"str2\"): %d\n", sizeof("str2"));
printf("Charactes:\n");
printf("str[0] = %c\n", s2[0]);
printf("str[1] = %c\n", s2[1]);
printf("str[2] = %c\n", s2[2]);
printf("str[3] = %c\n", s2[3]);
printf("str[4] = '\\%o'\n", s2[4]);

return EXIT_SUCCESS;
}

```

Изпълнение на програмата:

```

Array of integers: int array[5]
Sizeof(int): 4
Sizeof(array): 20
Elements:
array[0] = 2009288233 - undefined
array[1] = 2009288258 - undefined
array[2] = 4201840 - undefined
array[3] = 2293592 - undefined
array[4] = 4201942 - undefined
Index 7 is out of bounds, but array[7] = 3
      Array of integers: int array2[3] = {-101}
Sizeof(int): 4

```

```

Sizeof(array2): 12
Elements:
array2[0] = -101
array2[1] = 0
array2[2] = 0
    Array of integers: int array3[2][2] = {{1,2},{3,4}}
Sizeof(int): 4
Sizeof(array3): 16
Elements:
array3[0][0] = 1
array3[0][1] = 2
array3[1][0] = 3
array3[1][1] = 4
    C string: char s[10] = "str"
String: str
Length(s): 3
Sizeof(s): 15
Length("str"): 3
Sizeof("str"): 4
Charactes:
s[0] = s
s[1] = t
s[2] = r
s[3] = '\0'
    C string: char s2[] = "str2"
String: str2
Length(s2): 4
Sizeof(s2): 5
Length("str2"): 4
Sizeof("str2"): 5
Charactes:
str[0] = s
str[1] = t
str[2] = r
str[3] = 2
str[4] = '\0'

```

Програма Variables.c

```

/*
=====
Name      : Variables.c
Version   : 03.2011
Description : C variables and functions
=====
*/
#include <stdio.h>
#include <stdlib.h>

void func() {
    static int counter = 0;
    counter++;
    printf("Function is called %d time(s)\n", counter);
}

int main(void) {
    int i;
    for (i = 0; i < 5; i++) {
        printf("    i = %d\n", i);
        func();
    }
    return EXIT_SUCCESS;
}

```

Изпълнение на програмата:

```
i = 0
Function is called 1 time(s)
    i = 1
Function is called 2 time(s)
    i = 2
Function is called 3 time(s)
    i = 3
Function is called 4 time(s)
    i = 4
Function is called 5 time(s)
```

Програма IONumbers.c

```
/*
=====
Name      : IONumbers.c
Version   : 03.2011
Description : Reading and printing numbers in C
=====
*/
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    int n;
    printf("Enter an integer: ");

    scanf("%d", &n);
    printf("Integer %d is entered\n", n);

    float x;
    printf("Enter a float: ");
    scanf("%f", &x);
    printf("Float %f is entered\n", x);

    return EXIT_SUCCESS;
}
```

Изпълнение на програмата **IONumbers.exe**, копирана на E: в директория **ExecutableFiles**, от Windows:

След изпълнение на Start => Run => cmd:

```
C:\Documents and Settings\Butterfly>E:
E:>cd ExecutableFiles

E:\ExecutableFiles> IONumbers
Enter an integer: 123
Integer 123 is entered
Enter a float: 123.456
Float 123.456001 is entered

E:\ExecutableFiles>exit
```

Изпълнение на програмата в Eclipse:

```
123
123.456
```

```
Enter an integer: Integer 123 is entered
Enter a float: Float 123.456001 is entered
```

Програма IONumbersEclipse.c

```
/*
=====
Name      : IONumbersEclipse.c
Version   : 03.2011
Description : Reading and printing numbers in C (Eclipse)
=====
*/
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    int n;
    printf("    Enter an integer: ");
    fflush(stdout);
    scanf("%d", &n);
    printf("Integer %d is entered\n", n);
    fflush(stdout);

    float x;
    printf("    Enter a float: ");
    fflush(stdout);
    scanf("%f", &x);
    printf("Float %f is entered\n", x);

    return EXIT_SUCCESS;
}
```

Изпълнение на програмата в Eclipse:

```
Enter an integer: 123
Integer 123 is entered
Enter a float: 123.456
Float 123.456001 is entered
```

Операции и изрази. Аритметични операции. Операции за отношение и логически операции. Операция за присвояване. Изрази. Преобразуване на типовете. Операции за условен израз, за последователно изпълнение, за преобразуване на типовете и за размер на обект

Операции	Java	C
Конкатенация на низове	s1 + s2 s1 += s2	#include <string.h> strcat(s1, s2);
Логически операции	&&, , !	&&, , !
Операции за отношение	=, !=, >, <, >=, <=	=, !=, >, <, >=, <=
Аритметични операции	+, -, *, /, %, unary -	+, -, *, /, %, unary -
Побитови операции	>>, <<, >>>, &, , ^	>>, <<, &, , ^
Операции за присвояване	=, *=, /=, +=, -=, <<=, >>=, >>>=, =, ^=, =, %=	=, *=, /=, +=, -=, <<=, >>=, =, ^=, =, %=

Операции в C

Операция	Пример	Резултат
()	f()	Извикване на функцията f
[]	a[10]	Елементът с индекс 10 на масива a
->	s->a	Елементът a на структура или обединение с адрес s
.	s.a	Елементът a на структурата или обединението s
+ (унарен)	+a	Стойността на a
- (унарен)	-a	Стойността на a със сменен знак
* (унарен)	*a	Обектът с адрес a
& (унарен)	&a	Адресът на a
~	~a	Побитовото инвертиране на a
++ (префикс)	++a	Стойността на a след увеличаването с 1
++ (постфикс)	a++	Стойността на a преди увеличаването с 1
-- (префикс)	--a	Стойността на a след намаляването с 1
-- (постфикс)	a--	Стойността на a преди намаляването с 1
sizeof	sizeof (t)	Размерът в байтове на обект от тип t
sizeof	sizeof e	Размерът в байтове на обект от типа на израза e
+	a + b	Сумата на a и b
-	a - b	Разликата на a и b
*	a * b	Произведението на a и b
/	a / b	Частното от делението на a с b
%	a % b	Остатъкът от делението на a с b
>>	a >> b	a, изместено b бита надясно
<<	a << b	a, изместено b бита наляво
<	a < b	1 ако a < b; 0 в противен случай
>	a > b	1 ако a > b; 0 в противен случай
<=	a <= b	1 ако a <= b; 0 в противен случай
>=	a >= b	1 ако a >= b; 0 в противен случай
==	a == b	1 ако a е равно на b; 0 в противен случай
!=	a != b	1 ако a не е равно на b; 0 в противен случай
&	a & b	Побитовата конюнкция на a и b
	a b	Побитовата дизюнкция на a и b
^	a ^ b	Побитовото изключващо или (сума по модул две) на a и b
&&	a && b	Логическата конюнкция на a и b (результат 0 или 1)
	a b	Логическата дизюнкция на a и b (результат 0 или 1)
!	!a	Логическото отрицание на a (результат 0 или 1)
?:	a ? e1 : e2	Изразът e1, ако a не е 0 Изразът e2, ако a е 0
=	a = b	a, след като b е присвоено на a
+=	a += b	Сумата на a и b, присвоена на a

-=	a -= b	Разликата на a и b, присвоена на a
*=	a *= b	Произведенето на a и b, присвоено на a
/=	a /= b	Резултатът от делението на a с b, присвоен на a
%=	a %= b	Остатъкът от делението на a с b, присвоен на a
>>=	a >>= b	a, изместено b бита надясно, присвоено на a
<<=	a <<= b	a, изместено b бита наляво, присвоено на a
&=	a &= b	Побитовата конюнкция на a и b, присвоена на a
=	a = b	Побитовата дизюнкция на a и b, присвоена на a
^=	a ^= b	Побитовото изключващо или на a и b, присвоено на a
,	e1,e2	e2 (първо се пресмята e1)

Следващата таблица съдържа всички операции в C, наредени в намаляващ ред на приоритетите им и асоциативното правило за всяка група операции с един и същ приоритет.

Постфиксни операции	() [] -> . ++ --	Отляво надясно
Унарни (префиксни) операции	++ -- + - ~ ! (тип) * & sizeof	Отдясно наляво
Мултипликативни операции	* / %	Отляво надясно
Адитивни операции	+ -	Отляво надясно
Операции за изместване	<< >>	Отляво надясно
Отношения	< > <= >=	Отляво надясно
Отношения	== !=	Отляво надясно
Побитова конюнкция	&	Отляво надясно
Побитова сума по модул две	^	Отляво надясно
Побитова дизюнкция		Отляво надясно
Конюнкция	&&	Отляво надясно
Дизюнкция		Отляво надясно
Условна операция	? :	Отдясно наляво
Операции за присвояване	= += -= *= /= %= &= ^= = <<= >>=	Отдясно наляво
Запетая	,	Отляво надясно

Управляващи структури. Съставен и условен оператор. Оператори за цикъл. Многовариантен избор. Оператори за прекъсване на цикъл и за завършване на итерация на цикъл. Влагане на оператори

Оператори	Java	C
if	<pre>if (number >= 1000 && number <= 9999) { result = true; } else { result = false; }</pre>	<pre>if (number >= 1000 && number <= 9999) { result = 1; } else { result = 0; }</pre>
switch	<pre>switch (n) { case 1: ... break; case 2: ... break; default: ... }</pre>	<pre>switch (n) { case 1: ... break; case 2: ... break; default: ... }</pre>
goto	Няма	goto End;
for	<pre>for (int i = 1; i < 11; i++) { sum += i; }</pre>	<pre>int i; for (i = 1; i < 11; i++) { sum += i; }</pre>
while	<pre>while (a % 2 == 0) { a /= 2; }</pre>	<pre>while (a % 2 == 0) { a /= 2; }</pre>
do-while	<pre>do { a /= 2; }while (a % 2 == 0);</pre>	<pre>do { a /= 2; }while (a % 2 == 0);</pre>
continue	continue;	continue;
continue c	continue c1;	Няма

етикет		
break	<code>break;</code>	<code>break;</code>
break с етикет	<code>break C1;</code>	Няма
return	<code>return x;</code> <code>return;</code>	<code>return x;</code> <code>return;</code>
Съставен оператор (блок)	{ A = B; B = P; P = A % B; }	{ A = B; B = P; P = A % B; }
Изключения	<code>throw, try-catch-finally</code>	Няма
Извикване на метод	<code>f(x);</code> <code>System.out.println(C);</code> <code>MyClass.print(C);</code>	<code>f(x);</code>

Програма IOArrays.c

```
/*
=====
Name      : IOArrays.c
Version   : 03.2011
Description : Reading and printing arrays in C
=====
*/
#include <stdio.h>
#include <stdlib.h>

const int max_n = 10;
const int max_m = 10;

int main(void) {
    printf("      Reading and printing vector of numbers...\n");
    fflush(stdout);

    int vector[max_n];
    int n;
    printf("Enter number of elements 0 <= n <= %d: ", max_n);
    fflush(stdout);

    if (scanf("%d", &n) == 0 || n <= 0 || n > max_n) {
        printf("Incorrect data. Try again!\n");
        return EXIT_FAILURE;
    }

    //Input
    printf("Enter elements\n");
    fflush(stdout);
    int i;
    for (i = 0; i < n; i++) {
        scanf("%d", &vector[i]);
    }

    //Output
    printf("Elements entered\n");
    fflush(stdout);
    for (i = 0; i < n; i++) {
        printf("%10d", vector[i]);
        fflush(stdout);
    }

    printf("\n      Reading and printing table of numbers...\n");
    fflush(stdout);
}
```

```

int table[max_n][max_m];
printf("Enter number of rows 0 <= n <= %d: ", max_n);
fflush(stdout);

if (scanf("%d", &n) == 0 || n <= 0 || n > max_n) {
    printf("Incorrect data. Try again!\n");
    return EXIT_FAILURE;
}

int m;
printf("Enter number of columns 0 <= m <= %d: ", max_m);
fflush(stdout);

if (scanf("%d", &m) == 0 || m <= 0 || m > max_m) {
    printf("Incorrect data. Try again!\n");
    return EXIT_FAILURE;
}

//Input
printf("Enter elements\n");
fflush(stdout);
int j;
for (i = 0; i < n; i++) {
    for (j = 0; j < m; j++)
        scanf("%d", &table[i][j]);
}

//Output
printf("Elements entered\n");
fflush(stdout);
for (i = 0; i < n; i++) {
    for (j = 0; j < m; j++) {
        printf("%10d", table[i][j]);
        fflush(stdout);
    }
    printf("\n");
    fflush(stdout);
}

printf("Done!\n");

return EXIT_SUCCESS;
}

```

Изпълнение на програмата:

```

Reading and printing vector of numbers...
Enter number of elements 0 <= n <= 10: 5
Enter elements
1 2 3 4 5
Elements entered
      1          2          3          4          5
Reading and printing table of numbers...
Enter number of rows 0 <= n <= 10: 2
Enter number of columns 0 <= m <= 10: 3
Enter elements
6 7 8
9 10 11
Elements entered
      6          7          8
      9         10         11
Done!

```

Програма IOCharacters.c

```
/*
```

```
=====
Name      : IOCharacters.c
Version   : 03.2011
Description : Reading and printing characters in C
=====
*/
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    int count = 0, C;

    printf("Reading and printing characters...\n");
    fflush(stdout);

    while ((C = getchar()) != EOF) {
        count++;
        putchar(C);
        printf("      decimal %d\n", C);
    }

    printf("Reads and prints %d characters\n", count);

    return EXIT_SUCCESS;
}
```

Изпълнение на програмата:

```
Reading and printing characters...
ab
c
def
<ctrl-Z>
a      decimal 97
b      decimal 98

c      decimal 10
c      decimal 99

d      decimal 10
d      decimal 100
e      decimal 101
f      decimal 102

decimal 10
Reads and prints 9 characters
```

```
Reading and printing characters...
Текст на кирилица
т      decimal 210
е      decimal 229
к      decimal 234
с      decimal 241
т      decimal 242
      decimal 32
н      decimal 237
а      decimal 224
      decimal 32
к      decimal 234
и      decimal 232
р      decimal 240
и      decimal 232
л      decimal 235
и      decimal 232
```

```
ц      decimal 246
а      decimal 224

decimal 10
Reads and prints 18 characters
```

Програма IOStrings.c

```
/*
=====
Name       : IOStrings.c
Version    : 03.2011
Description : Reading and printing strings in C
=====*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void) {
    int count = 0;
    char str[100];

    printf("Reading and printing strings...\n");
    fflush(stdout);

    while (strlen(gets(str)) != 0) {
        count++;
        puts(str);
    }

    printf("Reads and prints %d lines\n", count);

    return EXIT_SUCCESS;
}
```

Изпълнение на програмата:

```
Reading and printing strings...
Line 1
Line 2
Line 3
<Enter>
Line 1
Line 2
Line 3
Reads and prints 3 lines
```

Програма NumOf1Bits.c

```
/*
=====
Name       : NumOf1Bits.c
Version    : 03.2011
Description : Demonstrates bitwise operators
=====
*/
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    unsigned int n;
```

```

unsigned int count =0;

printf("Enter an integer: ");
fflush(stdout);

if (scanf("%u", &n) == 0) {
    printf("Error: not an integer\n");
    return EXIT_FAILURE;
}

printf("Unsigned number is %u\n", n);
fflush(stdout);

for (; n > 0; n >= 1)
    count += (n & 1);

printf("Number of 1 bits is %u\n", count);

return EXIT_SUCCESS;
}

```

Изпълнение на програмата:

```

Enter an integer: -1
Unsigned number is 4294967295
Number of 1 bits is 32

```

```

Enter an integer: 127
Unsigned number is 127
Number of 1 bits is 7

```

```

Enter an integer: a12
Error: not an integer

```

Програма ShiftOps.c

```

/*
=====
Name      : ShiftOps.c
Version   : 03.2011
Description : Demonstrates shift operators
=====
*/
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    int n;
    int pow;

    printf("Enter an integer: ");
    fflush(stdout);

    if (scanf("%d", &n) == 0) {
        printf("Error: not an integer\n");
        return EXIT_FAILURE;
    }

    printf("Enter a positive power: ");
    fflush(stdout);

    if (scanf("%d", &pow) == 0) {
        printf("Error: not an integer\n");
        return EXIT_FAILURE;
    }
}

```

```

}

printf("%d * 2 ^ %d = %d\n", n, pow, n << pow);
printf("%d / 2 ^ %d = %d\n", n, pow, n >> pow);

return EXIT_SUCCESS;
}

```

Изпълнение на програмата:

Enter an integer: 5

Enter a power: 3

5 * 2 ^ 3 = 40

5 / 2 ^ 3 = 0

Enter an integer: -16

Enter a power: 2

-16 * 2 ^ 2 = -64

-16 / 2 ^ 2 = -4

Enter an integer: -1

Enter a positive power: 1

-1 * 2 ^ 1 = -2

-1 / 2 ^ 1 = -1

Enter an integer: 12

Enter a positive power: 2

12 * 2 ^ 2 = 48

12 / 2 ^ 2 = 3

Програма IntToBinaryString.c

```

/*
=====
Name      : IntToBinaryString.c
Version   : 03.2011
Description : Integer to Binary String conversion
=====
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void) {
    int n;
    if (scanf("%d", &n) == 0) {
        printf("Error: not an integer\n");
        return EXIT_FAILURE;
    }

    char str[40] = "";
    unsigned int mask;
    for (mask = 0x80000000; mask > 0; mask >>= 1)
        if (n & mask)
            strcat(str, "1");
        else
            strcat(str, "0");

    printf("%s\n", str);

    return EXIT_SUCCESS;
}

```

Изпълнение на програмата:

-1

Программа BinaryStringToInt.c

```
/*
=====
Name      : BinaryStringToInt.c
Version   : 03.2011
Description : Binary String to Integer conversion
=====

*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void) {
    char str[40];
    scanf("%s", str);
    int length = strlen(str);
    if (length != 32) {
        printf("Error: invalid string length %d\n", length);
        return EXIT_FAILURE;
    }

    int n = 0;
    unsigned int mask = 0x80000000;
    int i;
    for (i = 0; i < length; i++) {
        char c = str[i];
        if (c == '1')
            n |= mask;
        else if (c != '0') {
            printf("Error: invalid character %c\n", c);
            return EXIT_FAILURE;
        }
        mask >>= 1;
    }

    printf("%d\n", n);

    return EXIT_SUCCESS;
}
```

Изпълнение на програмата:

223

1111111111000100

Error: invalid string length 17

000000000a000000000000000011011111

Error: invalid character a

Задачи за упражнение

Задача: Да се състави програма, която запълва по редове квадратна таблица A($n \times n$), $n < 6$ с главните латински букви, започвайки от буквата A и след това транспонира елементите на таблицата A без използване на допълнителен масив.

Задача: Да се състави програма, която чете текст и извършва следните обработки:

- Определя броя на редовете в текста
- Определя броя на графичните символи в текста
- Определя колко пъти се среща всяка от малките латински букви в текста

Задача: Да се състави програма, която проверява дали последователност от символи, завършваща със символа #, е съставена от малки и главни латински букви, цифри и символа за подчертаване _ и започва с буква или _.

Задача: Да се състави програма, която:

1. Кодира даден текст, като заменя всяка малка латинска буква със съответната ѝ по място малка латинска буква от дадено множество
2. Декодира кодирания текст
3. Сравнява декодирания текст с първоначалния, за да установи дали те съвпадат или не

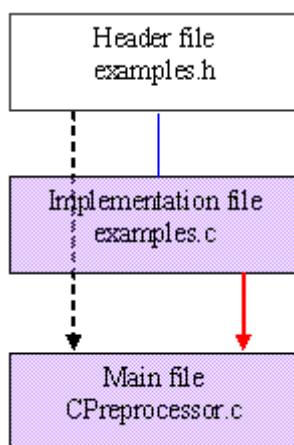
Задача: Да се състави програма, която чете последователност от символи, завършваща със символа '=' и представляваща аритметичен израз баз скоби с операции +, -, *, / и операнди – цели десетични положителни числа. Пресмята стойността на израза, като изпълнява операциите отляво надясно, преобразува я в символен вид и я извежда.

Препроцесор. Принципи на работа на препроцесора. Директиви на препроцесора. Макроопределения. Макроопределения с аргументи. Включване на файлове. Условна компилация

Директиви на препроцесора на С

Директива	Предназначение
#define	За дефиниране на макроопределение
#undef	За отмяна на макроопределение
#include	За включване на текст от файл
#if	За включване на текст в зависимост от стойността на константен израз
#ifdef	За включване на текст в зависимост от това дали е дефинирано макроопределение. Ако е дефинирано, текстът се включва. В противен случай не се включва
#ifndef	За включване на текст в зависимост от това дали е дефинирано макроопределение. Ако не е дефинирано, текстът се включва. В противен случай не се включва
#elif	За включване на текст в зависимост от стойността на константен израз, ако това не е станало при предходните директиви #if, #ifdef, #ifndef или #elif
#else	За включване на текст, ако това не е станало при предходните директиви #if, #ifdef, #ifndef или #elif
#endif	За означаване на края на условна конструкция
#error	За прекратяване на транслацията и извеждане на съобщение
#line	За присвояване на номер на програмен ред

ПРОЕКТ CPreprocessor:



Интерфейсът **examples.h** представя функции за илюстрация на директиви на препроцесора:

```

/*
 * examples.h
 *
 * Created on: 06.2012
 *      Examples Interface
 */

#ifndef EXAMPLES_H_
#define EXAMPLES_H_

void example1();
void example2();


```

```
void example3();
void example4();

#endif /* EXAMPLES_H_ */
```

Реализацията на функциите е във файла **examples.c**:

```
/*
 * examples.c
 *
 * Created on: 06.2012
 *      Examples Implementation
 */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

//Macros
#define PI 3.14
#define CircleArea(r) (PI*(r)*(r))
#define InvalidCircleArea(r) PI*r*r

void example1() {
    printf("Starting...\n");

    float result;
    srand(time(0));
    float x = rand() % 5 + 3;
    printf("Number %f is created\n", x);
    result = CircleArea(x);
    printf("CircleArea(%f) = %f\n", x, result);
    result = InvalidCircleArea(x);
    printf("InvalidCircleArea(%f) = %f\n", x, result);

    printf("Number %f is created\n", x + 2.0);
    result = CircleArea(x + 2.0);
    printf("CircleArea(%f+2.0) = %f\n", x, result);
    result = InvalidCircleArea(x + 2.0);
    printf("InvalidCircleArea(%f+2.0) = %f\n", x, result);

    printf("Done!\n");
}

#define gcd(x,y,r) { int A = (x); \
                  int B = (y); \
                  int P = A % B; \
                  while (P != 0) { \
                      A = B; \
                      B = P; \
                      P = A % B; \
                  } \
                  r = B; \
              }

void example2() {
    printf("Starting...\n");

    int i, a, b, p;
    srand(time(0));
    for (i = 1; i < 4; i++) {
        a = rand() % 10 + 10;
        printf("Number %d is created\n", a);
        b = rand() % 100 + 10;
        printf("Number %d is created\n", b);
        gcd(a,b,p);
    }
}
```

```

printf("GCD(%d,%d) = %d\n", a, b, p);

printf("Number %d is created\n", 2 * a);
printf("Number %d is created\n", 3 * b);
gcd(2*a,3*b,p);
printf("GCD(2*%d,3*%d) = %d\n", a, b, p);
}

printf("Done!\n");
}

//Conditional compiling
#ifndef mode
#define mode 4
#endif

void example3() {
    printf("Starting...\n");

    //Conditional compiling
#if mode == 1
    printf("mode: Action 1\n");
#elif mode == 2
    printf("mode: Action 2\n");
#elif mode == 3
    printf("mode: Action 3\n");
#else
#error("mode must be 1, 2 or 3");
#endif

    printf("Done!\n");
}

#ifndef mode2
#define mode2
#endif

void example4() {
    printf("Starting...\n");

    //Conditional compiling
#ifdef mode2
    printf("mode2: Action\n");
#else
#error("mode2 must be defined\n");
#endif

    printf("Done!\n");
}

```

Програмата **CPreprocessor.c** извиква всяка от функциите:

```

/*
=====
Name      : CPreprocessor.c
Version   : 06.2012
Description : C Preprocessor Examples
=====

#include <stdio.h>
#include <stdlib.h>
#include "examples.h"

int main(void) {
    printf("Example1\n");

```

```

example1();
printf("      Example2\n");
example2();
printf("      Example3\n");
example3();
printf("      Example4\n");
example4();
return EXIT_SUCCESS;
}

```

Изпълнение на програмата:

```

**** Build of configuration Debug for project CPreprocessor ****

**** Internal Builder is used for build
***** gcc -O0 -g3 -Wall -c -fmessage-length=0 -o src\examples.o ..\src\examples.c
..\src\examples.c:91:2: error: #error ("mode must be 1, 2 or 3");
Build error occurred, build is stopped
Time consumed: 531 ms.

```

Модификация:

```

/*
 * examples.c
 *
 * Created on: 06.2012
 *      Examples Implementation
 */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

//Macros
#define PI 3.14
#define CircleArea(r) (PI*(r)*(r))
#define InvalidCircleArea(r) PI*r*r

void example1() {
    printf("Starting...\n");

    float result;
    srand(time(0));
    float x = rand() % 5 + 3;
    printf("Number %f is created\n", x);
    result = CircleArea(x);
    printf("CircleArea(%f) = %f\n", x, result);
    result = InvalidCircleArea(x);
    printf("InvalidCircleArea(%f) = %f\n", x, result);

    printf("Number %f is created\n", x + 2.0);
    result = CircleArea(x + 2.0);
    printf("CircleArea(%f+2.0) = %f\n", x, result);
    result = InvalidCircleArea(x + 2.0);
    printf("InvalidCircleArea(%f+2.0) = %f\n", x, result);

    printf("Done!\n");
}

#define gcd(x,y,r) { int A = (x); \
                  int B = (y); \
                  int P = A % B; \
                  while (P != 0) { \
                      A = B; \
                      B = P; \

```

```

        P = A % B; \
    } \
    r = B; \
}

void example2() {
    printf("Starting...\n");

    int i, a, b, p;
    rand(time(0));
    for (i = 1; i < 4; i++) {
        a = rand() % 10 + 10;
        printf("Number %d is created\n", a);
        b = rand() % 100 + 10;
        printf("Number %d is created\n", b);
        gcd(a,b,p);
        printf("GCD(%d,%d) = %d\n", a, b, p);

        printf("Number %d is created\n", 2 * a);
        printf("Number %d is created\n", 3 * b);
        gcd(2*a,3*b,p);
        printf("GCD(2*%d,3*%d) = %d\n", a, b, p);
    }

    printf("Done!\n");
}

//Conditional compiling
#ifndef mode
#define mode 2
#endif

void example3() {
    printf("Starting...\n");

    //Conditional compiling
#if mode == 1
    printf("mode: Action 1\n");
#elif mode == 2
    printf("mode: Action 2\n");
#elif mode == 3
    printf("mode: Action 3\n");
#else
#error("mode must be 1, 2 or 3");
#endif

    printf("Done!\n");
}

#ifndef mode2
#define mode2
#endif

void example4() {
    printf("Starting...\n");

    //Conditional compiling
#ifdef mode2
    printf("mode2: Action\n");
#else
#error("mode2 must be defined\n");
#endif

    printf("Done!\n");
}

```

Изпълнение на програмата:

```
Example1
Starting...
Number 4.000000 is created
CircleArea(4.000000) = 50.240002
InvalidCircleArea(4.000000) = 50.240002
Number 6.000000 is created
CircleArea(4.000000+2.0) = 113.040001
InvalidCircleArea(4.000000+2.0) = 22.559999
Done!
Example2
Starting...
Number 11 is created
Number 37 is created
GCD(11, 37) = 1
Number 22 is created
Number 111 is created
GCD(2*11, 3*37) = 1
Number 18 is created
Number 64 is created
GCD(18, 64) = 2
Number 36 is created
Number 192 is created
GCD(2*18, 3*64) = 12
Number 18 is created
Number 15 is created
GCD(18, 15) = 3
Number 36 is created
Number 45 is created
GCD(2*18, 3*15) = 9
Done!
Example3
Starting...
mode: Action 2
Done!
Example4
Starting...
mode2: Action
Done!
```

Функции. Дефиниция на функция. Параметри на функция. Връщани резултати. Обръщение към функция. Прототип на функция. Модули. Област на действие на променливите. Класове памет. Инициализиране на променливите

Програма Functions1_1: Програмата илюстрира предаването на параметри от примитивен тип. Изпълнете програмата и обясните защо се получава такъв резултат.

```
/*
=====
Name      : Functions1_1.c
Version   : 03.2011
Description : Passing Parameters to a Function
=====

*/
#include <stdio.h>
#include <stdlib.h>

//Example for Pass by Value
void swap1(int arg1, int arg2) {
    int work = arg1;
    arg1 = arg2;
    arg2 = work;
}

//Example for Pass by Reference
void swap2(int *arg1, int *arg2) {
    int work = *arg1;
    *arg1 = *arg2;
    *arg2 = work;
}

int main(void) {
    int a = 10, b = 20;
    printf("      a = 10, b = 20 => swap1(a,b)\n");
    swap1(a, b);
    if (a == 20 && b == 10)
        printf("swap1: Arguments are swaped!\n");
    else if (a == 10 && b == 20)
        printf("swap1: Arguments are not changed!\n");
    else
        printf("swap1: Too strange!\n");

    a = 10;
    b = 20;
    printf("      a = 10, b = 20 => swap2(a,b)\n");
    swap2(&a, &b);
    if (a == 20 && b == 10)
        printf("swap2: Arguments are swaped!\n");
    else if (a == 10 && b == 20)
        printf("swap2: Arguments are not changed!\n");
    else
        printf("swap2: Too strange!\n");

    return EXIT_SUCCESS;
}
```

Изпълнение на програмата:

```
a = 10, b = 20 => swap1(a,b)
swap1: Arguments are not changed!
a = 10, b = 20 => swap2(a,b)
```

```
swap2: Arguments are swaped!
```

Задача: Да се състави програма, която чете n числа и ги извежда в ред обратен на въвеждането.

Реализация: Програма **Functions1_2**, съгласно спецификацията:

```
void error(char *str) {...}
//Извежда съобщение за грешка и прекратява изпълнението на програмата
int main(void) {...}
//Решава задачата, като използва локален масив numbers

/*
=====
Name      : Functions1_2.c
Version   : 03.2011
Description: Local array in a function
=====
*/
#include <stdio.h>
#include <stdlib.h>

#define MAX_SIZE 10

void error(char *str);

int main(void) {
    double numbers[MAX_SIZE];
    int n = MAX_SIZE;
    int i;

    //Input:
    printf("      Enter n>0: ");
    fflush(stdout);
    if (scanf("%d", &n) == 0 || n < 1 || n > MAX_SIZE)
        error("invalid n");

    printf("      Input:\n");
    for (i = 1; i <= n; i++) {
        printf("number[%d]: ", i);
        fflush(stdout);
        scanf("%lg", &numbers[i - 1]);
    }

    //Output:
    printf("      Output:\n");
    for (i = n - 1; i >= 0; i--)
        printf("number[%d] = %lg\n", i + 1, numbers[i]);

    return EXIT_SUCCESS;
}

void error(char *str) {
    fprintf(stderr, "ERROR: %s\n", str);
    exit(EXIT_FAILURE);
}
```

Изпълнение на програмата:

Първо изпълнение:

```
Enter n>0: 3
Input:
number[1]: 2.345
```

Второ изпълнение:

```
Enter n>0: 0
ERROR: invalid n
```

Трето изпълнение:

```
Enter n>0: -1
ERROR: invalid n
```

```
number[2]: 1e-2
number[3]: 0
    Output:
number[3] = 0
number[2] = 0.01
number[1] = 2.345
```

Реализация: Програма **Functions1_3**, съгласно спецификацията:

```
//Декларация на глобален масив numbers
//Декларация на глобална променлива n за актуалния брой на елементите в масива numbers

void error(char *str) {...}
//Извежда съобщение за грешка и прекратява изпълнението на програмата
void read() {...}
//Чете n и n числа и ги съхранява в масива numbers
void print() {...}
//Извежда n числа от масива numbers в ред обратен на въвеждането им
int main(void) {...}
//Извиква двета метода

/*
=====
Name      : Functions1_3.c
Version   : 03.2011
Description : Global array and functions
=====
*/
#include <stdio.h>
#include <stdlib.h>

#define MAX_SIZE 10

double numbers[MAX_SIZE];
int n = MAX_SIZE;

void error(char *str);
void read();
void print();

int main(void) {
    read();
    print();

    return EXIT_SUCCESS;
}

void error(char *str) {
    fprintf(stderr, "ERROR: %s\n", str);
    exit(EXIT_FAILURE);
}

void read() {
    int i;
    //Input:
    printf("Enter n>0: ");
    fflush(stdout);
    if (scanf("%d", &n) == 0 || n < 1 || n > MAX_SIZE)
        error("invalid n");

    printf("Input:\n");
    for (i = 1; i <= n; i++) {
        printf("number[%d]: ", i);
        fflush(stdout);
```

```

        scanf("%lg", &numbers[i - 1]);
    }

void print() {
    int i;
    //Output:
    printf("      Output:\n");
    for (i = n - 1; i >= 0; i--)
        printf("number[%d] = %lg\n", i + 1, numbers[i]);
}

}

```

Реализация: Програма **Functions1_4**, съгласно спецификацията:

```

void error(char *str) { ... }
//Извежда съобщение за грешка и прекратява изпълнението на програмата
void read(double numbers[],int *n) { ... }
//Чете n и ги съхранява в масива numbers
void print(const double numbers[],const int n) { ... }
//Извежда n числа от масива numbers в ред обратен на въвеждането им. Не може да променя параметрите
public static void main(String[] args) { ... }
//Създава локален масив и локална променлива за актуалния брой елементи. Чете данните с метода read.
//Извежда числата с метода print

/*
=====
Name      : Functions1_4.c
Version   : 03.2011
Description : Parameter array to a function
=====
*/
#include <stdio.h>
#include <stdlib.h>

#define MAX_SIZE 10

void error(char *str);
void read(double a[], int *n);
void print(const double a[], int n);

int main(void) {
    double numbers[MAX_SIZE];
    int n = MAX_SIZE;

    read(numbers, &n);
    print(numbers, n);

    return EXIT_SUCCESS;
}

void error(char *str) {
    fprintf(stderr, "ERROR: %s\n", str);
    exit(EXIT_FAILURE);
}

void read(double a[], int *n) {
    int i;
    //Input:
    printf("      Enter n>0: ");
    fflush(stdout);
    if (scanf("%d", n) == 0 || *n < 1 || *n > MAX_SIZE)
        error("invalid n");

    printf("      Input:\n");
}

```

```

for (i = 1; i <= *n; i++) {
    printf("number[%d]: ", i);
    fflush(stdout);
    scanf("%lg", &a[i - 1]);
}
}

void print(const double a[], int n) {
    int i;
    //Output:
    printf("      Output:\n");
    for (i = n - 1; i >= 0; i--)
        printf("number[%d] = %lg\n", i + 1, a[i]);
}

```

Задача: Да се състави програма, която за дадено цяло число $n \in [2; 40]$ пресмята итеративно и рекурсивно n -ия член на редицата от числа на Фибоначи a_k , за всяко $k \in [1; n]$, като определя и времето за всяко от пресмятанията и извежда получените резултати на стандартния изход.

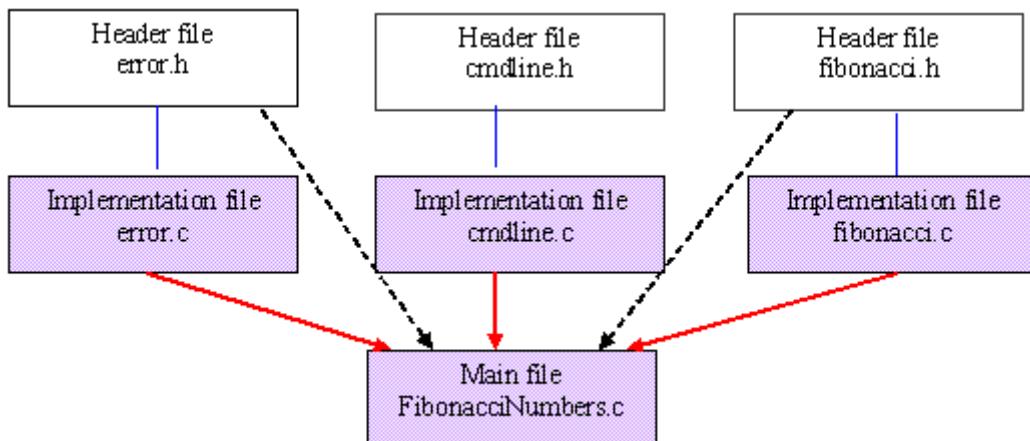
*Спецификация на програмата **FibonacciNumbers.c**:*

```

int cmdLine(int argc, char *argv[]) {...}
//Анализира параметрите на командния ред и връща стойността на n
void error(char *str) {...}
//Извежда съобщение за грешка и прекратява изпълнението на програмата
unsigned long iterFib(int n) {...}
//Пресмята итеративно  $n$ -ия член на редицата от числа на Фибоначи
unsigned long recFib(int n) {...}
//Пресмята рекурсивно  $n$ -ия член на редицата от числа на Фибоначи
int main(int argc, char * argv[]) {...}
//Решава задачата

```

*Проект **FibonacciNumbers**:*



Реализация:

```

/*
 * error.h
 *
 * Created on: 03.2011
 *
 */
#ifndef ERROR_H_
#define ERROR_H_

void error(char *str);


```

```
#endif /* ERROR_H_ */

/*
 * error.c
 *
 * Created on: 03.2011
 */
#include <stdio.h>
#include <stdlib.h>

void error(char *str){
    fprintf(stderr,"ERROR: %s\n",str);
    exit(EXIT_FAILURE);
}

/*
 * cmdline.h
 *
 * Created on: 03.2011
 */
#ifndef CMDLINE_H_
#define CMDLINE_H_

int cmdLine(int argc, char *argv[]);

#endif /* CMDLINE_H_ */

/*
 * cmdline.c
 *
 * Created on: 03.2011
 */
#include <stdlib.h>
#include "error.h"

int cmdLine(int argc, char *argv[]) {
    if (argc < 2)
        error("missing argument");
    else if (argc > 2)
        error("too many arguments");

    int n = atoi(argv[1]);
    if (n == 0)
        error("not an integer");

    if (n < 2)
        error("a number is less than 2");
    else if (n > 40)
        error("a number is greater than 40");

    return n;
}

/*
 * fibonacci.h
 *
 * Created on: 03.2011
 */

```

```

#ifndef FIBONACCI_H_
#define FIBONACCI_H_

unsigned long iterFib(int n);
unsigned long recFib(int n);

#endif /* FIBONACCI_H_ */

/*
 * fibonacci.c
 *
 * Created on: 03.2011
 *
 */

/*
 * fibonacci.c
 *
 * Created on: 03.2011
 *
 */

unsigned long iterFib(int n) {
    if (n == 0)
        return 0;
    else if (n == 1)
        return 1;
    else {
        unsigned long a0 = 0, a1 = 1, a = 1, i;
        for (i = 2; i <= n; i++) {
            a = a0 + a1;
            a0 = a1;
            a1 = a;
        }
        return a;
    }
}

unsigned long recFib(int n) {
    if (n == 0)
        return 0;
    else if (n == 1)
        return 1;
    else
        return (recFib(n - 2) + recFib(n - 1));
}

/*
=====
Name      : FibonacciNumbers.c
Version   : 03.2011
Description : Finds the Fibonacci(i) number, 1 <= i <= n,
              using recFib and iterFib functions.
              n must be between 2 and 40.
mode:
          0 - Reading n from stdin
          1 - n is command line argument
=====
*/
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "error.h"
#include "cmdline.h"
#include "fibonacci.h"

#ifndef mode
#define mode 1
#endif

int main(int argc, char * argv[]) {
    int n;

```

```

//Conditional compiling
#if mode == 1
    n = cmdLine(argc, argv);
#elif mode == 0
    if (scanf("%d", &n) == 0)
        error("not an integer");
    else if (n < 2)
        error("a number is less than 2");
    else if (n > 40)
        error("a number is greater than 40");
#else
    error("mode must be 1 or 0");
#endif

int i;
unsigned long a;
clock_t t1, t2, t3, t4;
for (i = 1; i <= n; ++i) {
    //IterTime
    t1 = clock();
    a = iterFib(i);
    t2 = clock();

    //RecTime
    t3 = clock();
    a = recFib(i);
    t4 = clock();

    printf("n = %3d, an = %15lu, IterTime: %7ld sec, RecTime: %7ld sec\n",
           i, a, (t2 - t1) / CLOCKS_PER_SEC, (t4 - t3) / CLOCKS_PER_SEC);
}

return EXIT_SUCCESS;
}

```

Изпълнение на програмата **FibonacciNumbers.exe**, копирана на E: в директория **ExecutableFiles**, от Windows:

След изпълнение на Start => Run => cmd:

```

C:\Documents and Settings\Butterfly>E:
E:>cd ExecutableFiles

E:\ExecutableFiles>FibonacciNumbers
ERROR: missing argument

E:\ExecutableFiles>FibonacciNumbers 49
ERROR: a number is greater than 40

E:\ExecutableFiles>FibonacciNumbers 5
n = 1, an =      1, IterTime: 0 sec, RecTime: 0 sec
n = 2, an =      1, IterTime: 0 sec, RecTime: 0 sec
n = 3, an =      2, IterTime: 0 sec, RecTime: 0 sec
n = 4, an =      3, IterTime: 0 sec, RecTime: 0 sec
n = 5, an =      5, IterTime: 0 sec, RecTime: 0 sec

E:\ExecutableFiles>FibonacciNumbers 40 > FibonacciNumbers.txt

E:\ExecutableFiles>exit

```

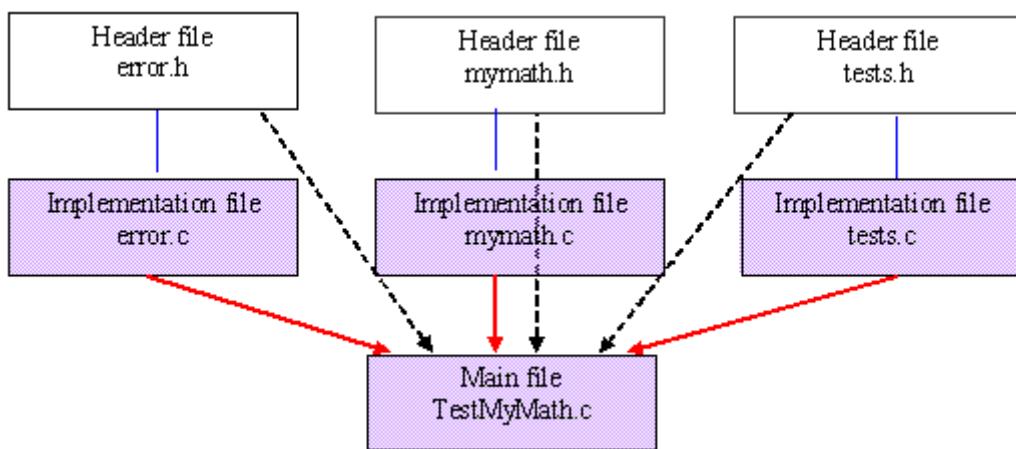
Задача: Да се реализират функциите:

```

double E = 1E-12;
//Променлива за точността на изчисленията
double e() {...}
//Пресмята числото е с точност Е
double pi() {...}
//Пресмята числото pi с точност Е
double sqrt(double arg) {...}
//Пресмята корен квадратен от arg с точност Е
double exp(double arg) {...}
//Пресмята earg с точност Е
double sin(double arg) {...}
//Пресмята sin(arg) с точност Е
double cos(double arg) {...}
//Пресмята cos(arg) с точност Е
double arctg(double arg) {...}
//Пресмята arctg(arg) с точност Е

```

Проект TestMyMath:



Задача: Да се реализират функциите:

```

int gcd(int a, int b) {...}
//Определя най-големия общ делител на целите числа a и b
int isTrue(int a, int b) {...}
//Проверява дали целите числа a и b са взаимно прости
int isEven(int n) {...}
//Проверява дали цялото число n е четно
int sum(int n) {...}
//Определя сумата от делителите на цялото число n
int isPerfect(int n) {...}
//Проверява дали естественото число n е съвършено
int areFriendly(int a, int b) {...}
//Проверява дали естествените числа a и b са приятелски
int isPrime(int n) {...}
//Проверява дали цялото число n е просто
int number_of_digits(int n) {...}
//Определя броя на цифрите на цялото число n
int sum_of_digits(int n) {...}
//Определя сумата от цифрите на цялото число n
int reverse_number(int n) {...}
//Определя обратното число на цялото число n
int isPolindrom(int n) {...}
//Проверява дали цялото число n е палиндром (съвпада със своето обратно число)
int number_of_digits(int n, int p) {...}
//Определя броя на цифрите на р-ичния запис на цялото число n
int sum_of_digits(int n, int p) {...}
//Определя сумата от цифрите на р-ичния запис на цялото число n

```

Задача: Да се състави програма, която за дадена колекция от числа определя различните числа в колекцията и колко пъти всяко се съдържа в нея и извежда резултата в сортиран вид на различните числа от колекцията.

Реализация:

```
/*
=====
Name      : DifferentNumbers.c
Version   : 03.2011
Description : Different numbers
=====
*/
#include <stdio.h>
#include <stdlib.h>

#define MAX_ROWS 10
#define MAX_COLUMNS 2

void error(char *str);
void read(double a[], int *n);
void print(double a[][MAX_COLUMNS], int n);
int linearSearch(double v, double a[][MAX_COLUMNS], int left, int right);
void differentElements(double a[], int n, double b[][MAX_COLUMNS], int *m);
int compareTo(double a[][MAX_COLUMNS], int i, int j);
void swap(double a[][MAX_COLUMNS], int i, int j);
void bubbleSort(double a[][MAX_COLUMNS], int n);

int main(void) {
    double A[MAX_ROWS];
    int n = MAX_ROWS;
    read(A, &n);

    printf("      Different elements\n");
    double B[MAX_ROWS][MAX_COLUMNS];
    int count = MAX_ROWS;
    differentElements(A, n, B, &count);
    print(B, count);

    printf("      Sorted different elements\n");
    bubbleSort(B, count);
    print(B, count);

    return EXIT_SUCCESS;
}

void error(char *str) {
    fprintf(stderr, "ERROR: %s\n", str);
    exit(EXIT_FAILURE);
}

void read(double a[], int *n) {
    int i;
    //Input:
    printf("      Enter n>0: ");
    fflush(stdout);
    if (scanf("%d", n) == 0 || *n < 1 || *n > MAX_ROWS)
        error("invalid n");

    printf("      Input:\n");
    for (i = 1; i <= *n; i++) {
        printf("number[%d]: ", i);
        fflush(stdout);
        scanf("%lg", &a[i - 1]);
    }
}
```

```

    }

}

void print(double a[][][MAX_COLUMNS], int n) {
    int i;

    //Output:
    printf("      Output:\n");
    for (i = 0; i < n; i++)
        printf("number[%d] = %15lg ==> %7lg\n", i + 1, a[i][0], a[i][1]);
}

int linearSearch(double v, double a[][][MAX_COLUMNS], int left, int right) {
    int i, result = -1;
    for (i = left; i <= right; i++)
        if (a[i][0] == v) {
            result = i;
            break;
        }
    return result;
}

void differentElements(double a[], int n, double b[][][MAX_COLUMNS], int *m) {
    int i;
    for (i = 0; i < n; i++)
        b[i][0] = b[i][1] = 0;
    int index, k = -1;
    for (i = 0; i < n; i++) {
        index = linearSearch(a[i], b, 0, k);
        if (index == -1) {
            b[++k][0] = a[i];
            index = k;
        }
        b[index][1] = b[index][1] + 1;
    }
    *m = (k + 1);
}

int compareTo(double a[][][MAX_COLUMNS], int i, int j) {
    return a[i][0] - a[j][0];
}

void swap(double a[][][MAX_COLUMNS], int i, int j) {
    double work = a[i][0];
    a[i][0] = a[j][0];
    a[j][0] = work;
    work = a[i][1];
    a[i][1] = a[j][1];
    a[j][1] = work;
}

void bubbleSort(double a[][][MAX_COLUMNS], int n) {
    int i, sorted;
    do {
        sorted = 1;
        for (i = 0; i < n - 1; i++)
            if (compareTo(a, i, i + 1) > 0) {
                swap(a, i, i + 1);
                sorted = 0;
            }
    } while (!sorted);
}

```

Изпълнение на програмата:

Enter n>0: 10

```

Input:
number[1]: -12
number[2]: 234
number[3]: 1
number[4]: 10
number[5]: 234
number[6]: 10
number[7]: 234
number[8]: 1
number[9]: 10
number[10]: 2
    Different elements
Output:
number[1] =           -12 ==>      1
number[2] =           234 ==>      3
number[3] =             1 ==>      2
number[4] =            10 ==>      3
number[5] =            2 ==>      1
    Sorted different elements
Output:
number[1] =           -12 ==>      1
number[2] =             1 ==>      2
number[3] =             2 ==>      1
number[4] =            10 ==>      3
number[5] =           234 ==>      3

```

Задача: Да се реализират функциите, като се използват крайни детерминирани автомати:

```

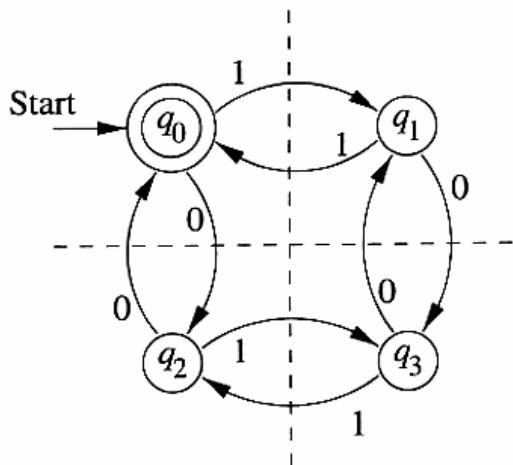
int checkBinaryString(char *str);
//Проверява дали низът str принадлежи на езика  $L=\{\omega \mid \omega \in \{0,1\}^*, N_0(\omega)=0 \pmod 2 \text{ и } N_1(\omega)=0 \pmod 2\}$ 
int isIdentifier(char *str);
//Проверява дали низът str е идентификатор на езика C
int isDecimalLiteral(char *str);
//Проверява дали низът str е десетичен литерал на езика C

```

Решение: КДА $A = \langle \{q_0, q_1, q_2, q_3\}, \{0, 1\}, q_0, \delta, \{q_0\} \rangle$ с функция на преходите δ :

	0	1
$* \rightarrow q_0$	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

разпознава езика L :



Указатели. Основни операции. Адресна аритметика. Масиви и указатели. Предаване на резултати чрез формални параметри. Формални параметри и масиви. Указатели към функции. Функции с променлив брой параметри

Програма Pointers.c

```
/*
=====
Name      : Pointers.c
Version   : 03.2011
Description : C Pointer
=====
*/
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    printf("Name\t\tSizeof\t\tAddress\t\tValue\n");

    unsigned int n = 0xa1b2c3d4;
    unsigned int *pn = &n;
    printf("\n\t\t%d\t\t%d\t\t%x\n", sizeof(n), pn, n);

    void *temp = pn;
    unsigned char *addr = temp;
    printf("addr\t\t%d\t\t%d\t\t%d\t\t%d\n", sizeof(addr), &addr, addr);

    int i;
    for (i = 0; i < 4; i++) {
        printf("addr[%d]\t\t%d\t\t%d\t\t%d\t\t%x\n", i, sizeof(addr[i]), &addr[i], addr[i]);
        printf("addr+%d\t\t%d\t\t%d\t\t%d\t\t%d\n", i, sizeof(addr+i), addr + i,
        *(addr + i));
    }

    return EXIT_SUCCESS;
}
```

Изпълнение на програмата:

Name	Sizeof	Address	Value
n	4	2293568	a1b2c3d4
addr	4	2293564	2293568
addr[0]	1	2293568	d4
addr+0	4	2293568	d4
addr[1]	1	2293569	c3
addr+1	4	2293569	c3
addr[2]	1	2293570	b2
addr+2	4	2293570	b2
addr[3]	1	2293571	a1
addr+3	4	2293571	a1

Програма PointersAndArrays.c

```
/*
=====
Name      : PointersAndArrays.c
Version   : 04.2011
Description : Pointers and Arrays
=====
*/

```

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void error(char *str);
void printArray(double *base, double *endPtr);

int main(void) {
    int n;
    printf("      Enter n>0: ");
    fflush(stdout);
    if (scanf("%d", &n) == 0 || n < 1)
        error("invalid n");

    int size = sizeof(double);
    double *data = calloc(n, size);
    /*
     * Allocates the requested memory and returns a pointer to it.
     * The requested size is n each size bytes long (total memory requested is
     * n*size).
     * The space is initialized to all zero bits. On success a pointer to the
     * requested space is returned. On failure a null pointer is returned.
     */
    if (data == NULL)
        error("impossible to allocate the requested memory");

    int i;
    for (i = 0; i < n; i++)
        //data[i] = 1000 * sin(i) / (2.4 + sqrt(i));
        *(data + i) = 1000 * sin(i) / (2.4 + sqrt(i));

    double *endOfArray = data + n;
    printArray(data, endOfArray);

    return EXIT_SUCCESS;
}

void error(char *str) {
    fprintf(stderr, "ERROR: %s\n", str);
    exit(EXIT_FAILURE);
}

```

Програма PointersAndPointerIdioms.c

```

/*
=====
Name      : PointersAndPointerIdioms.c
Version   : 04.2011
Description : Pointers and Arrays
=====

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void error(char *str);

int main(void) {
    int n;

```

```

printf("      Enter n>0: ");
fflush(stdout);
if (scanf("%d", &n) == 0 || n < 1)
    error("invalid n");

int size = sizeof(int);
int *data = calloc(n, size);
/*
 * Allocates the requested memory and returns a pointer to it.
 * The requested size is n each size bytes long (total memory requested is
 * n*size).
 * The space is initialized to all zero bits. On success a pointer to the
 * requested space is returned. On failure a null pointer is returned.
 */
if (data == NULL)
    error("impossible to allocate the requested memory");

int i;
for (i = 0; i < n; i++)
    data[i] = 10 * i;

printf("++*data = %d\n", ++*data);
printf("*++data = %d\n", *++data);

return EXIT_SUCCESS;
}

void error(char *str) {
    fprintf(stderr, "ERROR: %s\n", str);
    exit(EXIT_FAILURE);
}

```

Изпълнение на програмата:

```

Enter n>0: 3
++*data = 1
*++data = 10

```

Програма PointersAndFunctions.c

```

/*
=====
Name      : PointersAndFunctions.c
Version   : 04.2011
Description : Pointers and Functions
=====

#include <stdio.h>
#include <stdlib.h>

typedef void (*menuFunction) ();
menuFunction command[4];

char *item[5];

void assignItems();
void assignFunctions();
void function1();
void function2();
void function3();
void function4();

int main(void) {

```

```

int choice = 1;
assignItems();
assignFunctions();
while (choice) {
    printf("%s", item[0]);
    printf("%s", item[1]);
    printf("%s", item[2]);
    printf("%s", item[3]);
    printf("%s", item[4]);
    fflush(stdout);
    scanf("%d", &choice);
    if (choice >= 1 && choice <= 4)
        command[choice - 1]();
    else
        break;
}

return EXIT_SUCCESS;
}

void assignItems() {
    item[0] = "      Menu item 1 \n";
    item[1] = "      Menu item 2 \n";
    item[2] = "      Menu item 3 \n";
    item[3] = "      Menu item 4 \n";
    item[4] = "      Enter your choice: ";
}

void assignFunctions() {
    command[0] = function1;
    command[1] = function2;
    command[2] = function3;
    command[3] = function4;
}

void function1() {
    printf("Activated menu item 1\n");
}

void function2() {
    printf("Activated menu item 2\n");
}

void function3() {
    printf("Activated menu item 3\n");
}

void function4() {
    printf("Activated menu item 4\n");
}

```

Задача: Да се реализират функциите:

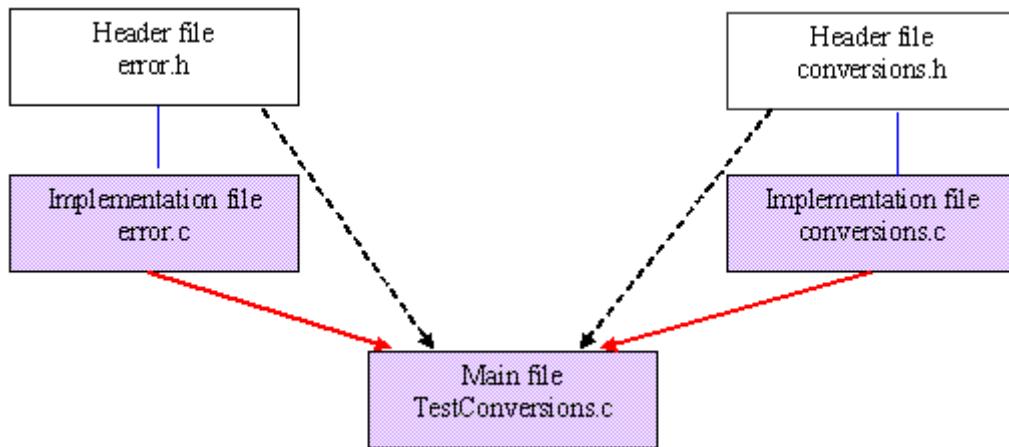
```

char *intToBinaryString(int n) {...}
//Определя двоичен низ – вътрешното представяне на п
int binaryStringToInt(char *str) {...}
//Определя стойност на типа int с вътрешно представяне двоичния низ str
char *floatToBinaryString(float n) {...}
//Определя двоичен низ – вътрешното представяне на п
float binaryStringToFloat(char *str) {...}
//Определя стойност на типа float с вътрешно представяне двоичния низ str
char *intToDecimalString(int n) {...}
//Преобразува п в символен вид
int decimalStringToInt(char *str) {...}

```

//Преобразува цяло десетично без знак в символен вид от str в стойност на типа int

Проект **TestConversions**:



Програма VarArgs.c – виж http://www.acm.uiuc.edu/webmonkeys/book/c_guide/2.10.html

```

/*
=====
Name      : VarArgs.c
Version   : 03.2011
Description : Variable length argument list
=====

*/
#include <stdlib.h>
#include <stdio.h>
#include <stdarg.h>
#include <time.h>
#include <math.h>

int sum(int argc, ...);
/*
 * Returns the sum of a variable number of arguments of type int.
 * Parameter args contains the number of actual arguments
 */
int print(char *format, ...);
/*
 * Outputs the type and contents of a variable number of arguments
 * with different types. Parameter format is a string containing one
 * character for each actual argument type:
 * 1. 'c' : the argument type is char
 * 2. 's' : the argument type is *char (string)
 * 3. 'd' : the argument type is int
 * 4. 'f' : the argument type is double
 * 5. 'n' : new line
 */
int main() {
    srand(time(0));
    int n = rand() % 3 + 1;
    print("sdn", "Number of integers is ", n);
    int n1, n2, n3, s;
    switch (n) {
        case 1:
            n1 = rand();
            s = sum(1, n1);
            print("d", s);
            break;
        case 2:
            n1 = rand();
            n2 = rand();
            s = sum(2, n1, n2);
            print("dd", s);
            break;
        case 3:
            n1 = rand();
            n2 = rand();
            n3 = rand();
            s = sum(3, n1, n2, n3);
            print("ddd", s);
            break;
    }
}

```

```

    print("sdsdn", "sum(", n1, ") = ", s);
    break;
case 2:
    n1 = rand();
    n2 = rand();
    s = sum(2, n1, n2);
    print("sdsdsdn", "sum(", n1, ", ", n2, ") = ", s);
    break;
case 3:
    n1 = rand();
    n2 = rand();
    n3 = rand();
    s = sum(3, n1, n2, n3);
    print("sdsdsdsdn", "sum(", n1, ", ", n2, ", ", n3, ") = ", s);
    break;
}

return EXIT_SUCCESS;
}

int sum(int argc, ...) {
    int result = 0;
    va_list argptr;

    //Initialize argument pointer
    va_start(argptr, argc);

    //Loop through arguments and add to result
    int i;
    for (i = 0; i < argc; i++)
        result += va_arg(argptr, int);

    //Clean up and return result
    va_end(argptr);

    return result;
}

int print(char *format, ...) {
    int n = 0;
    va_list argptr;

    //Initialize argument pointer
    va_start(argptr, format);

    //Loop through arguments and print
    while (format[n]) {
        switch (format[n]) {
        case 'c':
            printf("%c", va_arg(argptr, int)); //compile error if char
            break;
        case 's':
            printf("%s", va_arg(argptr, char *));
            break;
        case 'd':
            printf("%d", va_arg(argptr, int));
            break;
        case 'f':
            printf("%f", va_arg(argptr, double));
            break;
        case 'n':
            printf("\n");
            break;
        default:
            printf("Error: unsupported type argument %c\n", format[n]);
            return EXIT_FAILURE;
        }
    }
}

```

```
        }
        n++;
    }

//Clean up and return result
va_end(argptr);

return EXIT_SUCCESS;
}
```

Изпълнение на програмата:

```
Number of integers is 3
sum(18467, 6334, 26500) = 51301
```

Структури. Основни операции. Вложени структури. Функции и структури. Структури и масиви. Обединения. Дефиниране имена на типове. Изброен тип

Програма Addressing.c

```
/*
=====
Name      : Addressing.c
Version   : 03.2011
Description : Addressing
=====
*/
#include <stdio.h>
#include <stdlib.h>

union {
    long Long;
    char Char[sizeof(long)];
} u;

int main(void) {
    u.Long = 1;
    if (u.Char[0] == 1)
        printf("Addressing is right-to-left\n");
    else if (u.Char[sizeof(long) - 1] == 1)
        printf("Addressing is left-to-right\n");
    else
        printf("Addressing is strange\n");

    return EXIT_SUCCESS;
}
```

Изпълнение на програмата:

Addressing is right-to-left

Програма QuickGenericSorting.c: Използване на стандартната функция от stdlib.h с прототип

void qsort(void *base, size_t nmemb, size_t size,int(*compar)(const void *, const void *));
за сортиране на данни от произволен тип

```
/*
=====
Name      : QuickGenericSorting.c
Version   : 04.2011
Description : Usage of quick generic sorting function
=====
*/
#include <stdio.h>
#include <stdlib.h>

int IntegerCompareTo(const void *arg1, const void *arg2);

int main(void) {
    int array[10];
    int i;
    for (i = 0; i < 10; ++i)
        array[i] = 10 - i;
    qsort(array, 10, sizeof(int), IntegerCompareTo);
```

```

for (i = 0; i < 10; ++i)
    printf("%d\n", array[i]);

return EXIT_SUCCESS;
}

int IntegerCompareTo(const void *arg1, const void *arg2) {
    int *r1 = (int *)arg1;
    int *r2 = (int *)arg2;
    int first = *r1;
    int second = *r2;
    return first - second;
}

```

Задача: Да се състави програма, която чете за всеки от N на брой студенти факултетен номер, име, дата на раждане, адрес и 10 оценки от изпити и предоставя следните справки:

1. Справка 1. Извежда списък от данните за студентите, сортиран по имената
2. Справка 2. Извежда списък, съдържащ за всеки студент – факултетен номер и среден успех, сортиран по средния успех
3. Справка 3. Извежда списък от данните за студентите и среден успех, сортиран по факултетните номера
4. Справка 4. Извежда:
 - Брой студенти със среден успех в интервала [3,00; 3,50) и сортиран списък от факултетните номера на студентите с такъв успех
 - Брой студенти със среден успех в интервала [3,50; 4,50) и сортиран списък от факултетните номера на студентите с такъв успех
 - Брой студенти със среден успех в интервала [4,50; 5,50) и сортиран списък от факултетните номера на студентите с такъв успех
 - Брой студенти със среден успех в интервала [5,50; 6,00] и сортиран списък от факултетните номера на студентите с такъв успех

Програма StudentStructure.c

```

/*
=====
Name      : StudentStructure.c
Version   : 04.2011
Description : Students
=====
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct date {
    int day;
    int month;
    int year;
};

typedef struct date Date;

struct address {
    int houseNumber;
    char street[50];
    int code;
    char country[20];
};

typedef struct address Address;

```

```

struct student {
    int fn;
    char name[50];
    Date birthday;
    Address addr;
    int mark[10];
};

typedef struct student Student;

#define MAX_SIZE 100
Student list[MAX_SIZE];
int n = MAX_SIZE;

void readList();
void printList(Student list[], int size);
int StudentCompareToN(const void *arg1, const void *arg2);
void function1();

#define MAX_ITEMS 3

typedef void (*menuFunction)();
menuFunction command[MAX_ITEMS - 1];

char *item[MAX_ITEMS];

void assignItems();
void assignFunctions();

int main(void) {
    int choice = 1;
    assignItems();
    assignFunctions();
    while (choice) {
        printf("%s", item[0]);
        printf("%s", item[1]);
        printf("%s", item[2]);
        fflush(stdout);
        scanf("%d", &choice);
        if (choice >= 1 && choice <= MAX_ITEMS - 1)
            command[choice - 1]();
        else
            break;
    }
    return EXIT_SUCCESS;
}

void assignItems() {
    item[0] = "      1. Read data\n";
    item[1] = "      2. List of students sorted by name\n";
    item[2] = "      Enter your choice: ";
}

void assignFunctions() {
    command[0] = readList;
    command[1] = function1;
}

void readList() {
    printf("      Enter N>0: ");
    fflush(stdout);
    scanf("%d", &n);

    printf("Please enter the student information:\n");
    fflush(stdout);
    int i;
}

```

```

for (i = 0; i < n; i++) {
    printf("\nFN:");
    fflush(stdout);
    scanf("%d", &list[i].fn);

    printf("\nName:");
    fflush(stdout);
    scanf("%s", list[i].name);

    printf("\nBirthday:");
    fflush(stdout);
    scanf("%d", &list[i].birthday.day);
    scanf("%d", &list[i].birthday.month);
    scanf("%d", &list[i].birthday.year);

    printf("\nAddress:");
    fflush(stdout);
    scanf("%d", &list[i].addr.houseNumber);
    scanf("%s", list[i].addr.street);
    scanf("%d", &list[i].addr.code);
    scanf("%s", list[i].addr.country);

    printf("\nMark:");
    fflush(stdout);
    scanf("%d", &list[i].mark[0]);
}

}

void printList(Student list[], int size) {
    printf("Students' information:\n");
    int i;
    for (i = 0; i < size; i++) {
        printf("\nFN: %d", list[i].fn);
        printf("\nName: %s", list[i].name);
        printf("\nBirthday: %d, %d, %d", list[i].birthday.day,
               list[i].birthday.month, list[i].birthday.year);
        printf("\nAddress: %d, %s, %d, %s\n", list[i].addr.houseNumber,
               list[i].addr.street, list[i].addr.code, list[i].addr.country);
    }
}

int StudentCompareToN(const void *arg1, const void *arg2) {
    Student *r1 = (Student *) arg1;
    Student *r2 = (Student *) arg2;
    char *first = r1->name;
    char *second = r2->name;
    return strcmp(first, second);
}

void function1() {
    qsort(list, n, sizeof(Student), StudentCompareToN);
    printList(list, n);
}

```

Структури от данни. Родови (универсални, с обикновено предназначение) единици

Задача: Да се реализира тип **рационално число**.

Решение:

1. Множество от стойности $D = \{ \frac{p}{q} \mid p \text{ и } q > 0 \text{ са взаимно прости цели числа} \}$
2. Операции:
 - Създаване на рационално число
 - Унищожаване на рационално число
 - Аритметични операции
 - Умножение на рационални числа
 - Деление на рационални числа
 - Събиране на рационални числа
 - Изваждане на рационални числа
 - Операции за сравнение на рационални числа ==, !=, <, <=, > и >=
 - Операции за вход и изход
 - Четене на рационално число от стандартния вход
 - Извеждане на рационално число на стандартния изход
 - Операции за преобразуване
 - Цяло число в рационално число
 - Рационално число в цяло число
 - Рационално число в реално число
 - Рационално число в низ
 - Достъп до данните

Реализация, съгласно спецификацията:

Дефиниране на тип Ratio

```
//Създаване
Ratio RatioNew(int p, int q);

//Унищожаване (при необходимост)
void RatioDelete(Ratio r);

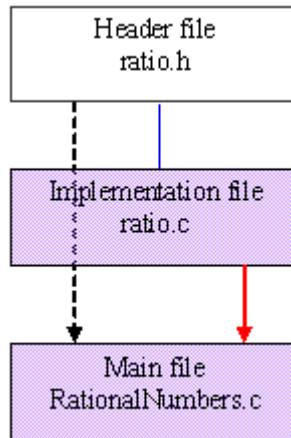
//Операции
Ratio RatioAdd(Ratio r1, Ratio r2);
Ratio RatioDiv(Ratio r1, Ratio r2);
Ratio RatioSubtr(Ratio r1, Ratio r2);
Ratio RatioMult(Ratio r1, Ratio r2);
int RatioCompareTo(Ratio r1, Ratio r2);

//Вход/изход
Ratio RatioRead();
void RatioPrint(Ratio r);

//Достъп до данните (при необходимост)
int RatioGetNum(Ratio r);
void RatioSetNum(Ratio r, int newP);
int RatioGetDen(Ratio r);
void RatioSetDen(Ratio r, int newQ);

//Преобразувания
double Ratio.ToDouble(Ratio r);
int RatioToInt(Ratio r);
char *RatioToString(char *str, Ratio r);
```

Проект RationalNumbers:

**Вариант 1***Интерфейс:*

```

/*
 * ratio.h
 *
 * Created on: 04.2011
 *      Ratio Interface
 */

#ifndef RATIO_H_
#define RATIO_H_

struct ratio {
    int p;
    int q;
};

typedef struct ratio Ratio;

//New
Ratio RatioNew(int p, int q);

//Operations
Ratio RatioAdd(Ratio r1, Ratio r2);
Ratio RatioDiv(Ratio r1, Ratio r2);
Ratio RatioSubtr(Ratio r1, Ratio r2);
Ratio RatioMult(Ratio r1, Ratio r2);
int RatioCompareTo(Ratio r1, Ratio r2);

//Input/Output
Ratio RatioRead();
void RatioPrint(Ratio r);

//Conversions
double Ratio.ToDouble(Ratio r);
int RatioToInt(Ratio r);
char *RatioToString(char *str, Ratio r);

#endif /* RATIO_H_ */

```

Реализация:

```

/*
 * ratio.c
 *
 * Created on: 04.2011
 */

```

```

*      Ratio Implementation
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "ratio.h"

//Private
static int gcd(int a, int b) {
    if (a == 0)
        b = 1;
    else {
        a = fabs(a);
        b = fabs(b);
        int p = a % b;
        while (p != 0) {
            a = b;
            b = p;
            p = a % b;
        }
    }
    return b;
}

//New
Ratio RatioNew(int p, int q) {
    Ratio obj;
    int a = gcd(p, q);
    obj.p = p / a;
    obj.q = q / a;
    return obj;
}

//Operations
Ratio RatioAdd(Ratio r1, Ratio r2) {
    return RatioNew(r1.p * r2.q + r1.q * r2.p, r1.q * r2.q);
}

Ratio RatioDiv(Ratio r1, Ratio r2) {
    if (r2.p == 0){
        Ratio r={0,0};
        return r;
    }
    int p = r1.p * r2.q;
    int q = fabs(r1.q * r2.p);
    return r2.p < 0 ? RatioNew(-p, q) : RatioNew(p, q);
}

Ratio RatioSubtr(Ratio r1, Ratio r2) {
    printf("Unsupported Operation: subtr!");
    Ratio r={0,0};
    return r;
}

Ratio RatioMult(Ratio r1, Ratio r2) {
    printf("Unsupported Operation: mult!");
    Ratio r={0,0};
    return r;
}

int RatioCompareTo(Ratio r1, Ratio r2) {
    return r1.p * r2.q - r1.q * r2.p;
}

```

```

}

//Input/Output
Ratio RatioRead() {
    int p, q;
    scanf("%d%d", &p, &q);
    return RatioNew(p, q);
}

void RatioPrint(Ratio r) {
    printf("%d/%d\n", r.p, r.q);
}

//Conversions
double RatiotoDouble(Ratio r) {
    return (double) r.p / (double) r.q;
}

int RatioToInt(Ratio r) {
    return r.p / r.q;
}

char *RatioToString(char *str, Ratio r) {
    sprintf(str, "%d/%d\n", r.p, r.q);
    return str;
}

```

Текущо: Програма RationalNumbers.c

```

/*
=====
Name      : RationalNumbers.c
Version   : 04.2011
Description : Usage of Rational Numbers
=====
*/
#include <stdio.h>
#include <stdlib.h>
#include "ratio.h"

#define MAX_SIZE 100

Ratio sum(Ratio *arg, int n);

int main(void) {
    Ratio array[MAX_SIZE];
    int i;
    array[0]=RatioNew(0,1);
    for (i = 1; i < 4; i++)
        array[i] = RatioNew(1, i);
    RatioPrint(sum(array, 4));

    return EXIT_SUCCESS;
}

Ratio sum(Ratio *arg, int n) {
    Ratio result = arg[0];
    int i;
    for (i = 1; i < n; i++)
        result = RatioAdd(result, arg[i]);
    return result;
}

```

Вариант 2 – невъзможно в MinGW

Интерфейс:

```

/*
 * ratio.h
 *
 * Created on: 04.2011
 *      Ratio Interface
 */

#ifndef RATIO_H_
#define RATIO_H_

typedef struct ratio Ratio;

//New
Ratio RatioNew(int p, int q);

//Operations
Ratio RatioAdd(Ratio r1, Ratio r2);
Ratio RatioDiv(Ratio r1, Ratio r2);
Ratio RatioSubtr(Ratio r1, Ratio r2);
Ratio RatioMult(Ratio r1, Ratio r2);
int RatioCompareTo(Ratio r1, Ratio r2);

//Input/Output
Ratio RatioRead();
void RatioPrint(Ratio r);

//Data access
int RatioGetNum(Ratio r);
void RatioSetNum(Ratio r, int newP);
int RatioGetDen(Ratio r);
void RatioSetDen(Ratio r, int newQ);

//Conversions
double Ratio.ToDouble(Ratio r);
int RatioToInt(Ratio r);
char *RatioToString(char *str, Ratio r);

#endif /* RATIO_H_ */

```

Реализация:

```

/*
 * ratio.c
 *
 * Created on: 04.2011
 *      Ratio Implementation - както във Вариант 1
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "ratio.h"

struct ratio {
    int p;
    int q;
};

//Private ...

//New ...

```

```
//Operations ...
//Input/Output ...
//Data access ...
//Conversions ...
```

Вариант 3, включен в Проекта CDataStructures

Интерфейс:

```
/*
 * ratio.h
 *
 * Created on: 04.2011
 *      Ratio Interface
 */

#ifndef RATIO_H_
#define RATIO_H_

typedef struct ratio *Ratio;

//New
Ratio RatioNew(int p, int q);

//Delete
void RatioDelete(Ratio r);

//Operations
Ratio RatioAdd(Ratio r1, Ratio r2);
Ratio RatioDiv(Ratio r1, Ratio r2);
Ratio RatioSubtr(Ratio r1, Ratio r2);
Ratio RatioMult(Ratio r1, Ratio r2);
int RatioCompareTo(Ratio r1, Ratio r2);

//Input/Output
Ratio RatioRead();
void RatioPrint(Ratio r);

//Data access
int RatioGetNum(Ratio r);
void RatioSetNum(Ratio r, int newP);
int RatioGetDen(Ratio r);
void RatioSetDen(Ratio r, int newQ);

//Conversions
double RatiotoDouble(Ratio r);
int RatioToInt(Ratio r);
char *RatioToString(char *str, Ratio r);

#endif /* RATIO_H_ */
```

Реализация:

```
/*
 * ratio.c
 *
 * Created on: 04.2011
 *      Ratio Implementation
 */

#include <stdio.h>
```

```

#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "ratio.h"

struct ratio {
    int p;
    int q;
};

//Private
static int gcd(int a, int b) {
    if (a == 0)
        b = 1;
    else {
        a = fabs(a);
        b = fabs(b);
        int p = a % b;
        while (p != 0) {
            a = b;
            b = p;
            p = a % b;
        }
    }
    return b;
}

//New
Ratio RatioNew(int p, int q) {
    Ratio obj = malloc(sizeof(struct ratio));
    int a = gcd(p, q);
    obj->p = p / a;
    obj->q = q / a;
    return obj;
}

void RatioDelete(Ratio r) {
    free(r);
}

//Operations
Ratio RatioAdd(Ratio r1, Ratio r2) {
    return RatioNew(r1->p * r2->q + r1->q * r2->p, r1->q * r2->q);
}

Ratio RatioDiv(Ratio r1, Ratio r2) {
    if (r2->p == 0)
        return NULL;
    int p = r1->p * r2->q;
    int q = fabs(r1->q * r2->p);
    return r2->p < 0 ? RatioNew(-p, q) : RatioNew(p, q);
}

Ratio RatioSubtr(Ratio r1, Ratio r2) {
    printf("Unsupported Operation: subtr!");
    return NULL;
}

Ratio RatioMult(Ratio r1, Ratio r2) {
    printf("Unsupported Operation: mult!");
    return NULL;
}

int RatioCompareTo(Ratio r1, Ratio r2) {

```

```

    return r1->p * r2->q - r1->q * r2->p;
}

//Input/Output
Ratio RatioRead() {
    int p, q;
    scanf("%d%d", &p, &q);
    return RatioNew(p, q);
}

void RatioPrint(Ratio r) {
    printf("%d/%d\n", r->p, r->q);
}

//Data access
int RatioGetNum(Ratio r) {
    return r->p;
}
void RatioSetNum(Ratio r, int newP) {
    int a = gcd(newP, r->q);
    r->p = newP / a;
    r->q = r->q / a;
}

int RatioGetDen(Ratio r) {
    return r->q;
}

void RatioSetDen(Ratio r, int newQ) {
    int a = gcd(r->p, newQ);
    r->p = r->p / a;
    r->q = newQ / a;
}

//Conversions
double RatiotoDouble(Ratio r) {
    return (double) r->p / (double) r->q;
}

int RatioToInt(Ratio r) {
    return r->p / r->q;
}

char *RatioToString(char *str, Ratio r) {
    sprintf(str, "%d/%d\n", r->p, r->q);
    return str;
}

```

Задача: Да се реализира родова функция за сортиране, съгласно дадения интерфейс:

```

/*
 * genericSort.h
 *
 * Created on: 04.2011
 *      Generic Sort Interface
 */
#ifndef GENERICSORT_H_
#define GENERICSORT_H_

typedef int (*compare)(const void *, const void *);

void sort(void *base, int n, int elementSize, compare cmp);

```

```
#endif /* GENERICSORT_H_ */
```

Реализация, включена в проекта **CDataStructures**:

```
/*
 * genericssort.c
 *
 * Created on: 04.2011
 *      Generic Sort Implementation
 */

#include "genericssort.h"

static void swap(char *arg1, char *arg2, int elementSize);

void sort(void *base, int n, int elementSize, compare cmp) {
    int i, sorted;
    do {
        sorted = 1;
        for (i = 0; i < n - 1; i++) {
            char *e1 = base + i * elementSize;
            char *e2 = base + (i + 1) * elementSize;
            if ((*cmp)(e1, e2) > 0) {
                swap(e1, e2, elementSize);
                sorted = 0;
            }
        }
    } while (!sorted);
}

static void swap(char *arg1, char *arg2, int elementSize) {
    int i;
    for (i = 0; i < elementSize; i++) {
        char temp = arg1[i];
        arg1[i] = arg2[i];
        arg2[i] = temp;
    }
}
```

Задача: Да се реализира **родов масив** (с елементи от произволен тип), съгласно дадения интерфейс:

```
/*
 * array.h
 *
 * Created on: 04.2011
 *      Array Interface
 *
 */

#ifndef ARRAY_H_
#define ARRAY_H_

typedef struct array *Array;

//New
Array ArrayNew();

//Delete
void ArrayDelete(Array array);

//Add
void ArrayAdd(Array array, void *datap);
```

```
//Data access
void *ArrayGetElements(Array array);
int ArrayGetSize(Array array);
int ArrayGetLength(Array array);
void *ArrayGetElementAt(Array array, int i);
void *ArraySetElementAt(Array array, int i, void *datap);

#endif /* ARRAY_H_ */
```

Реализация, включена в проекта **CDataStructures**:

```
/*
 * array.c
 *
 * Created on: 04.2011
 *      Array Implementation
 */

#include <stdlib.h>
#include "array.h"

#define MAX_SIZE 100

struct array {
    void *elements[MAX_SIZE];
    int num_elements;
};

//New
Array ArrayNew() {
    Array array = malloc(sizeof(struct array));
    array->num_elements = 0;
    return array;
}

//Delete
void ArrayDelete(Array array) {
    free(array);
}

//Add
void ArrayAdd(Array array, void *datap) {
    int index = array->num_elements;
    array->elements[index] = datap;
    array->num_elements++;
}

//Data access
void *ArrayGetElements(Array array) {
    return array->elements;
}

int ArrayGetSize(Array array) {
    return MAX_SIZE;
}

int ArrayGetLength(Array array) {
    return array->num_elements;
}

void *ArrayGetElementAt(Array array, int i) {
    return array->elements[i];
}

void *ArraySetElementAt(Array array, int i, void *datap) {
    void *result = array->elements[i];
    array->elements[i] = datap;
    return result;
}
```

```

array->elements[i] = datap;
return result;
}

```

Задача: Да се реализира функция, която създава масив от реални числа и го сортира, съгласно дадената спецификация:

```

/*
 * doublenumbers.h
 *
 * Created on: 04.2011
 *
 */
#ifndef DOBLENUMBERS_H_
#define DOBLENUMBERS_H_

void DoubleNumbersSorting();

#endif /* DOBLENUMBERS_H_ */

```

Реализация, включена в проекта **CDataStructures**:

```

/*
 * double.h
 *
 * Created on: 04.2011
 *      Double Interface
 */

#ifndef DOUBLE_H_
#define DOUBLE_H_

typedef struct real *Double;

//New
Double DoubleNew(double p);

//Delete
void DoubleDelete(Double r);

//Conversions
double DoubleToDouble(Double r);

#endif /* DOUBLE_H_ */

/*
 * double.c
 *
 * Created on: 04.2011
 *      Double Implementation
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include "double.h"

struct real {
    double r;
};

//New
Double DoubleNew(double p) {
    Double obj = malloc(sizeof(struct real));

```

```

obj->r = p;
return obj;
}

//Delete
void DoubleDelete(Double r) {
    free(r);
}

//Conversions
double DoubleToDouble(Double r) {
    return r->r;
}

/*
 * doublenumbers.c
 *
 * Created on: 04.2011
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include "error.h"
#include "array.h"
#include "genericsort.h"
#include "double.h"

static Array array;
static int n;

static void DoubleArrayNew();
static void DoubleArrayPrint();
static void DoubleArrayDelete();
static int DoubleCmp(const void *arg1, const void *arg2);

void DoubleNumbersSorting() {
    DoubleArrayNew();
    printf("      Double Numbers\n");
    DoubleArrayPrint();

    printf("      Sorted Double Numbers\n");
    sort(ArrayGetElements(array), ArrayGetLength(array), sizeof(Double),
        DoubleCmp);
    DoubleArrayPrint();

    DoubleArrayDelete();
}

static void DoubleArrayNew() {
    array = ArrayNew();
    printf("      Enter n>0: ");
    fflush(stdout);
    if (scanf("%d", &n) == 0 || n < 1 || n > ArrayGetSize(array))
        error("invalid n");
    int i;
    double f;
    for (i = 1; i <= n; i++) {
        printf("Double: ");
        fflush(stdout);
        scanf("%lg", &f);
        ArrayAdd(array, DoubleNew(f));
    }
}

static void DoubleArrayPrint() {

```

```

int i;
for (i = 0; i < ArrayGetLength(array); i++)
    printf("%lg\n", DoubleToDouble(ArrayGetElementAt(array, i)));
}

static void DoubleArrayDelete() {
    int i;
    for (i = 0; i < ArrayGetLength(array); i++)
        DoubleDelete(ArrayGetElementAt(array, i));
    ArrayDelete(array);
}

static int DoubleCmp(const void *arg1, const void *arg2) {
    Double *r1 = (Double *) arg1;
    Double *r2 = (Double *) arg2;
    double left = DoubleToDouble(*r1);
    double right = DoubleToDouble(*r2);
    if (left < right)
        return -1;
    else if (left > right)
        return 1;
    else
        return 0;
}

```

Задача: Да се реализира функция, която създава масив от **рационални числа** и го сортира, съгласно дадената спецификация

```

/*
 * rationumbers.h
 *
 * Created on: 04.2011
 *
 */

#ifndef RATIONUMBERS_H_
#define RATIONUMBERS_H_

void RatioNumbersSorting();

#endif /* RATIONUMBERS_H_ */

```

*Реализация, включена в проекта **CDataStructures**:*

```

/*
 * rationumbers.c
 *
 * Created on: 04.2011
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include "error.h"
#include "array.h"
#include "genericsort.h"
#include "ratio.h"

static Array array;
static int n;

static void RatioArrayNew();
static void RatioArrayPrint();
static void RatioArrayDelete();
static int RatioCmp(const void *arg1, const void *arg2);

```

```

void RatioNumbersSorting() {
    RatioArrayNew();
    printf("      Rational Numbers\n");
    RatioArrayPrint();

    printf("      Sorted Rational Numbers\n");
    sort(ArrayGetElements(array), ArrayGetLength(array), sizeof(Ratio),
          RatioCmp);
    RatioArrayPrint();

    RatioArrayDelete();
}

static void RatioArrayNew() {
    array = ArrayNew();
    printf("      Enter n>0: ");
    fflush(stdout);
    if (scanf("%d", &n) == 0 || n < 1 || n > ArrayGetSize(array))
        error("invalid n");
    int i;
    for (i = 1; i <= n; i++) {
        printf("Ratio: ");
        fflush(stdout);
        ArrayAdd(array, RatioRead());
    }
}

static void RatioArrayPrint() {
    int i;
    for (i = 0; i < ArrayGetLength(array); i++)
        RatioPrint(ArrayGetElementAt(array, i));
}

static void RatioArrayDelete() {
    int i;
    for (i = 0; i < ArrayGetLength(array); i++)
        RatioDelete(ArrayGetElementAt(array, i));
    ArrayDelete(array);
}

static int RatioCmp(const void *arg1, const void *arg2) {
    Ratio *r1 = (Ratio *) arg1;
    Ratio *r2 = (Ratio *) arg2;
    return RatioCompareTo(*r1, *r2);
}

```

Задача: Да се реализира **двойка от вида (низ, цяло число)**, съгласно дадения интерфейс:

```

/*
 * pair.h
 *
 * Created on: 09.04.2011
 *      Pair Interface
 */

#ifndef PAIR_H_
#define PAIR_H_

typedef struct pair *Pair;

//New
Pair PairNew(char *p, int q);

//Delete
void PairDelete(Pair r);

```

```
//Operations
int PairCompareToN(Pair r1, Pair r2);
int PairCompareToV(Pair r1, Pair r2);

//Input/Output
Pair PairRead();
void PairPrint(Pair r);

//Data access
char *PairGetN(Pair r);
void PairSetN(Pair r, char *newN);
int PairGetV(Pair r);
void PairSetV(Pair r, int newV);

//Conversions
char *PairToString(char *str, Pair r);

#endif /* PAIR_H_ */
```

Реализацията да се включи в проекта **CDataStructures**.

Задача: Да се реализира функция, която създава масив от **двойки** и го сортира

- В азбучен ред на низовете в тях
- В нарастващ ред на числата в тях

съгласно дадената спецификация:

```
/*
 * pairobjects.h
 *
 * Created on: 04.2011
 *
 */

#ifndef PAIROBJECTS_H_
#define PAIROBJECTS_H_

void PairsSorting();

#endif /* PAIROBJECTS_H_ */
```

Реализацията да се включи в проекта **CDataStructures**.

Реализация на програма за тестване, включена в проекта **CDataStructures**:

```
/*
=====
Name      : CDataStructures.c
Version   : 04.2011
Description : Tests
=====
*/
#include <stdio.h>
#include <stdlib.h>
#include "doublenumbers.h"
#include "rationumbers.h"
#include "pairobjects.h"

#define MAX_ITEMS 4

typedef void (*menuFunction)();
menuFunction command[MAX_ITEMS - 1];
```

```

char *item[MAX_ITEMS];

void assignItems();
void assignFunctions();

int main(void) {
    int choice = 1;
    assignItems();
    assignFunctions();
    while (choice) {
        printf("%"s, item[0]);
        printf("%"s, item[1]);
        printf("%"s, item[2]);
        printf("%"s, item[3]);
        fflush(stdout);
        scanf("%"d, &choice);
        if (choice >= 1 && choice <= MAX_ITEMS - 1)
            command[choice - 1]();
        else
            break;
    }

    return EXIT_SUCCESS;
}

void assignItems() {
    item[0] = " 1. Array of doubles generic sorting\n";
    item[1] = " 2. Array of ratio generic sorting\n";
    item[2] = " 3. Array of pairs generic sorting\n";
    item[3] = " Enter your choice: ";
}

void assignFunctions() {
    command[0] = DoubleNumbersSorting;
    command[1] = RatioNumbersSorting;
    command[2] = PairsSorting;
}

```

Изпълнение на програмата:

```

1. Array of doubles generic sorting
2. Array of ratio generic sorting
3. Array of pairs generic sorting
    Enter your choice: 1
    Enter n>0: 3
Double: 22.5
Double: 21
Double: 1e-3
    Double Numbers
22.5
21
0.001
    Sorted Double Numbers
0.001
21
22.5
    1. Array of doubles generic sorting
    2. Array of ratio generic sorting
    3. Array of pairs generic sorting
        Enter your choice: 2
        Enter n>0: 3
Ratio: 1 2
Ratio: 1 3
Ratio: 1 4
    Rational Numbers

```

```
1/2
1/3
1/4
    Sorted Rational Numbers
1/4
1/3
1/2
    1. Array of doubles generic sorting
    2. Array of ratio generic sorting
    3. Array of pairs generic sorting
        Enter your choice: 3
        Enter n>0: 2
Pair:
seven
7
Pair:
nine
9
    Pair Objects
(seven, 7)
(nine, 9)
    Sorted Pair Objects by name
(nine, 9)
(seven, 7)
    Sorted Pair Objects by value
(seven, 7)
(nine, 9)
    1. Array of doubles generic sorting
    2. Array of ratio generic sorting
    3. Array of pairs generic sorting
        Enter your choice: 7
```

Файлове. Текстови и двоични файлове. Етапи за работа с файлове.

Функции за буфериран вход и изход

Програма SourceFileName.c

```
/*
=====
Name      : SourceFileName.c
Version   : 04.2011
Description : Outputs the source file name
=====
*/
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    printf("The source file name is %s\n", __FILE__);
    return EXIT_SUCCESS;
}
```

Изпълнение на програмата:

```
The source file name is ..\src\SourceFileName.c
```

Програма CFilesExample1.c

```
/*
=====
Name      : CFilesExample1.c
Version   : 04.2011
Description : Formatted Input: fscanf, sscanf, scanf
              : Formatted Output: fprintf, sprintf, printf
=====
*/
#include <stdio.h>
#include <stdlib.h>

void error(char *str);

int main(void) {
    char name[30];
    FILE *fp;
    char s[80];
    int n, t;

    printf("Enter file name: ");
    fflush(stdout);
    scanf("%s", name); /* read from keyboard */
    if ((fp = fopen(name, "w")) == NULL)
        error("cannot open file\n");

    printf("Enter n>0: ");
    fflush(stdout);
    scanf("%d", &n); /* read from keyboard */
    fprintf(fp, "%d", n); /* write to file */

    int i;
    for (i = 0; i < n; i++) {
        printf("Enter a string and a number: ");
        fflush(stdout);
        fscanf(stdin, "%s%d", s, &t); /* read from keyboard */
    }
}
```

```

    fprintf(fp, "%s %d", s, t); /* write to file */
}

fclose(fp);

if ((fp = fopen(name, "r")) == NULL)
    error("cannot open file\n");

fscanf(fp, "%d", &n); /* read from file */
fprintf(stdout, "%d", n); /* print on screen */
for (i = 0; i < n; i++) {
    fscanf(fp, "%s%d", s, &t); /* read from file */
    fprintf(stdout, "%s %d", s, t); /* print on screen */
}

fclose(fp);

return EXIT_SUCCESS;
}
}

void error(char *str) {
    fprintf(stderr, "ERROR: %s\n", str);
    exit(EXIT_FAILURE);
}

```

Задача: Да се реализира командата `cat file ...`, която конкатенира файловете и извежда резултата на стандартния изход.

```

/*
=====
Name      : CFilesExample3.c
Version   : 04.2011
Description : Character Input and Output Functions: fgetc,fgets,
              fputc, fputs, ungetc
=====

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifndef mode
#define mode 1
#endif

void error(char *str);
void print(char *fileName);

int main(int argc, char * argv[]) {
    //Conditional compiling

#if mode == 1
    if (argc == 1)
        error("missing files\n");
    else {
        int i;
        for (i = 1; i < argc; i++)
            print(argv[i]);
    }
#elif mode == 0
    char fileName[30];
    scanf("%s", fileName);
    print(fileName);
#else
    error("mode must be 1 or 0");
#endif
}
```

```

    return EXIT_SUCCESS;
}

void error(char *str) {
    fprintf(stderr, "ERROR: %s\n", str);
    exit(EXIT_FAILURE);
}

void print(char *fileName) {
    FILE *fp;
    if ((fp = fopen(fileName, "r")) == NULL)
        error("cannot open file\n");

    int in;
    while ((in = fgetc(fp)) != EOF)
        putchar(in);

    fclose(fp);
}

```

Задача: Да се реализира команда **cp** *file1 file2*, която копира съдържанието на файла *file1* във файла *file2*.

Задача: Да се реализира команда **uniq** *file1 file2*, която копира съдържанието на файла *file1* във файла *file2*, като премахва повтарящите се редове.

Задача: Да се реализира команда **tr** *string1 string2 file*, която заменя във файла *file* всеки символ от *string1* със съответния по място символ от *string2*.

Задача: Да се реализира команда **crypt** *option file1 file2*, която:

- Ако *option* е **-c**, то шифрира *file1* и го записва във *file2*
- Ако *option* е **-d**, то дешифрира *file1* и го записва във *file2*

Задача: Да се реализира команда **comm** *file1 file2*, която сравнява съдържанието на файла *file1* със съдържанието на файла *file2* и извежда:

- редовете само във файла *file1*
- редовете само във файла *file2*
- общите за двета файла редове

Задача: Да се реализира команда **tail** *[+/-number] file*, която извежда:

- *+number* реда от началото към края на файла
- *-number* реда от края към началото на файла
- последните 10 реда, ако този параметър липсва

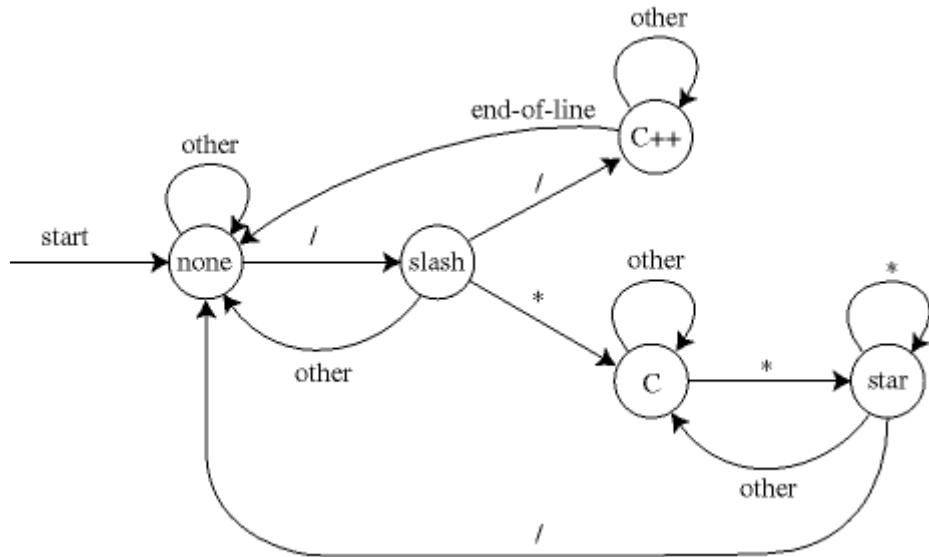
Задача: Да се реализира команда **wc** *file*, която определя и извежда брой символи, брой думи и брой редове във файла *file*.

Задача: Да се реализира команда **replace** *word1 word2 file1 file2*, която копира съдържанието на файла *file1* във файла *file2*, като заменя всяка дума *word1* с думата *word2*.

Задача: Да се реализира команда **sort** *option file*, която сортира редовете на файла *file*:

- в намаляващ ред, ако *option* е **-r**
- в нарастващ ред, ако *option* е **-a**

Задача: Да се състави програма, която премахва коментарите от програма на езика C, като реализацията се основава на използването на дадения краен детерминиран автомат



Задача: Да се реализират и тестват функции за замяна на всяка малка буква от даден файл с главна и обратно, като резултатът се записва в нов файл, с прототипи:

```

void toUpperCase(char *inFile, char *outFile);
void toLowerCase(char *inFile, char *outFile);

```

Задача: Да се състави програма, която чете даден текст, съдържащ цели десетични положителни числа, определя и извежда сумата на числата.

Реализацията се основава на използването на КДА $A = \langle \{q_0, q_1, q_2, q_3\}, \{0, 1\}, q_0, \delta, \{\} \rangle$ с функция на преходите δ :

	Digit	Space	other
q_0	q_1	q_2	q_3
q_1	q_1	q_2	q_3
q_2	q_1	q_2	q_3
q_3	q_3	q_2	q_3

```

/*
=====
Name      : CFilesExample6.c
Version   : 04.2011
Description : Calculator
=====

#include <stdio.h>
#include <stdlib.h>

void error(char *str);
int sum(char *inFile);

int main(void) {
    char inFile[30];
    printf("      Enter input file name: ");
    fflush(stdout);
    scanf("%s", inFile);

    printf("The sum is %d\n", sum(inFile));

    return EXIT_SUCCESS;
}

```

```

void error(char *str) {
    fprintf(stderr, "ERROR: %s\n", str);
    exit(EXIT_FAILURE);
}

int sum(char *inFile) {
    /*
        n      digit      space      other
        0      1          2          3
        1      1          2          3
        2      1          2          3
        3      3          2          3
    */

    FILE *fp;
    if ((fp = fopen(inFile, "r")) == NULL)
        error("cannot open file\n");

    int in, n = 0, result = 0, number = 0;
    while ((in = fgetc(fp)) != EOF) {
        switch (n) {
            case 0:
                if (in >= '0' && in <= '9') {
                    number = number * 10 + in - '0';
                    n = 1;
                } else if (in == ' ')
                    n = 2;
                else
                    n = 3;
                break;
            case 1:
                if (in >= '0' && in <= '9')
                    number = number * 10 + (in - '0');
                else {
                    result = result + number;
                    number = 0;
                    if (in == ' ')
                        n = 2;
                    else
                        n = 3;
                }
                break;
            case 2:
                if (in >= '0' && in <= '9') {
                    number = number * 10 + in - '0';
                    n = 1;
                } else if (in == ' ')
                    n = 2;
                else
                    n = 3;
                break;
            case 3:
                if (in == ' ')
                    n = 2;
                break;
        }
    }

    fclose(fp);

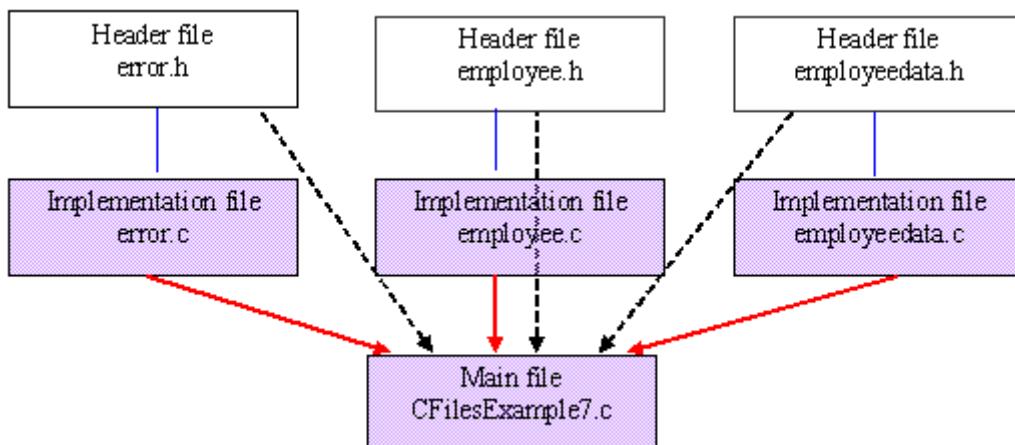
    return result;
}

```

Задача: Да се състави програма за създаване и поддържане на файл с данни за служители в предприятие, като за всеки служител се въвеждат име и заплата и се генерира служебен номер. Програмата да предоставя следните обработки:

1. Първоначално създаване на файла за служителите
2. Извеждане на данните за служителите от файла
3. Включване на нов служител във файла
4. Промяна на данните на служител по зададен служебен номер

Проект CFilesExample7



Реализация на грешка:

```

/*
 * error.h
 *
 * Created on: 04.2011
 *
 */

#ifndef ERROR_H_
#define ERROR_H_

void error(char *str);

#endif /* ERROR_H_ */

/*
 * error.c
 *
 * Created on: 04.2011
 *
 */

#include <stdio.h>
#include <stdlib.h>

void error(char *str) {
    fprintf(stderr, "ERROR: %s\n", str);
    exit(EXIT_FAILURE);
}
  
```

Реализация на служител:

```

/*
 * employee.h
 *
 */
  
```

```

*   Created on: 05.2011
*       Employee Interface
*/
#ifndef EMPLOYEE_H_
#define EMPLOYEE_H_

#define MAX_SIZE 80
struct employee {
    unsigned int number;
    char name[80];
    double salary;
};

typedef struct employee Employee;

//Input/output
Employee EmployeeRead();
void EmployeePrint(Employee e);

#endif /* EMPLOYEE_H_ */

/*
 * employee.c
 *
 * Created on: 05.2011
 *      Employee Implementation
 */

#include <stdio.h>
#include <string.h>
#include "employee.h"

Employee EmployeeRead() {
    Employee e;
    printf("\nName:");
    fflush(stdout);
    scanf("%s", e.name);
    printf("\nSalary:");
    fflush(stdout);
    scanf("%lg", &e.salary);
    return e;
}

void EmployeePrint(Employee e) {
    printf("\nNumber: %d", e.number);
    printf("\nName: %s", e.name);
    printf("\nSalary: %lg\n", e.salary);
}

```

Реализация на двоичен файл с данни за служители:

```

/*
 * employeedata.h
 *
 * Created on: 05.2011
 *      Employee DB Interface
*/
#ifndef EMPLOYEEDATA_H_
#define EMPLOYEEDATA_H_

void createEmployeeDB();
void showEmployeeDB();
void addNewEmployee();

```

```

void updateEmployeeData();

#endif /* EMPLOYEE DATA_H */

/*
 * employeedata.c
 *
 * Created on: 05.2011
 *      Employee DB Implementation
 */

#include <stdio.h>
#include <stdlib.h>
#include "employee.h"
#include "error.h"

#define OBJECT_SIZE sizeof(Employee)
char fileName[30];

static void writeEmployeeData(Employee e, FILE *fp);
static Employee readEmployeeData(FILE *fp);

void createEmployeeDB() {
    printf("Enter file name: ");
    fflush(stdout);
    scanf("%s", fileName);

    FILE *fp;
    if ((fp = fopen(fileName, "wb")) == NULL)
        error("cannot open file\n");
    Employee first;
    first.number = 0;
    writeEmployeeData(first, fp);

    fclose(fp);
}

void showEmployeeDB() {
    FILE *fp;
    if ((fp = fopen(fileName, "rb")) == NULL)
        error("cannot open file\n");
    Employee first = readEmployeeData(fp);
    int n;
    if ((n = first.number) == 0) {
        printf("No data\n");
        fflush(stdout);
    } else {
        int i;
        for (i = 1; i <= n; i++)
            EmployeePrint(readEmployeeData(fp));
    }

    fclose(fp);
}

void addNewEmployee() {
    FILE *fp;
    if ((fp = fopen(fileName, "rb+")) == NULL)
        error("cannot open file\n");
    Employee first = readEmployeeData(fp);
    first.number++;
    fseek(fp, 0, SEEK_SET);
    writeEmployeeData(first, fp);

    printf("Enter employee data:\n");
    Employee e = EmployeeRead();
}

```

```

e.number = first.number;
fseek(fp, first.number * OBJECT_SIZE, SEEK_SET);
writeEmployeeData(e, fp);

fclose(fp);
}

void updateEmployeeData() {
    FILE *fp;
    if ((fp = fopen(fileName, "rb+")) == NULL)
        error("cannot open file\n");
    Employee first = readEmployeeData(fp);
    int n = first.number;

    printf("      Enter employee number:\n");
    fflush(stdout);
    int number;
    scanf("%d", &number);
    if (number < 1 || number > n)
        error("invalid data");
    printf("      Enter employee data:\n");
    Employee e = EmployeeRead();
    e.number = number;
    fseek(fp, number * OBJECT_SIZE, SEEK_SET);
    writeEmployeeData(e, fp);

    fclose(fp);
}

static void writeEmployeeData(Employee e, FILE *fp) {
    fwrite(&e, OBJECT_SIZE, 1, fp);
}

static Employee readEmployeeData(FILE *fp) {
    Employee e;
    fread(&e, OBJECT_SIZE, 1, fp);
    return e;
}

```

Реализация на **CFilesExample7.c**:

```

/*
=====
Name      : CFilesExample7.c
Version   : 05.2011
Description : C Binary Files: Employee DB
=====
*/
#include <stdio.h>
#include <stdlib.h>
#include "employeedata.h"

#define MAX_ITEMS 5

typedef void (*menuFunction)();
menuFunction command[MAX_ITEMS - 1];

char *item[MAX_ITEMS];

void assignItems();
void assignFunctions();

int main(void) {
    int choice = 1;
    assignItems();

```

```
assignFunctions();
while (choice) {
    printf("%s", item[0]);
    printf("%s", item[1]);
    printf("%s", item[2]);
    printf("%s", item[3]);
    printf("%s", item[4]);
    fflush(stdout);
    scanf("%d", &choice);
    if (choice >= 1 && choice <= MAX_ITEMS - 1)
        command[choice - 1]();
    else
        break;
}

return EXIT_SUCCESS;
}

void assignItems() {
    item[0] = "      1. Create DB\n";
    item[1] = "      2. Show DB\n";
    item[2] = "      3. Add new employee\n";
    item[3] = "      4. Update employee data\n";
    item[4] = "          Enter your choice: ";
}

void assignFunctions() {
    command[0] = createEmployeeDB;
    command[1] = showEmployeeDB;
    command[2] = addNewEmployee;
    command[3] = updateEmployeeData;
}
```

Списък, стек, опашка, дек

Задача: Да се реализира функция **IntegersPrinter**, която чете цели положителни числа до въвеждане на числото 0 и извежда числата в ред, обратен на въвеждането, като се използва стек.

Задача: Да се реализира функция **TextPrinter**, която чете цяло число n от стандартния вход и текст от файл и записва последните n реда от текста в друг файл в ред, обратен на четенето, като се използва стек.

Задача: Да се реализира функция **PalindromChecker** за проверка дали дума принадлежи на езика $L = \{\omega @ O(\omega) \mid \omega \in \{a,b\}^*\}$.

Решение: Езикът L се разпознава от стеков автомат с таблица на преходите:

State	read	pop	StackState	push	NewState
0	a		empty	a	1
0	b		empty	b	1
0	@		empty		2
0	other		empty		5 other symbol
1	a		NotEmpty	a	1
1	b		NotEmpty	b	1
1	@		NotEmpty		3
1	other		NotEmpty		5 other symbol
1	\n		NotEmpty		5 missing @
2	\n		empty		4 yes
2	a		empty		5 missing symbols before @
2	b		empty		5 missing symbols before @
2	other		empty		5 other symbol
3	a	a	NotEmpty		3
3	a	b	NotEmpty		5 other symbol
3	a		empty		5 missing symbols after @
3	b	b	NotEmpty		3
3	b	a	NotEmpty		5 other symbol
3	b		empty		5 missing symbols after @
3	\n		empty		4 yes
3	\n		NotEmpty		5 missing symbols after @
3	other		NotEmpty		5 other symbol

Задача: Да се реализира функция **AkermanEvaluator** за итеративно пресмятане на стойността на функцията на Акерман за дадени стойности на променливите, като се използва стек. Дефиницията на функцията на Акерман е:

$$A(m,n) = \begin{cases} n+1, & \text{ако } m=0 \\ A(m-1,1), & \text{ако } m \neq 0 \text{ и } n=0 \\ A(m-1, A(m,n-1)), & \text{ако } m \neq 0 \text{ и } n \neq 0 \end{cases}$$

За някои стойности на m има формули, които може да се използват при тестването:

$$A(0,n)=n+1$$

$$A(3,n)=2^{n+3}-3$$

$$A(1,n)=n+2$$

$$A(2,n)=2n+3$$

$$A(4,n)=2^2-3, \text{ n+2 вложени степени}$$

Задача: Да се реализира клас **PostfixExpressionEvaluator** за пресмятане на стойността на аритметичен израз в обратен полски запис с операнди – цели числа без знак и операции: +, -, *, /.

Решение: Следващата таблица съдържа аритметични изрази в инфиксен запис и съответното им представяне в обратен полски запис:

Инфиксен запис	Обратен полски запис
$2 + (15 - 12) * 17$	2 15 12 -17 *+
B	B
A+B	A B +
A+B+C	A B +C +
A+B*C	A B C *+
A*(B+C)	A B C +*
A/B*C	A B /C *

Алгоритъм за пресмятане на стойността на аритметичен израз в обратен полски запис:

1. Създаване на празен стек
2. За всеки елемент на израза се извършва следното:

Ако елементът е число:

Включване на числото в стека

Иначе:

Ако елементът е операция:

Четене и изключване на десния operand от стека

Четене и изключване на левия operand от стека

Прилагане на операцията върху operandите

Включване на резултата в стека

Иначе:

Има грешка в израза

3. Четене и изключване на число от стека

4. Проверка:

Ако стекът не е празен:

Има грешка в израза

Иначе:

Последното прочетено число е стойността на израза

Пример: Пресмятане на стойността на $2 \ 15 \ 12 \ -17 \ *+$:

```

2
2, 15
2, 15, 12
2, 3      //3=15-12
2, 3, 17
2, 51     //51=3*17

```

```
53      //53=2+51
```

Задача: Да се реализира функция **StringReverser** за получаване на обратния низ на даден низ.

Задача: Да се реализира функция **BracketChecker** за проверка дали в даден низ са използвани правилно лява и дясна скоби (), [], {}.

Задача: Да се реализира функция за проверка дали дума принадлежи на езика $L=\{a^m b^{2m+1} \mid m \geq 0\}$.

Задача: Да се реализира функция за проверка дали дума принадлежи на езика $L=\{a^m b^n \mid m > n \geq 0\}$.

Задача: Да се реализира функция **PostfixToInfixConvertor** за преобразуване на аритметичен израз в обратен полски запис с операнди - цели числа без знак и операции: +, -, *, / в аритметичен израз в инфиксен запис.

Упътване: Използвайте алгоритъма за пресмятане на стойността на аритметичен израз в обратен полски запис, както е показано на примера:

Пример: Преобразуване на $2\ 15\ 12\ -17\ *+:$

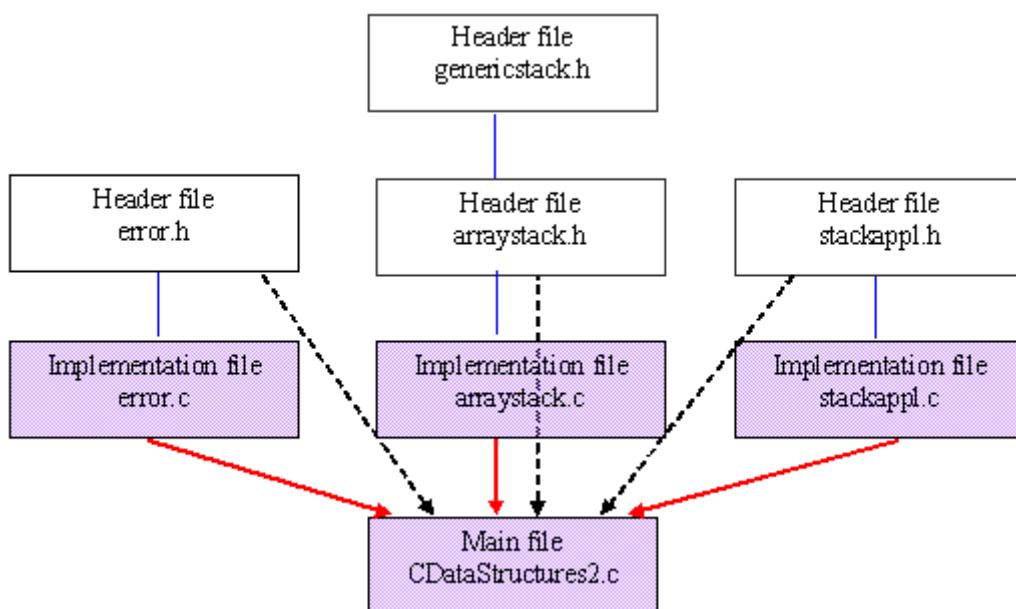
```

2
2, 15
2, 15, 12
2, (15-12)
2, (15-12), 17
2, ((15-12)*17)
(2+((15-12)*17))

```

Задача: Да се реализира функция **InfixToPostfixConvertor** за преобразуване на аритметичен израз в инфиксен запис с операнди - цели числа без знак и операции: +, -, *, / в аритметичен израз в обратен полски запис.

Задача: Проектът **CDataStructures2** включва реализация на *родов стек* и неговото използване за решаване на някои от горните задачи:



Интерфейсът **genericstack.h** представя операции на *родов стек*:

```

/*
 * genericstack.h
 *
 * Created on: 05.2011
 *      Generic Stack Interface
 */

#ifndef GENERICSTACK_H_
#define GENERICSTACK_H_

typedef struct stack *Stack;
/* A Stack object is a last-in-first-out collection of items */

Stack StackNew(void);
/* Returns a new Stack object, or NULL if insufficient memory is
available */

```

```

void StackFree(Stack s);
/* Frees Stack object s */

int StackIsEmpty(Stack s);
/* Returns 1 (TRUE) if s is empty, or 0 (FALSE) otherwise */

int StackPush(Stack s, void *item);
/* Inserts item onto the top of s. Returns 1 (TRUE) if successful,
   or 0 (FALSE) otherwise */

void *StackTop(Stack s);
/* Returns item at the top of s, or NULL if s is empty */

void *StackPop(Stack s);
/* Removes and returns item at the top of s, or NULL if s is empty */

int StackSize(Stack s);
/* Returns the number of items in s */

#endif /* GENERICSTACK_H_ */

```

Да се реализират:

1. Родов последователен стек, с ограничение за броя на елементите, съгласно дадения интерфейс:

```

/*
 * arraystack.h
 *
 * Created on: 05.2011
 *      Array Generic Stack Interface
 */

#ifndef ARRAYSTACK_H_
#define ARRAYSTACK_H_

#include "genericstack.h"

int StackIsFull(Stack s);
/* Return 1 (TRUE) if s is full, or 0 (FALSE) otherwise */

int StackCapacity(Stack s);
/* Return the capacity of s */

#endif /* ARRAYSTACK_H_ */

```

Реализация:

```

/*
 * arraystack.c
 *
 * Created on: 05.2011
 *      Array Generic Stack Implementation
 */

#include <stdlib.h>
#include "arraystack.h"

#define MAX_SIZE 100

struct stack {
    void *items[MAX_SIZE];
    int top;
};

```

```

Stack StackNew(void) {
    Stack s = malloc(sizeof(struct stack));
    if (s == NULL)
        return NULL;
    s->top = -1;
    return s;
}

void StackFree(Stack s) {
    free(s);
}

int StackIsEmpty(Stack s) {
    return s->top == -1;
}

int StackIsFull(Stack s) {
    return s->top == MAX_SIZE - 1;
}

int StackPush(Stack s, void *item) {
    if (StackIsFull(s))
        return 0;
    s->top = s->top + 1;
    int index = s->top;
    s->items[index] = item;
    return 1;
}

void *StackTop(Stack s) {
    if (StackIsEmpty(s))
        return NULL;
    int index = s->top;
    void *item = s->items[index];
    return item;
}

void *StackPop(Stack s) {
    if (StackIsEmpty(s))
        return NULL;
    void *item = StackTop(s);
    s->top = s->top - 1;
    return item;
}

int StackSize(Stack s) {
    return s->top + 1;
}

int StackCapacity(Stack s) {
    return MAX_SIZE;
}

```

2. Функции, съгласно дадения интерфейс:

```

/*
 * stackappl.h
 *
 * Created on: 05.2011
 *
 */

#ifndef STACKAPPL_H_
#define STACKAPPL_H_

void IntegersPrinter();

```

```

void TextWriter();
void PostfixExpressionEvaluator();
void PalindromChecker();
void AkermanEvaluator();

#endif /* STACKAPPL_H_ */

```

Реализация:

```

/*
 * stackappl.c
 *
 * Created on: 05.2011
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "error.h"
#include "arraystack.h"

//Integers Printer
void IntegersPrinter() {
    printf("To stop testing enter: 0\n");
    printf("Starting...\n");
    Stack integerStack = StackNew();
    int n,*p;
    do {
        printf("      n = ");
        fflush(stdout);
        scanf("%d", &n);
        if (n == 0)
            break;
        p = malloc(sizeof(int));
        *p = n;
        StackPush(integerStack, p);
    } while (1);

    while (!StackIsEmpty(integerStack)) {
        p = StackPop(integerStack);
        printf("pop: %d\n", *p);
        free(p);
    }

    printf("Done!\n");
    StackFree(integerStack);
}

//Text Writer
void TextWriter() {
    Stack stringStack = StackNew();
    char inFile[30];
    printf("      Enter input file name: ");
    fflush(stdout);
    scanf("%s", inFile);

    FILE *fp1;
    if ((fp1 = fopen(inFile, "r")) == NULL)
        error("cannot open file\n");

    char outFile[30];
    printf("      Enter output file name: ");
    fflush(stdout);
    scanf("%s", outFile);
}

```

```

FILE *fp2;
if ((fp2 = fopen(outFile, "w")) == NULL)
    error("cannot open file\n");

printf("      Enter an integer n > 0 and n <= %d: ", StackCapacity(
    stringStack));
fflush(stdout);
int n;
scanf("%d", &n);

printf(
    "Reads from %s and puts last %d (or less) lines into %s in reverse
order:\n",
    inFile, n, outFile);
printf("Reading...\n");
fflush(stdout);

char *str;
do {
    str = malloc(81);
    fgets(str, 80, fp1);
    if (feof(fp1))
        break;
    StackPush(stringStack, str);
} while (1);

printf("Writing...\n");
fflush(stdout);
int i;
for (i = 1; i <= n; i++) {
    if (!StackIsEmpty(stringStack)) {
        str = StackPop(stringStack);
        fputs(str, fp2);
        free(str);
    } else
        break;
}

printf("Done!\n");
StackFree(stringStack);
fclose(fp1);
fclose(fp2);
}

//Postfix Expression Evaluater
static char operators[] = "+-*/";
static int isOperator(char arg) {
    int i;
    for (i = 0; i < strlen(operators); i++)
        if (operators[i] == arg)
            return 1;
    return 0;
}

static int evaluate(char *str, int *value) {
    Stack integerStack = StackNew();
    int i, *p, c;
    for (i = 0; i < strlen(str) - 1; i++) {
        c = str[i];
        int number = -1;
        if (c >= '0' && c <= '9') {
            number = 0;
            while (c != ' ') {
                number = number * 10 + (c - '0');
                i++;
                c = str[i];
            }
        }
    }
}

```

```

p = malloc(sizeof(int));
*p = number;
StackPush(integerStack, p);
printf("Pushed constant %d\n", number);
} else if (isOperator(c)) {
    int x, y, answer = 0;
    if (!StackIsEmpty(integerStack)) {
        p = StackPop(integerStack);
        y = *p;
        printf("Popped the right operand %d of the %c\n", y, c);
    } else {
        printf("ERROR: not enough numbers in expression\n");
        return 1;
    }
    if (!StackIsEmpty(integerStack)) {
        p = StackPop(integerStack);
        x = *p;
        printf("Popped the left operand %d of the %c\n", x, c);
    } else {
        printf("ERROR: not enough numbers in expression\n");
        return 1;
    }

    switch (c) {
    case '+':
        answer = x + y;
        break;
    case '-':
        answer = x - y;
        break;
    case '*':
        answer = x * y;
        break;
    case '/':
        if (y)
            answer = x / y;
        else {
            printf("ERROR: devide by zero\n");
            return 1;
        }
        break;
    }
    p = malloc(sizeof(int));
    *p = answer;
    StackPush(integerStack, p);
    printf("Evaluated %c and pushed %d\n", c, answer);
} else {
    printf("ERROR: invalid operator found: %c\n", c);
    return 1;
}
}

if (StackIsEmpty(integerStack)) {
    printf("ERROR: no expression provided\n");
    return 1;
}

p = StackPop(integerStack);
*value = *p;
printf("Popped %d at end of expression\n", *value);
if (!StackIsEmpty(integerStack)) {
    printf("ERROR: not enough operators for all the numbers\n");
    return 1;
}

StackFree(integerStack);

```

```

    return 0;
}

void PostfixExpressionEvaluator() {
    FILE *fp;
    if ((fp = fopen("expressions.txt", "r")) == NULL)
        error("cannot open file\n");

    printf("Starting...\n");
    char *str = malloc(100);
    int value;
    do {
        fgets(str, 99, fp);
        if (feof(fp)) {
            printf("\nEnd of test!\n");
            break;
        }
        printf("\nPostfix Expression: %s\n", str);
        if (!evaluate(str, &value)) {
            printf("Value = %d\n\n", value);
        }
    } while (1);

    free(str);
    fclose(fp);
}

// Palindrom Checker
static int isPalindrom(char *str) {
    Stack integerStack = StackNew();
    int i, *p, c, s, step = 1, flag = 1;
    for (i = 0; i < strlen(str); i++) {
        c = str[i];
        if (step == 1) {
            if (c == 'a' || c == 'b') {
                p = malloc(sizeof(int));
                *p = c;
                StackPush(integerStack, p);
            } else if (c == '@')
                step = 2;
            else {
                printf("ERROR: %c at %d\n", c, i + 1);
                flag = 0;
                break;
            }
        } else { // step = 2
            if (StackIsEmpty(integerStack)) {
                printf("ERROR: missing symbols before @\n");
                flag = 0;
                break;
            } else if (c != 'a' && c != 'b') {
                printf("ERROR: %c at %d\n", c, i + 1);
                flag = 0;
                break;
            } else {
                p = StackPop(integerStack);
                s = *p;
                if (c != s) {
                    printf("ERROR: %c at %d\n", c, i + 1);
                    flag = 0;
                    break;
                }
            }
        }
    }
}

```

```

if (flag && step == 1) {
    printf("ERROR: missing symbol @\n");
    flag = 0;
}

if (flag)
    if (!StackIsEmpty(integerStack))
        printf("ERROR: missing symbols after @\n");

StackFree(integerStack);
return flag;
}

void PalindromChecker() {
    printf("Not implemented!\n");
}

//Akerman Function Evaluater
static int evaluate2(int m, int n) {
    Stack integerStack = StackNew();
    int *p = malloc(sizeof(int));
    *p = m;
    StackPush(integerStack, p);
    p = malloc(sizeof(int));
    *p = n;
    StackPush(integerStack, p);

    while (1) {
        p = StackPop(integerStack);
        n = *p;
        if (StackIsEmpty(integerStack))
            return n;
        p = StackPop(integerStack);
        m = *p;
        if (StackCapacity(integerStack) - StackSize(integerStack) >= 3) {
            if (m != 0) {
                if (n != 0) {
                    p = malloc(sizeof(int));
                    *p = m - 1;
                    StackPush(integerStack, p);
                    p = malloc(sizeof(int));
                    *p = m;
                    StackPush(integerStack, p);
                    p = malloc(sizeof(int));
                    *p = n - 1;
                    StackPush(integerStack, p);
                } else {
                    p = malloc(sizeof(int));
                    *p = m - 1;
                    StackPush(integerStack, p);
                    p = malloc(sizeof(int));
                    *p = 1;
                    StackPush(integerStack, p);
                }
            } else {
                p = malloc(sizeof(int));
                *p = n + 1;
                StackPush(integerStack, p);
            }
        } else {
            printf("ERROR: Impossible computing the value\n");
            return -1;
        }
    }

    StackFree(integerStack);
}

```

```

}

void AkermanEvaluator() {
    printf("Not implemented!\n");
}

```

3. Програма CDataStructures2.c за тестване:

Реализация:

```

/*
=====
Name      : CDataStructures2.c
Version   : 05.2011
Description : Stack Applications
=====

*/
#include <stdio.h>
#include <stdlib.h>
#include "stackappl.h"

#define MAX_ITEMS 6

typedef void (*menuFunction)();
menuFunction command[MAX_ITEMS - 1];

char *item[MAX_ITEMS];

void assignItems();
void assignFunctions();

int main(void) {
    int choice = 1;
    assignItems();
    assignFunctions();
    while (choice) {
        printf("%s", item[0]);
        printf("%s", item[1]);
        printf("%s", item[2]);
        printf("%s", item[3]);
        printf("%s", item[4]);
        printf("%s", item[5]);
        fflush(stdout);
        scanf("%d", &choice);
        if (choice >= 1 && choice <= MAX_ITEMS - 1)
            command[choice - 1]();
        else
            break;
    }

    return EXIT_SUCCESS;
}

void assignItems() {
    item[0] = " 1. Integers Printer\n";
    item[1] = " 2. Text Writer\n";
    item[2] = " 3. Postfix Expression Evaluater\n";
    item[3] = " 4. Palindrom Checker\n";
    item[4] = " 5. Akerman Evaluater\n";
    item[5] = " Enter your choice: ";
}

void assignFunctions() {
    command[0] = IntegersPrinter;

```

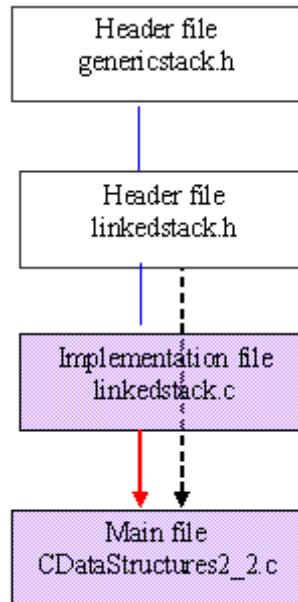
```

command[1] = TextWriter;
command[2] = PostfixExpressionEvaluator;
command[3] = PalindromChecker;
command[4] = AkermanEvaluator;
}

```

Задача: Да се реализира родов свързан стек.

Проект **CDataStructures2_2:**



Интерфейс и реализация на родов свързан стек:

```

/*
 * linkedstack.h
 *
 * Created on: 05.2011
 *      Linked Generic Stack Interface
 */

#ifndef LINKEDSTACK_H_
#define LINKEDSTACK_H_

#include "genericstack.h"

#endif /* LINKEDSTACK_H_ */

/*
 * linkedstack.c
 *
 * Created on: 05.2011
 *      Linked Generic Stack Implementation
 */

#include <stdlib.h>
#include "linkedstack.h"

/* Each item is stored in a stackNode. StackNodes are linked to
form a list */
struct stackNode {
    void *item;
    /* The item */
    struct stackNode *nextNode;
    /* The address of the next stackNode */
};


```

```

typedef struct stackNode *Node;

/* A stack is a structure that points to the first stackNode */
struct stack {
    Node firstNode;
/* The address of the first StackNode */
};

Stack StackNew(void) {
    Stack obj = malloc(sizeof(struct stack));
    if (obj == NULL)
        return NULL;
    obj->firstNode = NULL;
    return obj;
}

void StackFree(Stack s) {
    Node currentNode = s->firstNode;
    Node nextNode;
    for (currentNode = s->firstNode; currentNode != NULL; currentNode
          = nextNode) {
        nextNode = currentNode->nextNode;
        free(currentNode);
    }
    free(s);
}

int StackIsEmpty(Stack s) {
    return s->firstNode == NULL;
}

int StackPush(Stack s, void *item) {
    Node newNode;
    newNode = malloc(sizeof(struct stackNode));
    if (newNode == NULL)
        return 0;
    newNode->item = item;
    newNode->nextNode = s->firstNode;
    s->firstNode = newNode;
    return 1;
}

void *StackTop(Stack s) {
    if (StackIsEmpty(s))
        return NULL;
    return s->firstNode->item;
}

void *StackPop(Stack s) {
    void *result = StackTop(s);
    if (result != NULL) {
        Node nextNode;
        nextNode = s->firstNode->nextNode;
        free(s->firstNode);
        s->firstNode = nextNode;
    }
    return result;
}

int StackSize(Stack s) {
    int count = 0;
    Node temp = s->firstNode;
    while (temp != NULL) {
        count++;
        temp = temp->nextNode;
    }
}

```

```
    return count;
}
```

Задача: Да се реализира родова опашка, съгласно дадения интерфейс:

```
/*
 * genericqueue.h
 *
 * Created on: 05.2011
 *      Generic Queue Interface
 */

#ifndef GENERICQUEUE_H_
#define GENERICQUEUE_H_

typedef struct queue *Queue;
/* A Queue object is a first-in-first-out collection of items */

Queue QueueNew(void);
/* Returns a new Queue object, or NULL if insufficient memory is
available */

void QueueFree(Queue s);
/* Frees Queue object s */

int QueueIsEmpty(Queue s);
/* Returns 1 (TRUE) if s is empty, or 0 (FALSE) otherwise */

int QueueAdd(Queue s, void *item);
/* Inserts item at the tail of s. Returns 1 (TRUE) if successful,
or 0 (FALSE) otherwise */

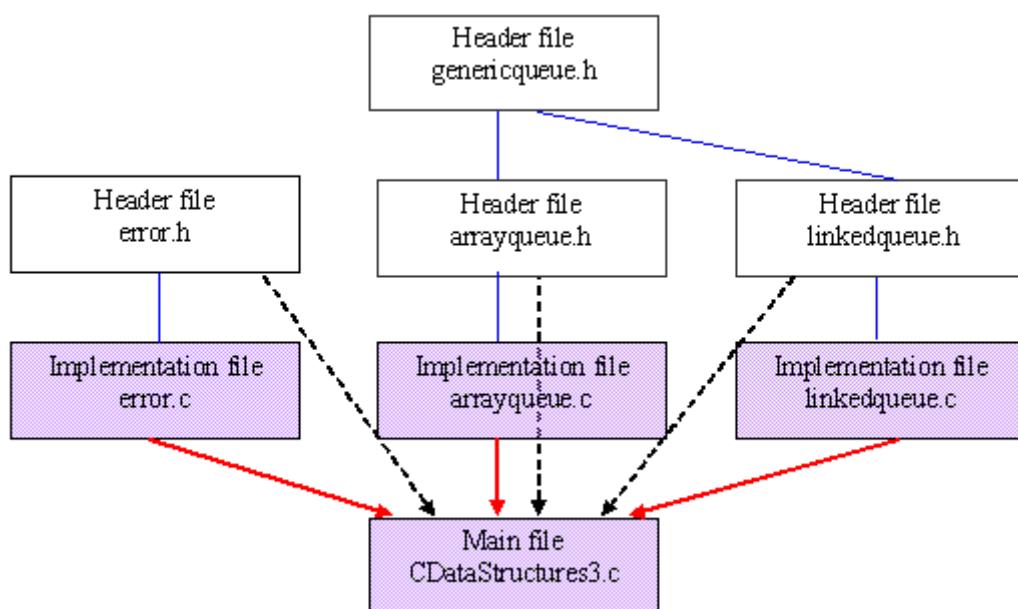
void *QueueElement(Queue s);
/* Returns the head item of s, or NULL if s is empty */

void *QueueRemove(Queue s);
/* Removes and returns the head item from s, or NULL if s is empty */

int QueueSize(Queue s);
/* Returns the number of items in s */

#endif /* GENERICQUEUE_H_ */
```

Проект **CDataStructures3**:



Задача: Да се реализира функция **QueueIntegersPrinter**, която чете последователност от цели числа и извежда първо отрицателните числа, а след това положителните числа в реда на въвеждането им.

Задача: Да се реализира функция **QueueWordsPrinter**, която чете последователност от думи и ги извежда в азбучен ред на първите им букви. Думите, започващи с една и съща буква, се извеждат в реда на въвеждането им.

Задача: Да се реализира *родов дек*, съгласно дадения интерфейс:

```
/*
 * genericdeque.h
 *
 * Created on: 05.2011
 *      Generic Deque Interface
 */

#ifndef GENERICDEQUE_H_
#define GENERICDEQUE_H_

typedef struct deque *Deque;
/* A Deque (deck) object is a double-ended queue collection of items */

Deque DequeNew(void);
/* Returns a new Deque object, or NULL if insufficient memory is
available */

void DequeFree(Deque s);
/* Frees Deque object s */

int DequeIsEmpty(Deque s);
/* Returns 1 (TRUE) if s is empty, or 0 (FALSE) otherwise */

int DequeAddFront(Deque s, void *item);
/* Inserts item at the beginning of s. Returns 1 (TRUE) if successful,
or 0 (FALSE) otherwise */

int DequeAddRear(Deque s, void *item);
/* Inserts item to the end of s. Returns 1 (TRUE) if successful,
or 0 (FALSE) otherwise */

void *DequeGetFirst(Deque s);
/* Returns the first item of s */

void *DequeGetLast(Deque s);
/* Returns the last item of s */

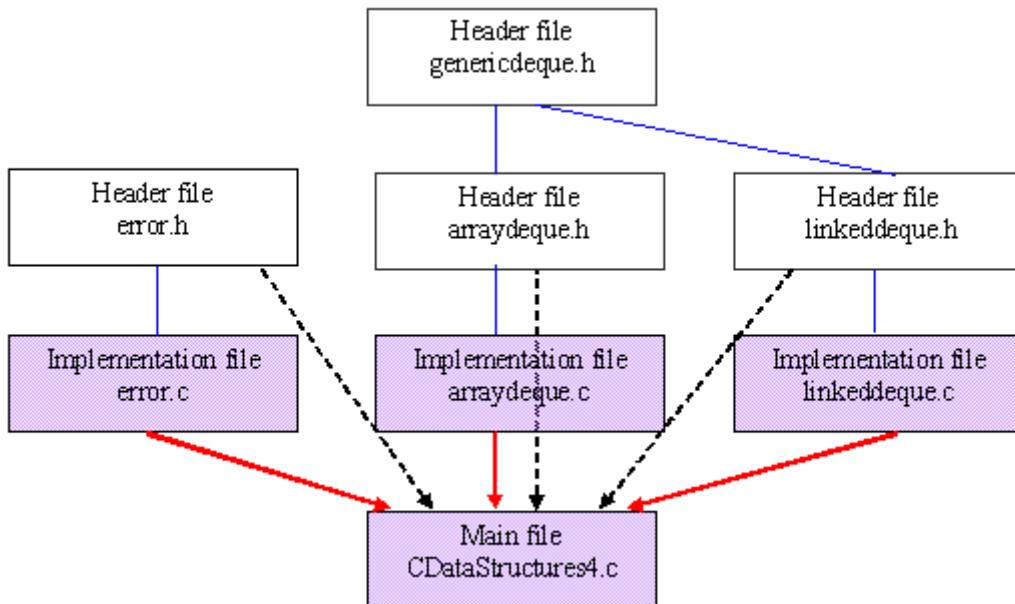
void *DequeRemoveFront(Deque s);
/* Removes and returns the first item of s */

void *DequeRemoveRear(Deque s);
/* Removes and returns the last item of s */

int DequeSize(Deque s);
/* Returns the number of items in s */

#endif /* GENERICDEQUE_H_ */
```

Проект **CDataStructures4**:



Задача: Като се използва дек, да се реализира функция за проверка дали низ е палиндром.

Задача: Да се реализира *родов списък*, съгласно дадения интерфейс:

```

/*
 * genericlist.h
 *
 * Created on: 05.2011
 *      Generic List Interface
 */

#ifndef GENERICLIST_H_
#define GENERICLIST_H_

typedef int (*equals)(const void *, const void *);
/* An equivalence relation */

typedef struct list *List;
/* A List object is a collection of items */

List ListNew(void);
/* Returns a new List object, or NULL if insufficient memory is
available */

void ListFree(List s);
/* Frees List object s */

int ListIsEmpty(List s);
/* Returns 1 (TRUE) if s is empty, or 0 (FALSE) otherwise */

int ListAdd(List s, int pos, void *item);
/* Inserts item at the specified position pos of s. Returns 1 (TRUE) if
successful,
or 0 (FALSE) otherwise */

int ListAddFirst(List s, void *item);
/* Inserts item at the beginning of s. Returns 1 (TRUE) if successful,
or 0 (FALSE) otherwise */

int ListAddLast(List s, void *item);
/* Inserts item to the end of s. Returns 1 (TRUE) if successful,
or 0 (FALSE) otherwise */
  
```

```

void *ListGet(List s, int pos);
/* Returns the item at the specified position pos in s if possible,
or NULL otherwise */

void *ListGetFirst(List s);
/* Returns the first item of s, or NULL if s is empty */

void *ListGetLast(List s);
/* Returns the last item of s, or NULL if s is empty */

void *ListRemove(List s, int pos);
/* Removes and returns the item at the specified position pos in s
if possible, or NULL otherwise */

void *ListRemoveFirst(List s);
/* Removes and returns the first item of s, or NULL if s is empty */

void *ListRemoveLast(List s);
/* Removes and returns the last item of s, or NULL if s is empty */

int ListSize(List s);
/* Returns the number of items in s */

void *ListSet(List s, int pos, void *item);
/* Replaces the item at the specified position pos in s with the specified item
if possible. Returns the old item, or NULL otherwise */

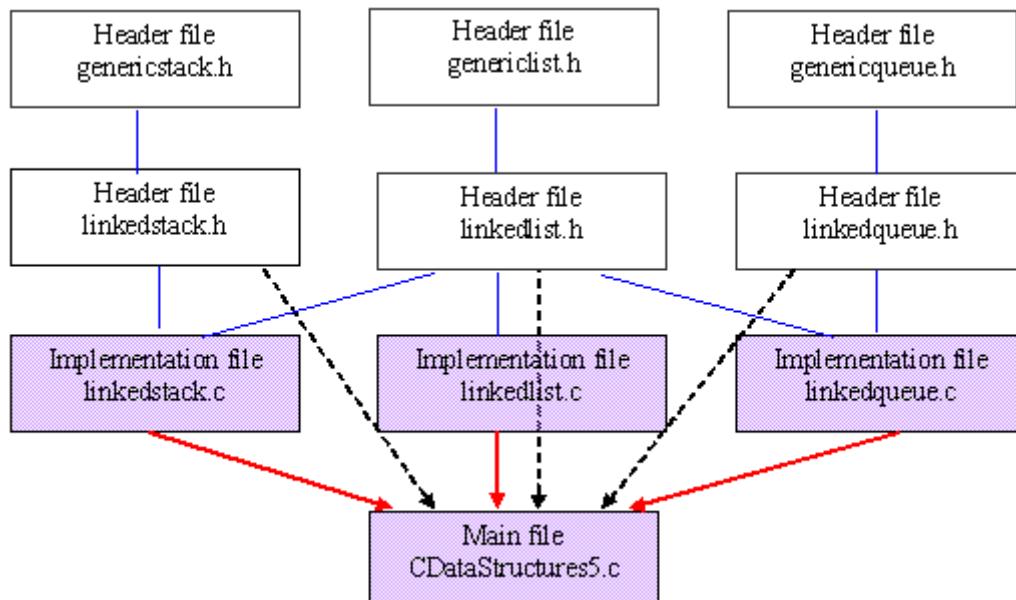
int ListContains(List s, void *item, equals eq);
/* Returns 1 (TRUE) if s contains item, or 0 (FALSE) otherwise */

List ListSubList(List s, int fromIndex, int toIndex);
/* Returns a view of the portion of list s between the specified fromIndex,
inclusive, and toIndex, exclusive */

#endif /* GENERICLIST_H_ */

```

Проектът **CDataStructures5** съдържа реализация на свързан списък и илюстрация за неговото използване - реализация на свързан стек и свързана опашка:



Интерфейс и реализация на свързан списък:

```

/*
 * linkedlist.h

```

```

/*
 * Created on: 05.2011
 *      Linked Generic List Interface
 */

#ifndef LINKEDLIST_H_
#define LINKEDLIST_H_

#include "genericlist.h"

#endif /* LINKEDLIST_H_ */

/*
 * linkedlist.c
 *
 * Created on: 05.2011
 *      Linked Generic List Implementation
 */

#include <stdio.h>
#include <stdlib.h>
#include "linkedlist.h"

/* Each item is stored in a listNode. List nodes are linked to
form a list */
struct listNode {
    void *item;
    /* The item */
    struct listNode *nextNode;
    /* The address of the next listNode */
};

typedef struct listNode *Node;

/* A list is a structure that points to the first listNode */
struct list {
    Node head;
    /* The address of the first listNode */
};

List ListNew(void) {
    List obj = malloc(sizeof(struct list));
    if (obj == NULL)
        return NULL;
    obj->head = NULL;
    return obj;
}

void ListFree(List s) {
    Node currentNode = s->head;
    Node nextNode;
    while (currentNode != NULL) {
        nextNode = currentNode->nextNode;
        free(currentNode);
        currentNode = nextNode;
    }
    free(s);
}

int ListIsEmpty(List s) {
    return s->head == NULL;
}

int ListAdd(List s, int pos, void *item) {
    if (pos < 0 || pos > ListSize(s))
        return 0;
    if (pos == 0)

```

```

    return ListAddFirst(s, item);
else {
    int i;
    Node currentNode = s->head;
    for (i = 0; i < pos - 1; i++)
        currentNode = currentNode->nextNode;
    Node newNode = malloc(sizeof(struct listNode));
    if (newNode == NULL)
        return 0;
    newNode->item = item;
    newNode->nextNode = NULL;
    currentNode->nextNode = newNode;
    return 1;
}
}

int ListAddFirst(List s, void *item) {
    Node newNode = malloc(sizeof(struct listNode));
    if (newNode == NULL)
        return 0;
    newNode->item = item;
    newNode->nextNode = s->head;
    s->head = newNode;
    return 1;
}

int ListAddLast(List s, void *item) {
    if (ListIsEmpty(s))
        return ListAddFirst(s, item);
    else {
        Node currentNode = s->head;
        while (currentNode->nextNode != NULL)
            currentNode = currentNode->nextNode;
        Node newNode = malloc(sizeof(struct listNode));
        if (newNode == NULL)
            return 0;
        newNode->item = item;
        newNode->nextNode = currentNode->nextNode;
        currentNode->nextNode = newNode;
        return 1;
    }
}

void *ListGet(List s, int pos) {
    if (pos < 0 || pos >= ListSize(s))
        return NULL;
    if (pos == 0) {
        return ListGetFirst(s);
    } else {
        int i;
        Node currentNode = s->head;
        for (i = 0; i < pos; i++)
            currentNode = currentNode->nextNode;
        return currentNode->item;
    }
}

void *ListGetFirst(List s) {
    if (ListIsEmpty(s))
        return NULL;
    return s->head->item;
}

void *ListGetLast(List s) {
    printf("Not implemented!\n");
    return NULL;
}

```

```

}

void *ListRemove(List s, int pos) {
    printf("Not implemented!\n");
    return NULL;
}

void *ListRemoveFirst(List s) {
    void *result = ListGetFirst(s);
    Node nextNode;
    if (result != NULL) {
        nextNode = s->head->nextNode;
        free(s->head);
        s->head = nextNode;
    }
    return result;
}

void *ListRemoveLast(List s) {
    printf("Not implemented!\n");
    return NULL;
}

int ListSize(List s) {
    int count = 0;
    Node currentNode = s->head;
    while (currentNode != NULL) {
        count++;
        currentNode = currentNode->nextNode;
    }
    return count;
}

void *ListSet(List s, int pos, void *item) {
    printf("Not implemented!\n");
    return NULL;
}

int ListContains(List s, void *item, equals eq) {
    Node currentNode = s->head;
    while (currentNode != NULL) {
        if (eq(currentNode->item, item))
            return 1;
        currentNode = currentNode->nextNode;
    }
    return 0;
}

List ListSubList(List s, int fromIndex, int toIndex) {
    List result = ListNew();
    int i;
    for (i = fromIndex; i < toIndex; i++)
        ListAddLast(result, ListGet(s, i));
    return result;
}

```

Интерфейс и реализация на родова свързана опашка с използване на родов свързан списък:

```

/*
 * linkedqueue.h
 *
 * Created on: 05.2011
 *      Linked Generic Queue Interface
 */

#ifndef LINKEDQUEUE_H_

```

```

#define LINKEDQUEUE_H_
#include "genericqueue.h"
#endif /* LINKEDQUEUE_H_ */

/*
 * linkedqueue.c
 *
 * Created on: 16.05.2011
 *      Linked Generic Queue Implementation
 */

#include <stdlib.h>
#include "linkedqueue.h"
#include "linkedlist.h"

/* A queue is a FIFO list structure */
struct queue {
    List list;
};

Queue QueueNew(void) {
    Queue obj = malloc(sizeof(struct queue));
    obj->list = ListNew();
    return obj;
}

void QueueFree(Queue s) {
    ListFree(s->list);
}

int QueueIsEmpty(Queue s) {
    return ListIsEmpty(s->list);
}

int QueueAdd(Queue s, void *item) {
    return ListAddLast(s->list, item);
}

void *QueueElement(Queue s) {
    return ListGetFirst(s->list);
}

void *QueueRemove(Queue s) {
    return ListRemoveFirst(s->list);
}

int QueueSize(Queue s) {
    return ListSize(s->list);
}

```

Реализация на програма за тестване:

```

/*
=====
Name      : CDataStructures5.c
Version   : 05.2011
Description : Test Linked Generic List, Linked Generic Stack,
              Linked Generic Queue
=====
*/
#include <stdio.h>
#include <stdlib.h>

```

```

#include "linkedlist.h"
#include "linkedstack.h"
#include "linkedqueue.h"

void ListIntegersPrinter();
static int IntegerEq(const void *arg1, const void *arg2);
void StackIntegersPrinter();
void QueueIntegersPrinter();

int main(void) {
    ListIntegersPrinter();
    StackIntegersPrinter();
    QueueIntegersPrinter();

    return EXIT_SUCCESS;
}

//Integers Printer
void ListIntegersPrinter() {
    printf("To stop testing enter: 0\n");
    printf("List Integers Printer Starting...\n");
    List integerList = ListNew();
    int *p, k = 0;
    do {
        printf("      n = ");
        fflush(stdout);
        p = malloc(sizeof(int));
        scanf("%d", p);
        if (*p == 0)
            break;
        ListAdd(integerList, k++, p);
    } while (1);

    int *q = malloc(sizeof(int));
    *q = 10;
    printf("%d\n", ListContains(integerList, q, IntegerEq));
    fflush(stdout);
    integerList = ListSubList(integerList, 1, 3);
    int i;
    for (i = 0; i < ListSize(integerList); i++) {
        p = ListGet(integerList, i);
        printf("remove: %d\n", *p);
        free(p);
    }

    printf("Done!\n");
    ListFree(integerList);
}

static int IntegerEq(const void *arg1, const void *arg2) {
    int *r1 = (int *) arg1;
    int *r2 = (int *) arg2;
    int left = (*r1);
    int right = (*r2);
    return left == right;
}

void StackIntegersPrinter() {
    printf("To stop testing enter: 0\n");
    printf("Stack Integers Printer Starting...\n");
    Stack integerStack = StackNew();
    int n, *p;
    do {
        printf("      n = ");
        fflush(stdout);
        scanf("%d", &n);
    }
}

```

```

if (n == 0)
    break;
p = malloc(sizeof(int));
*p = n;
StackPush(integerStack, p);
} while (1);

while (!StackIsEmpty(integerStack)) {
    p = StackPop(integerStack);
    printf("pop: %d\n", *p);
    free(p);
}

printf("Done!\n");
StackFree(integerStack);
}

void QueueIntegersPrinter() {
    printf("To stop testing enter: 0\n");
    printf("Queue Integers Printer Starting...\n");
    Queue integerQueue1 = QueueNew();
    Queue integerQueue2 = QueueNew();
    int *p;
    do {
        printf("      n = ");
        fflush(stdout);
        p = malloc(sizeof(int));
        scanf("%d", p);
        if (*p == 0)
            break;
        if (*p < 0)
            QueueAdd(integerQueue1, p);
        else
            QueueAdd(integerQueue2, p);
    } while (1);

    printf("Negative Integers:\n");
    while (!QueueIsEmpty(integerQueue1)) {
        p = QueueRemove(integerQueue1);
        printf("remove: %d\n", *p);
        free(p);
    }

    printf("Positive Integers:\n");
    while (!QueueIsEmpty(integerQueue2)) {
        p = QueueRemove(integerQueue2);
        printf("remove: %d\n", *p);
        free(p);
    }

    printf("Done!\n");
    QueueFree(integerQueue1);
    QueueFree(integerQueue2);
}

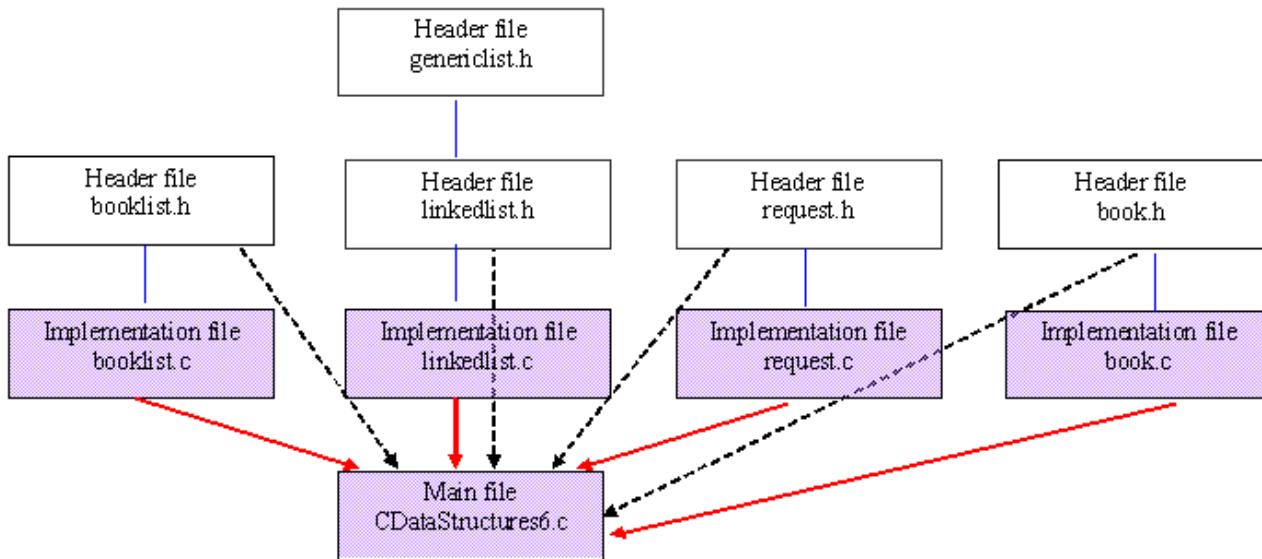
```

Задача: Да се състави програма за създаване и поддържане на *колекция от книги в книжарница*, като всяка книга има каталожен номер, наименование, автор, цена и налично количество екземпляри и колекцията е наредена във възходящ ред на каталожните номера. Програмата да предоставя следните обработки:

1. Първоначално създаване на празна колекция
2. Обработка на доставка на книга. Ако книгата я няма в колекцията, се включва в нея. В противен случай към наличното количество се добавя новополученото

3. Обработка на продажба на книга. Ако книгата я няма в колекцията, се извежда подходящи съобщение и като резултата се получава -1. В противен случай:
 - Ако исканото количество е по-малко от наличното, наличното количество се намалява с исканото и като резултата се получава исканото количество
 - Ако исканото количество е по-голямо или равно на наличното, книгата се изключва от колекцията и като резултата се получава наличното количество
4. Обработка на заявка за доставка или продажба по зададен списък от книги и вида на заявката. Като резултата се получава общ брой доставени или продадени книги
5. Обработка за получаване на наличните книги, написани от даден автор
6. Справка за наличните книги в колекцията

Проект **CDataStructures6:**



Интерфейс и реализация на книга:

```

/*
 * book.h
 *
 * Created on: 05.2011
 *      Book Interface
 */

#ifndef BOOK_H_
#define BOOK_H_

struct book {
    // Data
    int bookID;
    char *title, *author;
    int quantity;
};
typedef struct book *Book;

Book BookNew(int id, char *t, char *a, int q);
int BookCompareTo(Book b1, Book b2);
void BookPrint(Book b);

#endif /* BOOK_H_ */

/*
 * book.c
 *

```

```

/*
 * Created on: 05.2011
 *      Book Implementation
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "book.h"

Book BookNew(int id, char *t, char *a, int q) {
    Book obj = malloc(sizeof(struct book));
    obj->bookID = id;
    obj->title = malloc(strlen(t) + 1);
    strcpy(obj->title, t);
    obj->author = malloc(strlen(a) + 1);
    strcpy(obj->author, a);
    obj->quantity = q;
    return obj;
}

int BookCompareTo(Book b1, Book b2) {
    return b1->bookID - b2->bookID;
}

void BookPrint(Book b) {
    printf("\n(%d, %s , %s , %d)\n", b->bookID, b->title, b->author,
           b->quantity);
}

```

Интерфейс и реализация на заявка:

```

/*
 * request.h
 *
 * Created on: 05.2011
 *      Request Interface
 */

#ifndef REQUEST_H_
#define REQUEST_H_

#include "book.h"

#define MAX_SIZE 100
struct request {
    // Data
    int kind; // 1 - supplies, 2 - sales
    Book books[MAX_SIZE];
    int n;
};

typedef struct request *Request;

Request RequestNew(int k, Book b[], int n);
void RequestPrint(Request r);

#endif /* REQUEST_H_ */

/*
 * request.c
 *
 * Created on: 05.2011
 *      Request Implementation
 */

#include <stdio.h>

```

```
#include <stdlib.h>
#include "request.h"

Request RequestNew(int k, Book b[], int n) {
    Request obj = malloc(sizeof(struct request));
    obj->kind = k;
    obj->n = n;
    int i;
    for (i = 0; i < n; i++)
        obj->books[i] = b[i];
    return obj;
}

void RequestPrint(Request r) {
    printf("\n      Request of kind %d\n", r->kind);
    int i;
    for (i = 0; i < r->n; i++)
        BookPrint(r->books[i]);
}
```

Интерфейс и реализация на колекция от книги:

```
/*
 * booklist.h
 *
 * Created on: 05.2011
 *      Book List Interface
 */

#ifndef BOOKLIST_H_
#define BOOKLIST_H_

#include "request.h"
#include "linkedlist.h"

extern List data;

void BookListNew();
int BookListAddBook(Book b);
int BookListSaleBook(Book b);
List BookListGetBooks(char *author);
int BookListDoRequest(Request r);
void show(List list);

#endif /* BOOKLIST_H_ */

/*
 * booklist.c
 *
 * Created on: 05.2011
 *      Book List Implementation
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "booklist.h"

List data;

void BookListNew() {
    data = ListNew();
}

int BookListAddBook(Book b) {
```

```

int flag = 0;
int i = 0;
for (; i < ListSize(data); i++) {
    Book w = ListGet(data, i);
    if (BookCompareTo(b, w) < 0) {
        ListAdd(data, i, b);
        flag = 1;
        break;
    } else if (BookCompareTo(b, w) == 0) {
        flag = 1;
        w->quantity = w->quantity + b->quantity;
        break;
    }
}
if (!flag)
    ListAdd(data, i, b);
return b->quantity;
}

int BookListSaleBook(Book b) {
    int i;
    for (i = 0; i < ListSize(data); i++) {
        Book w = ListGet(data, i);
        if (BookCompareTo(b, w) == 0) {
            int q1 = w->quantity;
            int q2 = b->quantity;
            if (q1 > q2) {
                w->quantity = q1 - q2;
                return q2;
            } else {
                ListRemove(data, i);
                return q1;
            }
        }
    }
    printf("%d %s not found\n", b->bookID, b->title);
    return -1;
}

List BookListGetBooks(char *author) {
    List newList = ListNew();
    int i;
    for (i = 0; i < ListSize(data); i++) {
        Book w = ListGet(data, i);
        if (strcmp(author, w->author) == 0)
            ListAddFirst(newList, w);
    }
    return newList;
}

int BookListDoRequest(Request r) {
    int i, sum = 0, k;
    if (r->kind == 1)
        for (i = 0; i < r->n; i++)
            sum += BookListAddBook(r->books[i]);
    else {
        for (i = 0; i < r->n; i++) {
            k = BookListSaleBook(r->books[i]);
            if (k != -1)
                sum += k;
        }
    }
    return sum;
}

void show(List list) {

```

```
int i;
for (i = 0; i < ListSize(list); i++) {
    Book w = ListGet(list, i);
    BookPrint(w);
}
}
```

Обекти и класове. Предефиниране на операциите. Рационално число

Всички операции на C++

+	-	*	/	=	<	>	+=	--	*=	/=	<<	>>
<=<	>>=	==	!=	<=	>=	++	--	%	&	^	!	
~	&=	^=	=	&&		%=	[]	()	,	->*	->	new
delete	new []			delete []								

могат да се предефинират, с изключение на операциите:

- „..” (точка) - избор на член чрез име на обект или клас
- „..*” (точка, звезда) – указател към член на клас чрез обект
- „...” (двойно двоеточие) – за определяна обхват на идентификатор
- „?:” (въпросителна, двоеточие) – триаргументна условна операция

Задача: Да се реализира клас **Ratio** за представяне на рационално число, съгласно дадения интерфейс:

```
/*
 * Ratio.h
 *
 * Created on: 09.2011
 *
 */

#ifndef RATIO_H_
#define RATIO_H_

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

class Ratio {
    //Input/Output friend functions
    friend istream& operator>>(istream &s, Ratio &num);
    //Input stream overloading
    //Usage:
    //Ratio n1,n2;
    //cin>>n1>>n2;
    //This means (cin>>n1)>>n2;
    //This means operator>>((operator>>(cin,n1),n2)

    friend ostream& operator<<(ostream &s, const Ratio &n);
    //Output stream overloading
    //Usage:
    //Ratio n1,n2;
    //cout<<n1<<n2;
    //This means (cout<<n1)<<n2;
    //This means operator<<((operator<<(cout,n1),n2)

private:
    long num; //numerator
    long den; //denominator
    void simplify(Ratio&);

public:
    //Constructor(s)
    Ratio(long = 0, long = 1);

    //Type conversions
    operator double() const;
    //Conversion from Ratio to double
    //Usage:
    //Ratio n;double r;
    //r=(double)n or double(n) or static_cast<double>(n);
}
```

```

operator long() const;
//Conversion from Ratio to long
//Usage:
//Ratio n;long r;
//r=(long)n or long(n) or static_cast<long>(n);

//Assignment operators = += -= *= ==
Ratio& operator=(const Ratio &n);
//Assignment operator
//Usage:
//Ratio n1,n2,n3;
//n1=n2=n3;
//This means n1=(n2=n3);
//This means n1.operator=((n2.operator=(n3)))

Ratio& operator=(long whole_n);
//Assignment operator
//Usage:
//Ratio n1=n2;
//n1=n2=1234;
//This means n1.operator=((n2.operator=(1234))

Ratio& operator+=(const Ratio &n);
//Operator addition
//Usage:
//Ratio n1,n2;
//n2+=n1;

Ratio& operator-=(const Ratio &n);
//Operator subtraction
//Usage:
//Ratio n1,n2;
//n2-=n1;

Ratio& operator*=(const Ratio &n);
//Operator multiplication
//Usage:
//Ratio n1,n2;
//n2*=n1;

Ratio& operator/=(const Ratio &n);
//Operator division
//Usage:
//Ratio n1,n2;
//n2/=n1;
//If the denominator of n is zero no operation is performed

//Arithmetic operators + - * /
Ratio operator+(const Ratio &n) const;
//Operator addition
//Usage:
//Ratio n1,n2,n3;
//n3=n1+n2;

Ratio operator-(const Ratio &n) const;
//Operator subtraction
//Usage:
//Ratio n1,n2,n3;
//n3=n1-n2;

Ratio operator*(const Ratio &n) const;
//Operator multiplication
//Usage:
//Ratio n1,n2,n3;
//n3=n1*n2;

```

```

Ratio operator/(const Ratio &n) const;
//Operator division
//Usage:
//Ratio n1,n2,n3;
//n3=n1/n2;
//If the denominator of n is zero no operation is performed

//Relational operators == != < <= > >=
bool operator==(const Ratio &n) const;
//Operator equality
//Usage:
//Ratio n1,n2;
//if (n1==n2)...

bool operator!=(const Ratio &n) const;
//Operator inequality
//Usage:
//Ratio n1,n2;
//if (n1!=n2)...

bool operator<(const Ratio &n) const;
//Operator <
//Usage:
//Ratio n1,n2;
//if (n1<n2)...

bool operator<=(const Ratio &n) const;
//Operator <=
//Usage:
//Ratio n1,n2;
//if (n1<=n2)...

bool operator>(const Ratio &n) const;
//Operator >
//Usage:
//Ratio n1,n2;
//if (n1>n2)...

bool operator>=(const Ratio &n) const;
//Operator >=
//Usage:
//Ratio n1,n2;
//if (n1>=n2)...
};

#endif /* RATIO_H_ */

```

Реализация:

```

/*
 * Ratio.cpp
 *
 * Created on: 09.2011
 *
 */

#include <stdlib.h>
#include "Ratio.h"
#include <string>
#include <exception>
#include <iostream>
using namespace std;

void Ratio::simplify(Ratio &n) {
    string msg("RationalNumbers: invalid denominator 0!");

```

```

if (n.den == 0)
    throw msg;
if (n.num == 0)
    n.den = 1;
else {
    long a = abs(n.num);
    long b = abs(n.den);
    long p = a % b;
    while (p != 0) {
        a = b;
        b = p;
        p = a % b;
    }
    n.num /= b;
    n.den /= b;

    //Attach correct sign to the numerator
    if (n.num < 0 && n.den < 0) {
        n.num = -n.num;
        n.den = -n.den;
    } else if (n.num < 0 || n.den < 0) {
        n.num = -abs(n.num);
        n.den = abs(n.den);
    }
}
}

istream& operator>>(istream &s, Ratio &n) {
    long a, b;
    char ch;
    s >> a;
    s >> ch;
    s >> b;
    n = Ratio(a, b);
    return s;
}

ostream& operator<<(ostream &s, const Ratio &n) {
    if (n.den != 1)
        return s << n.num << "/" << n.den;
    else
        return s << n.num;
}

Ratio::Ratio(long n, long d) {
    num = n;
    den = d;
    simplify(*this);
}

Ratio::operator double() const {
    return (double(num) / double(den));
}

Ratio::operator long() const {
    return num / den;
}

Ratio& Ratio::operator=(const Ratio &n) {
    num = n.num;
    den = n.den;
    simplify(*this);
    return *this;
}

Ratio& Ratio::operator=(long whole_n) {

```

```

num = whole_n;
den = 1;
return *this;
}

Ratio& Ratio::operator+=(const Ratio &n) {
    num = num * n.den + den * n.num;
    den *= n.den;
    simplify(*this);
    return *this;
}

Ratio& Ratio::operator==(const Ratio &n) {
    num = num * n.den - den * n.num;
    den *= n.den;
    simplify(*this);
    return *this;
}

Ratio& Ratio::operator*=(const Ratio &n) {
    num *= num * n.num;
    den *= den * n.den;
    simplify(*this);
    return (*this);
}

Ratio& Ratio::operator/=(const Ratio &n) {
    string msg("RationalNumbers: divide by zero!");
    if (n.num != 0) {
        num *= n.den;
        den *= n.num;
        simplify(*this);
        return *this;
    } else
        throw msg;
}

Ratio Ratio::operator+(const Ratio &n) const {
    return Ratio(num * n.den + n.num * den, den * n.den);
}

Ratio Ratio::operator-(const Ratio &n) const {
    return Ratio(num * n.den - n.num * den, den * n.den);
}

Ratio Ratio::operator*(const Ratio &n) const {
    return Ratio(num * n.num, den * n.den);
}

Ratio Ratio::operator/(const Ratio &n) const {
    string msg("RationalNumbers: divide by zero!");
    if (n.num == 0)
        throw msg;
    int a = num * n.den;
    int b = abs(den * n.num);
    return n.num < 0 ? Ratio(-a, b) : Ratio(a, b);
}

bool Ratio::operator==(const Ratio &n) const {
    return num == n.num && den == n.den;
}

bool Ratio::operator!=(const Ratio &n) const {
    return num != n.num || den != n.den;
}

```

```
bool Ratio::operator<=(const Ratio &n) const {
    return num * n.den - n.num * den <= 0;
}
```

На адрес <http://www.cplusplus.com/reference/> може да намерите информация за стандартните библиотеки на C++. Може да се запознаете по-подробно с възможностите за използване на:

- Низове на адрес <http://www.cplusplus.com/reference/string/>
- Изключения на адрес <http://www.cplusplus.com/doc/tutorial/exceptions/>, както и с наличните изключения <http://www.cplusplus.com/reference/std/exception/exception/>

Задача: Да се реализират:

1. Клас **RatioCalculator** за представяне на обект-калкулатор, който чете двойки от рационални числа и за числата от всяка двойка прилага аритметичните операции и операция за сравнение и извежда получените резултати

Интерфейс:

```
/*
 * RatioCalculator.h
 *
 * Created on: 09.2011
 *
 */

#ifndef RATIOCALCULATOR_H_
#define RATIOCALCULATOR_H_


class RatioCalculator {
public:
    void start();
};

#endif /* RATIOCALCULATOR_H_ */
```

Реализация:

```
/*
 * RatioCalculator.cpp
 *
 * Created on: 09.2011
 *
 */

#include "Ratio.h"
#include "RatioCalculator.h"
#include <string>
#include <exception>
#include <iostream>
using namespace std;

void RatioCalculator::start() {
    int flag = 1;
    while (flag == 1) {
        Ratio A, B;
        try {
            cout << "      A = ";
            cin >> A;
            cout << "      B = ";
            cin >> B;
            cout << A << " + " << B << " = " << A + B << endl;
            cout << A << " - " << B << " = " << A - B << endl;
            cout << A << " * " << B << " = " << A * B << endl;
        }
```

```

        cout << A << " / " << B << " = " << A / B << endl;
        cout << A << " <= " << B << " = ";
        cout << (A <= B ? "true" : "false") << endl;
    } catch (string str) {
        cout << "Exception raised: " << str << endl;
    }
    cout << "Do you want to continue?(1-Yes/0-No): ";
    cin >> flag;
}
}

```

2. Клас **RatioBubbleSorter** за сортиране на масив от рационални числа по *метода на мехурчето*

Интерфейс:

```

/*
 * RatioBubbleSorter.h
 *
 * Created on: 09.2011
 *
 */

#ifndef BUBBLESORTER_H_
#define BUBBLESORTER_H_

#include "Ratio.h"

class RatioBubbleSorter {
public:
    void sort(Ratio[], int, int, int);
    void sort(Ratio[], int);
};

#endif /* BUBBLESORTER_H_ */

```

Реализация:

```

/*
 * RatioBubbleSorter.cpp
 *
 * Created on: 09.2011
 *
 */

#include "RatioBubbleSorter.h"
#include <string>
#include <exception>
#include <iostream>
using namespace std;

void RatioBubbleSorter::sort(Ratio array[], int n, int from, int to) {
    string msg("BubbleSorter: invalid arguments!");
    if (from >= 0 && to >= from && to <= n - 1) {
        bool sorted;
        do {
            sorted = true;
            for (int i = from; i <= to; i++) {
                if (array[i] <= array[i + 1]) {
                    Ratio temp = array[i];
                    array[i] = array[i + 1];
                    array[i + 1] = temp;
                    sorted = false;
                }
            }
        }
    }
}
```

```

        } while (!sorted);
    } else
        throw msg;
}

void RatioBubbleSorter::sort(Ratio array[], int n) {
    sort(array, n, 0, n - 1);
}

```

Тестване на класовете Ratio, RatioCalculator и RatioBubbleSorter: Реализация на клас **Tester** за провеждане на тестове, които илюстрират използването на обекти и масиви от обекти, както и обработка на изключения

Интерфейс на класа Tester:

```

/*
 * Tester.h
 *
 * Created on: 10.2011
 *
 */

#ifndef TESTER_H_
#define TESTER_H_

#include "Ratio.h"

class Tester {
private:
    Ratio numbers[100];
    int n;
    void testRatioCalculator();
    void testRatioBubbleSorter();
    void read();
    void print() const;
public:
    void start();
};

#endif /* TESTER_H_ */

```

Реализация на класа Tester:

```

/*
 * Tester.cpp
 *
 * Created on: 10.2011
 *
 */

#include "Tester.h"
#include "RatioCalculator.h"
#include "RatioBubbleSorter.h"
#include <string>
#include <iostream>
using namespace std;

void Tester::start() {
    cout << "      Test RatioCalculator" << endl;
    testRatioCalculator();
    cout << "      Test RatioBubbleSorter" << endl;
    testRatioBubbleSorter();
}

```

```

void Tester::testRatioCalculator() {
    RatioCalculator calculator;
    cout << "Start..." << endl;
    calculator.start();
    cout << "Done!" << endl;
}

void Tester::read() {
    cout << "      Enter n > 0: ";
    cin >> n;
    for (int i = 0; i < n; i++) {
        cout << "Enter a ratio: ";
        cin >> numbers[i];
    }
}

void Tester::print() const{
    for (int i = 0; i < n; i++)
        cout << numbers[i] << endl;
}

void Tester::testRatioBubbleSorter() {
    cout << "Start..." << endl;
    read();
    RatioBubbleSorter().sort(numbers, n);
    cout << "The sorted array is: " << endl;
    print();
    cout << "Done!" << endl;
}

```

Резултати от тестването:

```

//=====
// Name      : RationalNumbers.cpp
// Version   : 09.2011
// Description : Rational numbers tests
//=====

#include "Tester.h"
#include <iostream>
using namespace std;

int main() {
    cout << "      Test Rational numbers" << endl;
    Tester().start();
    return 0;
}

      Test Rational numbers
      Test RatioCalculator
Start...
      A  = 1/2
      B  = 0/2
1/2 + 0 = 1/2
1/2 - 0 = 1/2
1/2 * 0 = 0
Exception raised: RationalNumbers: divide by zero!
Do you want to continue?(1-Yes/0-No): 1
      A  = 1/2
      B  = 1/3
1/2 + 1/3 = 5/6
1/2 - 1/3 = 1/6
1/2 * 1/3 = 1/6
1/2 / 1/3 = 3/2
1/2 <= 1/3 = false

```

```
Do you want to continue?(1-Yes/0-No) : 0
Done!
Test RatioBubbleSorter
Start...
Enter n > 0: 3
Enter a ratio: 1/3
Enter a ratio: 1/2
Enter a ratio: 1/4
The sorted array is:
1/2
1/3
1/4
Done!
```

Задача: Число от вида $N = a + bi + cj + dk$, $a, b, c, d \in R$, се нарича **кватернион** и се представя геометрически в четиримерното пространство – виж <http://en.wikipedia.org/wiki/Quaternion>. Следващата таблица съдържа информация за резултата от произведението за всяка двойка от 1, i, j и k.

x	1	i	j	k
1	1	<i>i</i>	<i>j</i>	<i>k</i>
<i>i</i>	<i>i</i>	-1	<i>k</i>	- <i>j</i>
<i>j</i>	<i>j</i>	- <i>k</i>	-1	<i>i</i>
<i>k</i>	<i>k</i>	<i>j</i>	- <i>i</i>	-1

Числото $\bar{N} = a - bi - cj - dk$ се нарича *спрегнато на* числото N , а $\| N \| = \sqrt{a^2 + b^2 + c^2}$ - *норма на* числото N .

Аритметични операции:

1. Сумата на два кватерниона $a_1 + b_1i + c_1j + d_1k$ и $a_2 + b_2i + c_2j + d_2k$, е кватернионът, който се получава след преобразуване на сумата:

$$(a_1 + b_1i + c_1j + d_1k) + (a_2 + b_2i + c_2j + d_2k)$$

2. Произведението на два кватерниона $a_1 + b_1i + c_1j + d_1k$ и $a_2 + b_2i + c_2j + d_2k$, е кватернионът, който се получава след преобразуване на произведението:

$$(a_1 + b_1i + c_1j + d_1k) * (a_2 + b_2i + c_2j + d_2k)$$

3. Частното на два кватерниона $a_1 + b_1i + c_1j + d_1k$ и $a_2 + b_2i + c_2j + d_2k$, е кватернионът, който се получава след преобразуване на произведението:

$$\overline{(a_1 + b_1i + c_1j + d_1k)} * \overline{(a_2 + b_2i + c_2j + d_2k)}$$

Наредба: $N_1 \leq N_2 \Leftrightarrow \| N_1 \| \leq \| N_2 \|$

Да се състави спецификация на клас **Quaternion** за представяне на кватернион и клас **QuaternionTester** за пресмятане на стойността на даден полином на една променлива

$$P(x) = a_n x^n + \dots + a_0$$

с коефициенти-кватерниони за дадена стойност *value* (кватернион) на променливата. Да се реализират двата класа, съгласно съставената спецификация.

Обекти и масиви от обекти в свободната памет. Вектор

Пример: Дадени са интерфейсът и реализацията на типа T:

Интерфейс:

```
/*
 * T.h
 *
 * Created on: 09.2011
 *
 */

#ifndef T_H_
#define T_H_


#include <iostream>
using namespace std;

class T {
public:
    //Default constructor
    T();

    //Copy constructor
    T(const T&);

    //Assignment operator
    T& operator=(const T&);

    //Destructor
    ~T();
};

#endif /* T_H_ */
```

Реализация:

```
/*
 * T.cpp
 *
 * Created on: 09.2011
 *
 */

#include "T.h"
#include <iostream>
using namespace std;

T::T() {
    cout << "T: Default constructor" << endl;
}

T::T(const T& c) {
    cout << "T: Copy constructor" << endl;
}

T& T::operator=(const T& c) {
    cout << "T: Assignment operator" << endl;
    return *this;
}

T::~T() {
    cout << "T: Destructor" << endl;
```

}

Тестване: Реализация на клас **Tester** за провеждане на тестове, които илюстрират особеностите при използването на обекти и масиви от обекти в стека и в свободната памет

Интерфейс на класа Tester:

```
/*
 * Tester.h
 *
 * Created on: 10.2011
 *
 */

#ifndef TESTER_H_
#define TESTER_H_


#include "T.h"

class Tester {
private:
    void test1();
    void test2();
    void test3();
    void test4();
    void test5();
    void test6();
    void test7();
    T f1(T);
    T& f2(T&);

public:
    void start();
};

#endif /* TESTER_H_ */
```

Реализация на класа Tester:

```
/*
 * Tester.cpp
 *
 * Created on: 10.2011
 *
 */

#include "Tester.h"
#include <iostream>
using namespace std;

typedef T *TPtr; //type pointer

void Tester::start() {
    cout << "      Test 1: Objects in the stack" << endl;
    test1();
    cout << "      Test 2: Objects in the heap" << endl;
    test2();
    cout << "      Test 3: Static array of pointers of objects in the heap"
        << endl;
    test3();
    cout << "      Test 4: Static array of objects" << endl;
    test4();
    cout << "      Test 5: Array in the heap of objects" << endl;
    test5();
    cout << "      Test 6: Array in the heap of pointers of objects in the heap"
```

```

        << endl;
test6();
cout << "      Test 7: Objects as function parameters and return value"
     << endl;
test7();
}

void Tester::test1() {
    cout << "Start..." << endl;
    cout << "      T v1" << endl;
    T v1;
    cout << "      T v2 = v1" << endl;
    T v2 = v1;
    cout << "      T v3" << endl;
    T v3;
    cout << "      v3 = v2" << endl;
    v3 = v2;
    cout << "Done!" << endl;
}

void Tester::test2() {
    cout << "Start..." << endl;
    cout << "      T* t = new T()" << endl;
    T* t = new T();
    cout << "      Delete t" << endl;
    delete t;
    cout << "Done!" << endl;
}

void Tester::test3() {
    cout << "Start..." << endl;
    cout << "      T* p[2]" << endl;
    T* p[2];
    cout << "      p[0] = new T()" << endl;
    p[0] = new T();
    cout << "      p[1] = new T()" << endl;
    p[1] = new T();
    cout << "      Delete objects in p" << endl;
    for (int i = 0; i < 2; i++)
        delete p[i];
    cout << "Done!" << endl;
}

void Tester::test4() {
    cout << "Start..." << endl;
    cout << "      T p[2]" << endl;
    T p[2];
    cout << "Done!" << endl;
}

void Tester::test5() {
    cout << "Start..." << endl;
    cout << "      T* p=new T[2]" << endl;
    T* p = new T[2];
    cout << "      Delete p" << endl;
    delete[] p;
    cout << "Done!" << endl;
}

void Tester::test6() {
    cout << "Start..." << endl;
    cout << "      TPtr* p=new TPtr[2]" << endl;
    TPtr* p = new TPtr[2];
    cout << "      p[0] = new T()" << endl;
    p[0] = new T();
    cout << "      p[1] = new T()" << endl;
}

```

```

p[1] = new T();
cout << "      Delete objects in p" << endl;
for (int i = 0; i < 2; i++)
    delete p[i];
cout << "      Delete p" << endl;
delete p;
cout << "Done!" << endl;
}

void Tester::test7() {
    cout << "Start..." << endl;
    cout << "      T obj" << endl;
    T obj;
    cout << "      f1(obj)" << endl;
    f1(obj);
    cout << "      f2(obj)" << endl;
    f2(obj);
    cout << "Done!" << endl;
}

T Tester::f1(T x) {
    cout << "f1 start..." << endl;
    cout << "f1 done!" << endl;
    return x;
}

T& Tester::f2(T& x) {
    cout << "f2 start..." << endl;
    cout << "f2 done!" << endl;
    return x;
}

```

Тестване:

```

//=====
// Name       : ObjectsInTheHeap.cpp
// Version    : 09.2011
// Description : Objects in the heap tests
//=====

#include "Tester.h"
#include <iostream>
using namespace std;

int main() {
    cout << "      Objects in the heap tests" << endl;
    Tester().start();
    return 0;
}

```

Резултати от тестването:

```

Objects in the heap tests
Test 1: Objects in the stack
Start...
      T v1
T: Default constructor
      T v2 = v1
T: Copy constructor
      T v3
T: Default constructor
      v3 = v2
T: Assignment operator
Done!
T: Destructor

```

```
T: Destructor
T: Destructor
    Test 2: Objects in the heap
Start...
    T* t = new T()
T: Default constructor
    Delete t
T: Destructor
Done!
    Test 3: Static array of pointers of objects in the heap
Start...
    T* p[2]
    p[0] = new T()
T: Default constructor
    p[1] = new T()
T: Default constructor
    Delete objects in p
T: Destructor
T: Destructor
Done!
    Test 4: Static array of objects
Start...
    T p[2]
T: Default constructor
T: Default constructor
Done!
T: Destructor
T: Destructor
    Test 5: Array in the heap of objects
Start...
    T* p=new T[2]
T: Default constructor
T: Default constructor
    Delete p
T: Destructor
T: Destructor
Done!
    Test 6: Array in the heap of pointers of objects in the heap
Start...
    TPtr* p=new TPtr[2]
    p[0] = new T()
T: Default constructor
    p[1] = new T()
T: Default constructor
    Delete objects in p
T: Destructor
T: Destructor
    Delete p
Done!
    Test 7: Objects as function parameters and return value
Start...
    T obj
T: Default constructor
    f1(obj)
T: Copy constructor
f1 start...
f1 done!
T: Copy constructor
T: Destructor
T: Destructor
    f2(obj)
f2 start...
f2 done!
Done!
T: Destructor
```

Задача: Структурите от данни вектори се отличават от масивите по това, че по време на изпълнение на програмата може да променят капацитета си. Да се напише обектно-ориентирана реализация на АТД **Vector** - *вектор с елементи от тип T и операции*, съгласно дадената спецификация – виж <http://www.cplusplus.com/reference/stl/vector/>:

Member functions

<u>(constructor)</u>	Construct vector (public member function)
<u>(destructor)</u>	Vector destructor (public member function)
<u>operator=</u>	Copy vector content (public member function)

Iterators:

<u>begin</u>	Return iterator to beginning (public member type)
<u>end</u>	Return iterator to end (public member function)
<u>rbegin</u>	Return reverse iterator to reverse beginning (public member function)
<u>rend</u>	Return reverse iterator to reverse end (public member function)

Capacity:

<u>size</u>	Return size (public member function)
<u>max_size</u>	Return maximum size (public member function)
<u>resize</u>	Change size (public member function)
<u>capacity</u>	Return size of allocated storage capacity (public member function)
<u>empty</u>	Test whether vector is empty (public member function)
<u>reserve</u>	Request a change in capacity (public member function)

Element access:

<u>operator[]</u>	Access element (public member function)
<u>at</u>	Access element (public member function)
<u>front</u>	Access first element (public member function)
<u>back</u>	Access last element (public member function)

Modifiers:

<u>assign</u>	Assign vector content (public member function)
<u>push_back</u>	Add element at the end (public member function)
<u>pop_back</u>	Delete last element (public member function)
<u>insert</u>	Insert elements (public member function)
<u>erase</u>	Erase elements (public member function)
<u>swap</u>	Swap content (public member function)
<u>clear</u>	Clear content (public member function)

На адрес <http://www.cplusplus.com/reference/stl/> може да се запознаете по-подробно със стандартната библиотека STL, която предоставя ресурси за изграждане и опериране с най-често използваните структури от данни, независимо от типа на елементите им.

Възможни са следните основни начини за реализация:

- Родов клас **Vector** с елементи от произволен тип
- Клас шаблон **Vector** с параметър – типа на елементите
- Клас **Vector** с елементи от конкретен тип

Решението на задачата включва следните основни дейности:

1. Реализация на клас **Vector** с елементи от тип **T**
2. Тестване на класа **Vector** за типа **T**
3. Настройка за използване на класа **Vector** за примитивен тип на елементите **int**
4. Реализация на клас **VectorStackOfIntegers** - *стек с елементи от тип int без ограничение за капацитета*, като илюстрация за използването на вектор

Реализацията на класа **Vector** включва част от спецификацията и е осъществена така, че от нея лесно може да се получи клас шаблон, както и да се разшири с възможността за използване на итератор, което ще стане на по-късен етап на упражненията.

Изпълнението на т.3 се улеснява с локализирането на декларацията на конкретния тип на елементите на вектора във файла **type.h**:

```
/*
 * type.h
 *
 * Created on: 09.2011
 *
 */

#ifndef TYPE_H_
#define TYPE_H_

#include "T.h"

#endif /* TYPE_H_ */
```

Интерфейс на класа Vector:

```
/*
 * Vector.h
 *
 * Created on: 09.2011
 *
 */

#ifndef VECTOR_H_
#define VECTOR_H_

#include "type.h"
#include <cstddef>

const size_t initialCapacity = 1;

class Vector {
private:
    T* data;
    size_t c; //capacity
    size_t sz; //elements count
public:
    //Constructors
    Vector();
    //Default constructor: constructs an empty vector, with no content and a
    //size of zero.

    Vector(size_t n, const T& value = T());
    //Repetitive sequence constructor: Initializes the vector with its content
    //set to a repetition, n times, of copies of value.

    Vector(const Vector& x);
    //Copy constructor: The vector is initialized to have the same contents
    //(copies) as vector x.
}
```

```

Vector& operator=(const Vector& x);
//Assigns a copy of vector x as the new content for the vector object. The
//elements contained in the vector object before the call are dropped, and
//replaced by copies of those in vector x, if any. After a call to this
//member function, both the vector object and vector x will have the same
//size and compare equal to each other.

//Destructor
~Vector();
//Destructs the vector object. This calls each of the contained element's
//destructors, and deallocates all the storage capacity allocated by the
//vector.

//Capacity:
size_t size() const;
//Returns the number of elements in the vector.

size_t max_size() const;
//Returns the maximum number of elements that the vector can hold.

void resize(size_t sz, const T& c);
//Resizes the vector to contain sz elements.

size_t capacity() const;
//Returns the size of the allocated storage space for the elements of the
//vector.

bool empty() const;
//Returns whether the vector is empty, i.e. whether its size is 0.

//Element access:
T& operator[](size_t n);
//Returns a reference to the element at position n in the vector.

T& at(size_t n);
//Returns a reference to the element at position n in the vector. The
//difference between this member function and member operator function
//operator[] is that vector::at signals if the requested position is out of
//range by throwing an out_of_range exception.

T& front();
//Returns a reference to the first element in the vector.

T& back();
//Returns a reference to the last element in the vector.

//Modifiers:
void assign(size_t n, const T& u);
//Assigns new content to the vector object, dropping all the elements
//contained in the vector before the call and replacing them by those
//specified by the parameters: The new content is the repetition n times of
//copies of element u.

void push_back(const T& x);
//Adds a new element at the end of the vector, after its current last
//element. The content of this new element is initialized to a copy of x.
//This effectively increases the vector size by one, which causes a
//reallocation of the internal allocated storage if the vector size was
//equal to the vector capacity before the call.

void pop_back();
//Removes the last element in the vector, effectively reducing the vector
//size by one. This calls the removed element's destructor.

void insert(size_t position, size_t n, const T& x);

```

```
//The vector is extended by inserting n elements before the element at
//position. This effectively increases the vector size, which causes an
//automatic reallocation of the allocated storage space if, and only if, the
//new vector size surpasses the current vector capacity.

void erase(size_t position);
void erase(size_t first, size_t last);
//Removes from the vector either a single element (position) or a range of
//elements [first,last). This effectively reduces the vector size by the
//number of elements removed, calling each element's destructor before.

void clear();
//All the elements of the vector are dropped: their destructors are called,
//and then they are removed from the vector, leaving the vector with a size
//of 0.
};

#endif /* VECTOR_H_ */
```

Реализация на класа Vector:

```
/*
 * Vector.cpp
 *
 * Created on: 09.2011
 *
 */

#include "Vector.h"
#include <exception>
#include <iostream>
using namespace std;

Vector::Vector() {
    cout << "Vector: Default constructor" << endl;
    c = initialCapacity;
    data = new T[c];
    sz = 0;
}

Vector::Vector(size_t n, const T& value) {
    cout << "Vector: Constructor with parameters" << endl;
    c = sz = n;
    data = new T[c];
    for (size_t i = 0; i < n; i++)
        data[i] = value;
}

Vector::Vector(const Vector& x) {
    cout << "Vector: Copy constructor" << endl;
    c = x.c;
    sz = x.sz;
    data = new T[c];
    for (size_t i = 0; i < sz; i++)
        data[i] = x.data[i];
}

Vector& Vector::operator=(const Vector& x) {
    cout << "Vector: Assignment operator" << endl;
    if (this != &x) {
        delete[] data;
        c = x.c;
        sz = x.sz;
        data = new T[c];
        for (size_t i = 0; i < sz; i++)
            data[i] = x.data[i];
    }
}
```

```

    }
    return *this;
}

Vector::~Vector() {
    cout << "Vector: Destructor" << endl;
    delete[] data;
}

size_t Vector::size() const {
    return sz;
}

void Vector::resize(size_t sz, const T& c) {
    *this = Vector(sz, c);
}

size_t Vector::capacity() const {
    return c;
}

bool Vector::empty() const {
    return sz == 0;
}

T& Vector::operator[](size_t n) {
    return data[n];
}

T& Vector::at(size_t n) {
    string msg("Vector: index out of range!");
    if (n < 0 || n >= sz)
        throw msg;
    else
        return data[n];
}

T& Vector::front() {
    return data[0];
}

T& Vector::back() {
    return data[sz - 1];
}

void Vector::assign(size_t n, const T& u) {
    resize(n, u);
}

void Vector::push_back(const T& x) {
    insert(sz, 1, x);
}

void Vector::pop_back() {
    erase(sz - 1);
}

void Vector::insert(size_t position, size_t n, const T& x) {
    if (sz + n > c)
        c += 2 * (sz + n);
    T* temp = new T[c];
    for (size_t i = 0; i < position; i++)
        temp[i] = data[i];
    for (size_t i = 0; i < n; i++)
        temp[position + i] = x;
    for (size_t i = position; i < sz; i++)
        temp[i] = data[i];
    delete[] data;
    data = temp;
}

```

```

        temp[i + n] = data[i];
    delete[] data;
    data = temp;
    sz += n;
}

void Vector::erase(size_t position) {
    erase(position, position + 1);
}

void Vector::erase(size_t first, size_t last) {
    T* temp = new T[c];
    size_t n = 0;
    for (size_t i = 0; i < first; i++)
        temp[n++] = data[i];
    for (size_t i = last; i < sz; i++)
        temp[n++] = data[i];
    delete[] data;
    data = temp;
    sz -= last - first;
}

void Vector::clear() {
    delete[] data;
    c = initialCapacity;
    data = new T[c];
    sz = 0;
}

```

Тестване на класа *Vector* за елементи от тип *T*: Реализация на клас **Tester** за провеждане на тестове, които илюстрират особеностите при използването на обекти в свободната памет.

Класть **Vector** може да се използва за примитивен тип на елементите **int** след промяна на файла **type.h** и транслация:

```

/*
 * type.h
 *
 * Created on: 09.2011
 *
 */

#ifndef TYPE_H_
#define TYPE_H_

typedef int T;

#endif /* TYPE_H_ */

```

Реализацията на класа **VectorStackOfIntegers** се основава на следната спецификация – виж <http://www.cplusplus.com/reference/stl/stack/>:

Member functions

<u>(constructor)</u>	Construct stack (public member function)
<u>empty</u>	Test whether container is empty (public member function)
<u>size</u>	Return size (public member function)
<u>top</u>	Access next element (public member function)
<u>push</u>	Add element (public member function)
<u>pop</u>	Remove element (public member function)

Интерфейс на класа *VectorStackOfIntegers*:

```

/*
 * VectorStackOfIntegers.h
 *
 * Created on: 09.2011
 *
 */

#ifndef VECTORSTACKOFINTEGERS_H_
#define VECTORSTACKOFINTEGERS_H_


#include "Vector.h"
#include <cstddef>

class VectorStackOfIntegers {
private:
    Vector vector;
public:
    VectorStackOfIntegers();
    //Default constructor

    VectorStackOfIntegers(const VectorStackOfIntegers& x);
    //Copy constructor

    VectorStackOfIntegers& operator=(const VectorStackOfIntegers& x);
    //Assigns a copy of VectorStackOfIntegers x

    ~VectorStackOfIntegers();
    //Destructor

    bool empty() const;
    //Returns whether the stack is empty, i.e. whether its size is 0.

    size_t size() const;
    //Returns the number of elements in the stack.

    T& top();
    //Returns a reference to the next element in the stack.

    void push(const T& x);
    //Adds a new element at the top of the stack, above its current top element.

    void pop();
    //Removes the element on top of the stack.
};

#endif /* VECTORSTACKOFINTEGERS_H_ */

```

Реализация на класа *VectorStackOfIntegers*:

```

/*
 * VectorStackOfIntegers.cpp
 *
 * Created on: 09.2011
 *
 */

#include "VectorStackOfIntegers.h"
#include <iostream>
using namespace std;

VectorStackOfIntegers::VectorStackOfIntegers() {
    cout << "VectorStackOfIntegers: Default constructor" << endl;
}

```

```

VectorStackOfIntegers::VectorStackOfIntegers (const VectorStackOfIntegers& x) :
    vector(x.vector) {
    cout << "VectorStackOfIntegers: Copy constructor" << endl;
}

VectorStackOfIntegers& VectorStackOfIntegers::operator= (
    const VectorStackOfIntegers& x) {
    cout << "VectorStackOfIntegers: Assignment operator" << endl;
    vector = x.vector;
    return *this;
}

VectorStackOfIntegers::~VectorStackOfIntegers () {
    cout << "VectorStackOfIntegers: destructor" << endl;
}

bool VectorStackOfIntegers::empty () const {
    return vector.empty ();
}

size_t VectorStackOfIntegers::size () const {
    return vector.size ();
}

T& VectorStackOfIntegers::top () {
    return vector.back ();
}

void VectorStackOfIntegers::push (const T& x) {
    vector.push_back (x);
}

void VectorStackOfIntegers::pop () {
    vector.pop_back ();
}

```

Тестване на класа **VectorStackOfIntegers**: Реализация на клас **StackTester**.

Задача: Да се реализира клас **Matrix** за работа с матрици от реални числа, като елементите на матрица A(m,n) се съхраняват във вектор с m компоненти, като i-тата компонента е вектор с n компоненти - елементите на ред с номер i на A.

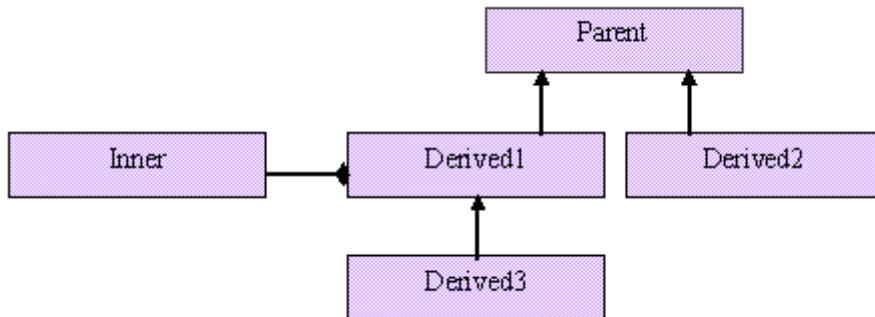
Задача: Да се реализира клас **SortedIntegersPrinter** за четене на числа, съхраняване във вектор, сортиране на вектора във възходящ ред и извеждане на вектора.

Задача: Да се реализира клас **SortedStringsPrinter** за четене на низове, съхраняване във вектор, сортиране на вектора във възходящ ред и извеждане на вектора.

Задача: Да се реализира клас **VectorBigInteger** за представяне на голямо цяло десетично число без знак, като цифрите на голямо цяло десетично число $N = a_n \text{base}^n + a_{n-1} \text{base}^{n-1} + \dots + a_1 \text{base} + a_0$, $0 \leq a_i < \text{base}$, се съхраняват във вектор $\text{digits} = (a_0, \dots, a_n)$ и $\text{base} = 10^k$, $k \geq 1$. Класът да предоставя средства за *въвеждане, събиране, умножение, сравнение и отпечатване* на такива числа.

Наследяване и полиморфизъм. Абстрактни класове. Хетерогенен сортиран линеен едносързан списък. Матрица

Пример: Разглеждаме йерархията от класове:



Реализация на класа Parent:

```

/*
 * Parent.h
 *
 * Created on: 09.2011
 *
 */

#ifndef PARENT_H_
#define PARENT_H_

#include <iostream>
using namespace std;

class Parent {
public:
    Parent(){
        cout << "Parent: Default constructor" << endl;
    }
    virtual ~Parent(){
        cout << "Parent: Destructor" << endl;
    }
    virtual void print() const{
        cout << "Parent: Print" << endl;
    }
};

#endif /* PARENT_H_ */
  
```

Реализация на класа Inner:

```

/*
 * Inner.h
 *
 * Created on: 29.09.2011
 *
 */

#ifndef INNER_H_
#define INNER_H_

#include <iostream>
using namespace std;

class Inner {
public:
    Inner() {
        cout << "Inner: Default constructor" << endl;
    }
    virtual ~Inner(){
        cout << "Inner: Destructor" << endl;
    }
    virtual void print() const{
        cout << "Inner: Print" << endl;
    }
};

#endif /* INNER_H_ */
  
```

```

        cout << "Inner: Default constructor" << endl;
    }
~Inner() {
    cout << "Inner: Destructor" << endl;
}
void print() const {
    cout << "Inner: Print" << endl;
}
};

#endif /* INNER_H_ */

```

Реализация на класа **Derived1**:

```

/*
 * Derived1.h
 *
 * Created on: 09.2011
 *
 */

#ifndef DERIVED1_H_
#define DERIVED1_H_

#include "Parent.h"
#include "Inner.h"
#include <iostream>
using namespace std;

class Derived1 : public Parent{
private:
    Inner obj;
public:
    Derived1() {
        cout << "Derived1: Default constructor" << endl;
    }
    ~Derived1() {
        cout << "Derived1: Destructor" << endl;
    }
    void print() const{
        cout << "Derived1: Print" << endl;
        Parent::print();
        obj.print();
    }
};
};

#endif /* DERIVED1_H_ */

```

Реализация на класа **Derived2**:

```

/*
 * Derived2.h
 *
 * Created on: 09.2011
 *
 */

#ifndef DERIVED2_H_
#define DERIVED2_H_

#include "Parent.h"
#include <iostream>
using namespace std;

class Derived2: public Parent {

```

```
public:
    Derived2() {
        cout << "Derived2: Default constructor" << endl;
    }
    ~Derived2() {
        cout << "Derived2: Destructor" << endl;
    }
    void print() const {
        cout << "Derived2: Print" << endl;
        Parent::print();
    }
};

#endif /* DERIVED2_H_ */
```

Реализация на класа **Derived3**:

```
/*
 * Derived3.h
 *
 * Created on: 09.2011
 *
 */

#ifndef DERIVED3_H_
#define DERIVED3_H_

#include "Derived1.h"
#include <iostream>
using namespace std;

class Derived3: public Derived1 {
public:
    Derived3() {
        cout << "Derived3: Default constructor" << endl;
    }
    ~Derived3() {
        cout << "Derived3: Destructor" << endl;
    }
    void print() const {
        cout << "Derived3: Print" << endl;
        Derived1::print();
    }
};
#endif /* DERIVED3_H_ */
```

Тестване: Реализация на клас **Tester**

Интерфейс на класа **Tester**:

```
/*
 * Tester.h
 *
 * Created on: 10.2011
 *
 */

#ifndef TESTER_H_
#define TESTER_H_

class Tester {
private:
    void test1();
    void test2();
};
```

```

public:
    void start();
};

#endif /* TESTER_H_ */

```

Реализация на класа Tester:

```

/*
 * Tester.cpp
 *
 * Created on: 10.2011
 *
 */

#include "Tester.h"
#include "Parent.h"
#include "Derived2.h"
#include "Derived3.h"
#include <iostream>
using namespace std;

void Tester::start() {
    cout << "      Test 1" << endl;
    test1();
    cout << "      Test 2" << endl;
    test2();
}

void Tester::test1() {
    cout << "Start..." << endl;
    cout << "      Derived3 d" << endl;
    Derived3 d;
    cout << "Done!" << endl;
}

void Tester::test2() {
    cout << "Start..." << endl;
    cout << "      Parent* p[4]" << endl;
    Parent* p[4];
    cout << "      p[0] = new Parent()" << endl;
    p[0] = new Parent();
    cout << "      p[1] = new Derived1()" << endl;
    p[1] = new Derived1();
    cout << "      p[2] = new Derived2()" << endl;
    p[2] = new Derived2();
    cout << "      p[3] = new Derived3()" << endl;
    p[3] = new Derived3();
    cout << "      Print" << endl;
    for (int i = 0; i < 4; i++)
        p[i]->print();
    cout << "      Delete" << endl;
    for (int i = 0; i < 4; i++)
        delete p[i];
    cout << "Done!" << endl;
}

```

Резултати от тестването:

```

//=====
// Name       : InheritanceAndPolymorphism.cpp
// Version    : 09.2011
// Description : Inheritance And Polymorphism tests
//=====

```

```

#include "Tester.h"
#include <iostream>
using namespace std;

int main() {
    cout << "      Test Inheritance And Polymorphism" << endl;
    Tester().start();
    return 0;
}

Test Inheritance And Polymorphism
Test 1
Start...
Derived3 d
Parent: Default constructor
Inner: Default constructor
Derived1: Default constructor
Derived3: Default constructor
Done!
Derived3: Destructor
Derived1: Destructor
Inner: Destructor
Parent: Destructor
Test 2
Start...
    Parent* p[4]
    p[0] = new Parent()
Parent: Default constructor
    p[1] = new Derived1()
Parent: Default constructor
Inner: Default constructor
Derived1: Default constructor
    p[2] = new Derived2()
Parent: Default constructor
Derived2: Default constructor
    p[3] = new Derived3()
Parent: Default constructor
Inner: Default constructor
Derived1: Default constructor
Derived3: Default constructor
Print
Parent: Print
Derived1: Print
Parent: Print
Inner: Print
Derived2: Print
Parent: Print
Derived3: Print
Derived1: Print
Parent: Print
Inner: Print
Delete
Parent: Destructor
Derived1: Destructor
Inner: Destructor
Parent: Destructor
Derived2: Destructor
Parent: Destructor
Derived3: Destructor
Derived1: Destructor
Inner: Destructor
Parent: Destructor
Done!

```

Задача: Напишете какво ще изведе програмата **Tests**:

```

//=====
// Name      : Tests.cpp
// Version   : 10.2011
// Description : Tests
//=====

#include "Tester1.h"
#include "Tester2.h"
#include "Tester3.h"
#include "Tester4.h"
#include "Tester5.h"
#include <iostream>
using namespace std;

int main() {
    cout << "      Test1..." << endl;
    Tester1().start();

    cout << "      Test2..." << endl;
    Tester2().start();

    cout << "      Test3..." << endl;
    Tester3().start();

    cout << "      Test4..." << endl;
    Tester4().start();

    cout << "      Test5..." << endl;
    Tester5().start();
    return 0;
}

```

Резултати от работата на Tester1():

```

/*
 * Tester1.h
 *
 * Created on: 10.2011
 *
 */

#ifndef TESTER1_H_
#define TESTER1_H_


class Tester1 {
public:
    void start();
};

#endif /* TESTER1_H_ */


/*
 * Tester1.cpp
 *
 * Created on: 10.2011
 *
 */

#include "Tester1.h"
#include <iostream>
using namespace std;

class Parent {
public:

```

```

void print(int n) const {
    cout << "Parent print: " << n << endl;
}
};

class Derived: public Parent {
public:
    void print(int n) const {
        cout << "Derived print: " << n << endl;
        if (n <= 1)
            Parent::print(n);
        else if (n % 2 == 0)
            print(n / 2);
        else
            print(3 * n + 1);
    }
};

void Tester1::start() {
    Derived D;
    D.print(5);
}

```

Резултати от работата на Tester2():

```

/*
 * Tester2.h
 *
 * Created on: 10.2011
 *
 */

#ifndef TESTER2_H_
#define TESTER2_H_


class Tester2 {
public:
    void start();
};

#endif /* TESTER2_H_ */


/*
 * Tester2.cpp
 *
 * Created on: 10.2011
 *
 */

#include "Tester2.h"
#include <iostream>
using namespace std;

void Tester2::start() {
    cout << "Not implemented!" << endl;
}

```

Резултати от работата на Tester3():

```

/*
 * Tester3.h
 *
 * Created on: 10.2011
 *
 */

```

```

#ifndef TESTER3_H_
#define TESTER3_H_

class Tester3 {
public:
    void start();
};

#endif /* TESTER3_H_ */

/*
 * Tester3.cpp
 *
 * Created on: 10.2011
 */
#include "Tester3.h"
#include <iostream>
using namespace std;

class Parent {
public:
    Parent() {
        cout << "Constructor Parent()" << endl;
    }
    Parent(int n) {
        cout << "Constructor Parent(" << n << ")" << endl;
    }
};

class Derived: public Parent {
private:
    Parent P;
public:
    Derived() {
        cout << "Constructor Derived()" << endl;
    }
    Derived(int n) :
        Parent(n) {
        P = Parent(-n);
        cout << "Constructor Derived(" << n << ")" << endl;
    }
};

void Tester3::start() {
    cout << "Derived(7):" << endl;
    Derived D = Derived(7);
    cout << "Derived():" << endl;
    D = Derived();
}

```

Резултати от работата на Tester4():

```

/*
 * Tester4.h
 *
 * Created on: 10.2011
 */
#ifndef TESTER4_H_
#define TESTER4_H_

```

```

class Tester4 {
public:
    void start();
};

#ifndef /* TESTER4_H */
/*
 * Tester4.cpp
 *
 * Created on: 10.2011
 *
 */

#include "Tester4.h"
#include <iostream>
using namespace std;

class Parent {
public:
    void method1() {
        cout << "Parent method1" << endl;
    }
    void method2() {
        cout << "Parent method2" << endl;
    }
};

class Derived: public Parent {
public:
    void method1() {
        Parent::method1();
        cout << "Derived method1" << endl;
    }
};

void Tester4::start() {
    Parent P;
    P.method1();
    P.method2();

    Derived D;
    D.method1();
    D.method2();

    P = D;
    P.method1();
    P.method2();
}

```

Резултати от работата на Tester5():

```

/*
 * Tester5.h
 *
 * Created on: 10.2011
 *
 */

#ifndef TESTER5_H_
#define TESTER5_H_


class Tester5 {
public:
    void start();

```

```

};

#ifndef /* TESTER5_H_ */

/*
 * Tester5.cpp
 *
 * Created on: 10.2011
 *
 */

#include "Tester5.h"
#include <iostream>
using namespace std;

class Parent {
public:
    virtual void method1() {
        cout << "Parent method1" << endl;
    }
    void method2() {
        cout << "Parent method2" << endl;
    }
};

class Derived: public Parent {
public:
    void method1() {
        Parent::method1();
        cout << "Derived method1" << endl;
    }
};

void Tester5::start() {
    Parent P;
    P.method1();
    P.method2();

    Derived D;
    D.method1();
    D.method2();

    Parent *Q = new Parent();
    Q->method1();
    Q->method2();
    delete Q;

    Q = new Derived();
    Q->method1();
    Q->method2();
    delete Q;
}
}

```

Задача: Колекция от печатни издания съдържа екземпляри от следните видове:



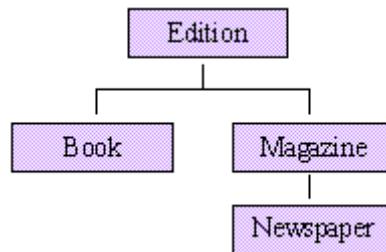
- Издание, което има заглавие, издателство и година на издаване
- Книга (вид печатно издание), която има заглавие, издателство, година на издаване и автора

- Списание (вид печатно издание), което има заглавие, издателство, година на издаване и пореден номер
- Вестник (вид печатно издание), който има заглавие, издателство, година на издаване, номер на месец, номер на ден и пореден номер

Да се реализира клас **EditionsList** за представяне на колекция от печатни издания чрез линеен едносвързан списък от различните екземпляри на видовете издания в колекцията, сортиран по годината на издаване.

Реализация:

1. Йерархия от класове за изданията



1.1. Клас Edition

Интерфейс:

```

/*
 * Edition.h
 *
 * Created on: 10.2011
 *
 */

#ifndef EDITION_H_
#define EDITION_H_

#include "EditionList2.h"
#include <string>
using namespace std;

class Edition {
    friend class EditionList1;
    friend class EditionList2;
private:
    Edition *next; //For EditionList2 only
    virtual Edition* create() const;
protected:
    string title;
    string publish;
    int year;
public:
    Edition(string = "", string = "", int = 0);
    virtual void print() const;
    bool operator<(const Edition&) const;
    bool operator==(const Edition&) const;
    virtual string toString() const;
    virtual bool equals(const Edition&) const;
    virtual ~Edition();
};

#endif /* EDITION_H_ */
  
```

Реализация:

```
/*
 * Edition.cpp
 *
 * Created on: 10.2011
 *
 */

#include "Edition.h"
#include <iostream>
#include <iostream>
using namespace std;

Edition::Edition(string t, string p, int y) {
    title = t;
    publish = p;
    year = y;
}

string Edition::toString() const {
    stringstream s;
    s << "\n" << title << "\n" << publish << "\n" << year << "\n";
    return s.str();
}

void Edition::print() const {
    cout << "Title: " << title << endl;
    cout << "Publish: " << publish << endl;
    cout << "Year: " << year << endl;
}

Edition* Edition::create() const {
    return new Edition(title, publish, year);
}

bool Edition::operator<(const Edition &n) const {
    return year < n.year;
}

bool Edition::operator==(const Edition &n) const {
    return year == n.year;
}

bool Edition::equals(const Edition &n) const {
    return this->toString() == (&n)->toString();
}

Edition::~Edition() {
    cout << "Edition: Destructor" << endl;
}
```

1.2. Клас Book

Интерфейс:

```
/*
 * Book.h
 *
 * Created on: 10.2011
 *
 */

#ifndef BOOK_H_
#define BOOK_H_
```

```
#include "Edition.h"
#include <string>
using namespace std;

class Book: public Edition {
private:
    Edition* create() const;
protected:
    string author;
public:
    Book(string = "", string = "", int = 0, string = "");
    void print() const;
    string toString() const;
    bool equals(const Edition&) const;
    ~Book();
    //Public methods - inherited from Edition
};

#endif /* BOOK_H_ */
```

Реализация:

```
/*
 * Book.cpp
 *
 * Created on: 10.2011
 *
 */

#include "Book.h"
#include <sstream>
#include <iostream>
using namespace std;

Book::Book(string t, string p, int y, string a) :
    Edition(t, p, y) {
    author = a;
}

string Book::toString() const {
    stringstream s;
    s << author << "\n";
    return Edition::toString() + s.str();
}

void Book::print() const {
    Edition::print();
    cout << "Author: " << author << endl;
}

Edition* Book::create() const {
    return new Book(title, publish, year, author);
}

bool Book::equals(const Edition &n) const {
    return this->toString() == (&n)->toString();
}

Book::~Book() {
    cout << "Book: Destructor" << endl;
}
```

1.3. Клас Magazine

Интерфейс:

```
/*
 * Magazine.h
 *
 * Created on: 10.2011
 *
 */

#ifndef MAGAZINE_H_
#define MAGAZINE_H_

#include "Edition.h"
#include <string>
using namespace std;

class Magazine: public Edition {
private:
    Edition* create() const;
protected:
    int number;
public:
    Magazine(string = "", string = "", int = 0, int = 0);
    void print() const;
    string toString() const;
    bool equals(const Edition&) const;
    ~Magazine();
    //Public methods - inherited from Edition
};

#endif /* MAGAZINE_H_ */
```

Реализация:

```
/*
 * Magazine.cpp
 *
 * Created on: 10.2011
 *
 */

#include "Magazine.h"
#include <iostream>
#include <iomanip>
using namespace std;

Magazine::Magazine(string t, string p, int y, int n) :
    Edition(t, p, y) {
    number = n;
}

string Magazine::toString() const {
    stringstream s;
    s << number << "\n";
    return Edition::toString() + s.str();
}

void Magazine::print() const {
    Edition::print();
    cout << "Number: " << number << endl;
}

bool Magazine::equals(const Edition &n) const {
```

```

    return this->toString() == (&n)->toString();
}

Edition* Magazine::create() const {
    return new Magazine(title, publish, year, number);
}

Magazine::~Magazine() {
    cout << "Magazine: Destructor" << endl;
}

```

1.4. Клас Newspaper

Интерфејс:

```

/*
 * Newspaper.h
 *
 * Created on: 10.2011
 *
 */

#ifndef NEWSPAPER_H_
#define NEWSPAPER_H_

#include "Edition.h"
#include "Magazine.h"
#include <iostream>
using namespace std;

class Newspaper: public Magazine {
private:
    Edition* create() const;
protected:
    int day;
    int month;
public:
    Newspaper(string = "", string = "", int = 0, int = 0, int = 0, int = 0);
    void print() const;
    string toString() const;
    bool equals(const Edition&) const;
    ~Newspaper();
    //Public methods - inherited from Edition and Magazine
};

#endif /* NEWSPAPER_H_ */

```

Реализация:

```

/*
 * Newspaper.cpp
 *
 * Created on: 10.2011
 *
 */

#include "Newspaper.h"
#include <iostream>
#include <iomanip>
using namespace std;

Newspaper::Newspaper(string t, string p, int y, int n, int d, int m) :
    Magazine(t, p, y, n) {
    day = d;
    month = m;
}

```

```

}

string Newspaper::toString() const {
    stringstream s;
    s << day << "\n" << month << "\n";
    return Magazine::toString() + s.str();
}

void Newspaper::print() const {
    Magazine::print();
    cout << "Day: " << day << endl;
    cout << "Month: " << month << endl;
}

bool Newspaper::equals(const Edition &n) const {
    return this->toString() == (&n)->toString();
}

Edition* Newspaper::create() const {
    return new Newspaper(title, publish, year, number, day, month);
}

Newspaper::~Newspaper() {
    cout << "Newspaper: Destructor" << endl;
}

```

2. Клас **EditionList1** – представяне на колекцията чрез линеен едносвързан списък, сортиран по годината на издаване, като всеки възел съдържа указател към обект, принадлежащ на горната йерархия

*Интерфейс на класа **EditionList1**:*

```

/*
 * EditionList1.h
 *
 * Created on: 10.2011
 *
 */

#ifndef EDITIONLIST1_H_
#define EDITIONLIST1_H_

#include "Edition.h"

class EditionNode {
    friend class EditionList1;
private:
    Edition *ptr;
    EditionNode *next;
public:
    EditionNode() {
        next = 0;
    }
};

class EditionList1 {
private:
    EditionNode *head;
public:
    EditionList1();
    void insert(Edition*);
    void print() const;
    ~EditionList1();
};

```

```
#endif /* EDITIONLIST1_H_ */
```

3. Клас **EditionList2** – представяне на колекцията чрез линеен едносвързан списък, сортиран по годината на издаване, като всеки възел съдържа обект, принадлежащ на горната иерархия

Интерфейс на класа EditionList2:

```
/*
 * EditionList2.h
 *
 * Created on: 10.2011
 *
 */

#ifndef EDITIONLIST2_H_
#define EDITIONLIST2_H_

#include "Edition.h"

class Edition;
class EditionList2 {
private:
    Edition *head;
public:
    EditionList2();
    void insert(Edition*);
    void print() const;
    ~EditionList2();
};

#endif /* EDITIONLIST2_H_ */
```

Реализация на класа EditionList2:

```
/*
 * EditionList2.cpp
 *
 * Created on: 10.2011
 *
 */

#include "EditionList2.h"
#include <iostream>
using namespace std;

EditionList2::EditionList2() {
    head = 0;
}

void EditionList2::insert(Edition *n) {
    Edition *current = head;
    Edition *previous = 0;
    while (current != 0 && *current < *n) {
        previous = current;
        current = current->next;
    }
    Edition* ptr = n->create();
    ptr->next = current;
    if (previous == 0)
        head = ptr;
    else
        previous->next = ptr;
}
```

```

void EditionList2::print() const {
    Edition *current = head;
    while (current != 0) {
        current->print();
        current = current->next;
    }
}

EditionList2::~EditionList2() {
    cout << "EditionList2: Destructor" << endl;
    while (head != 0) {
        Edition *current = head;
        head = head->next;
        delete current;
    }
}

```

При реализацията се използва обект на класа **stringstream** за преобразуване на обект, принадлежащ на йерархията от класове за изданията, в обект на класа **string** – виж <http://www.cplusplus.com/reference/iostream/stringstream/stringstream/>.

Модифицирайте функцията **insert**, така че в списъка да се съхраняват само различните екземпляри от дадено издание, съгласно условието на задачата.

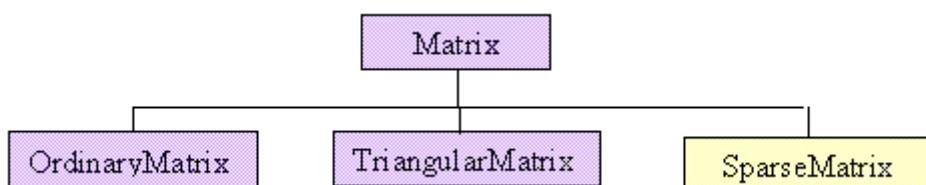
Задача: Да се реализират програмни средства за работа с различни видове математически матрици:



представящи операции за:

- Създаване и инициализация на матрица
- Получаване на броя на редовете на матрица
- Получаване на броя на стълбовете на матрица
- Получаване на стойност на елемент на матрица по зададени индекси
- Промяна на стойността на елемент на матрица по зададени индекси и нова стойност
- Събиране на матрици
- Умножение на матрици
- Умножение на матрица с число
- Транспониране на матрица
- Вход/Изход на матрица

Реализация: Йерархия от класове:



1. Абстрактен клас **Matrix**

*Интерфейс на абстрактен клас **Matrix**:*

```

/*
 * Matrix.h
 *
 * Created on: 09.2011
 *
 */

#ifndef MATRIX_H_
#define MATRIX_H_


const int MAX_N = 100;
const int MAX_M = 100;

class Matrix {
protected:
    int n; // number of rows
    int m; // number of columns

    int f(int i, int j) const;
    //Проверява дали i и j са коректни индекси. Ако е така, определя и връща
    //позицията във вътрешното представяне за i-j-тия елемент на матрица.
    //В противен случай активира изключение

    virtual Matrix& create(int rows, int cols) const=0;
    //Създава нулева матрица с rows реда и cols стълба
public:
    //Constructor
    Matrix(int rows = MAX_N, int cols = MAX_M);

    //Abstract methods
    virtual double elementAt(int i, int j) const =0;
    virtual double setElementAt(int i, int j, double value)=0;

    //Public methods
    int getRows() const;
    int getColumnns() const;
    virtual Matrix& operator=(const Matrix &arg);
    virtual Matrix& operator+(const Matrix &arg) const;
    //...

    //Desturctor
    virtual ~Matrix();
};

#endif /* MATRIX_H_ */

```

*Реализация на абстрактен клас **Matrix**:*

```

/*
 * Matrix.cpp
 *
 * Created on: 09.2011
 *
 */

#include "Matrix.h"
#include <string>
#include <exception>
#include <iostream>
using namespace std;

Matrix::Matrix(int rows, int cols) {
    n = rows;
    m = cols;
}

```

```

}

int Matrix::f(int i, int j) const {
    string msg("Incorrect indexes!");
    if (i < 1 || i > n || j < 1 || j > m)
        throw msg;
    return -1;
}

int Matrix::getRows() const {
    return n;
}

int Matrix::getColumns() const {
    return m;
}

Matrix& Matrix::operator=(const Matrix &arg) {
    cout << "Matrix: Operator =" << endl;
    string msg("Matrix: Illegal matrix dimensions!");
    if (arg.n != n || arg.m != m)
        throw msg;
    n = arg.getRows();
    m = arg.getColumns();
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            setElementAt(i, j, arg.elementAt(i, j));
    return *this;
}

Matrix& Matrix::operator+(const Matrix &arg) const {
    cout << "Matrix: Operator +" << endl;
    string msg("Matrix: Illegal matrix dimensions!");
    if (arg.n != n || arg.m != m)
        throw msg;
    Matrix &result = create(n, m);
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            result.setElementAt(i, j, elementAt(i, j) + arg.elementAt(i, j));
    return result;
}

Matrix::~Matrix() {
    cout << "Matrix: Destructor" << endl;
}

```

2. Клас **OrdinaryMatrix**. Реализацията са основава на представянето на елементите a_{ij} на матрица $A(n \times m)$, $i \in [1; n]$, $j \in [1; m]$ чрез масив **double elements[MAX_N][MAX_M]**, като $\text{elements}[i-1][j-1] = a_{ij}$

Интерфейс на класа OrdinaryMatrix:

```

/*
 * OrdinaryMatrix.h
 *
 * Created on: 09.2011
 *
 */

#ifndef ORDINARYMATRIX_H_
#define ORDINARYMATRIX_H_

#include "Matrix.h"

class OrdinaryMatrix: public Matrix {

```

```

private:
    double elements[MAX_N][MAX_M];
protected:
    //Protected method f - inherited from Matrix
    Matrix& create(int rows, int cols) const;
public:
    //Constructors
    OrdinaryMatrix(int rows = MAX_N, int cols = MAX_M);
    OrdinaryMatrix(double data[MAX_N][MAX_M], int rows, int cols);
    OrdinaryMatrix(const Matrix &arg);

    //Public methods
    double elementAt(int i, int j) const;
    double setElementAt(int i, int j, double value);

    //Public methods - inherited from Matrix
};

#endif /* ORDINARYMATRIX_H_ */

```

Реализация на класа OrdinaryMatrix:

```

/*
 * OrdinaryMatrix.cpp
 *
 * Created on: 09.2011
 *
 */

#include "OrdinaryMatrix.h"
#include <iostream>
using namespace std;

OrdinaryMatrix::OrdinaryMatrix(int rows, int cols) :
    Matrix(rows, cols) {
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            setElementAt(i, j, 0);
}

OrdinaryMatrix::OrdinaryMatrix(double data[MAX_N][MAX_M], int rows, int cols) :
    Matrix(rows, cols) {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            setElementAt(i + 1, j + 1, data[i][j]);
}

OrdinaryMatrix::OrdinaryMatrix(const Matrix& arg) :
    Matrix(arg.getRows(), arg.getColumns()) {
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            setElementAt(i, j, arg.elementAt(i, j));
}

double OrdinaryMatrix::elementAt(int i, int j) const {
    f(i, j);
    return elements[i - 1][j - 1];
}

double OrdinaryMatrix::setElementAt(int i, int j, double value) {
    f(i, j);
    double oldValue = elements[i - 1][j - 1];
    elements[i - 1][j - 1] = value;
    return oldValue;
}

```

```
Matrix& OrdinaryMatrix::create(int rows, int cols) const {
    cout << "OrdinaryMatrix::create" << endl;
    return *(new OrdinaryMatrix(rows, cols));
}
```

Променете реализациата така, че представянето на елементите a_{ij} на матрица $A(nxm)$, $i \in [1;n]$, $j \in [1;m]$ да е чрез масив

double elements[MAX_N*MAX_M]

като $\text{elements} = (a_{11}, \dots, a_{1m}, a_{21}, \dots, a_{2m}, \dots, a_{n1}, \dots, a_{nm})$ и $\text{elements}[(i-1)*m+j-1] = a_{ij}$.

3. Клас **TriangularMatrix**. Реализацията са основава на представянето на елементите a_{ij} на триъгълна матрица $A(nxm)$, $n=m$, $i \in [1;n]$, $j \in [1;m]$ чрез масив

double elements[((1+MAX_N)*MAX_N)/2]

като $\text{elements} = (a_{11}, a_{21}, a_{22}, \dots, a_{n1}, \dots, a_{nn})$ и $\text{elements}[(i*(i-1))/2 + j - 1] = a_{ij}$

Интерфейс на класа TriangularMatrix:

```
/*
 * TriangularMatrix.h
 *
 * Created on: 09.2011
 *
 */

#ifndef TRIANGULARMATRIX_H_
#define TRIANGULARMATRIX_H_

#include "Matrix.h"

class TriangularMatrix: public Matrix {
private:
    double elements[((1 + MAX_N) * MAX_N) / 2];
protected:
    //Override
    int f(int i, int j) const;
    Matrix& create(int rows, int cols) const;
public:
    //Constructors
    TriangularMatrix(int n = MAX_N);
    TriangularMatrix(double data[MAX_N][MAX_M], int n);
    TriangularMatrix(const Matrix &arg);

    //Public methods
    double elementAt(int i, int j) const;
    double setElementAt(int i, int j, double value);

    //Public methods - inherited from Matrix

    //Override
    Matrix& operator=(const Matrix &arg);
    Matrix& operator+(const Matrix &arg) const;
};

#endif /* TRIANGULARMATRIX_H_ */
```

Реализация на класа TriangularMatrix:

```
/*
 * TriangularMatrix.cpp
 *
 * Created on: 09.2011
 *
 */
```

```

#include "TriangularMatrix.h"
#include "OrdinaryMatrix.h"
#include <string>
#include <exception>
#include <iostream>
using namespace std;

TriangularMatrix::TriangularMatrix(int n) :
    Matrix(n, n) {
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            setElementAt(i, j, 0);
}

TriangularMatrix::TriangularMatrix(double data[MAX_N][MAX_M], int n) :
    Matrix(n, n) {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            setElementAt(i + 1, j + 1, data[i][j]);
}

TriangularMatrix::TriangularMatrix(const Matrix &arg) :
    Matrix(arg.getRows(), arg.getColumns()) {
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            setElementAt(i, j, arg.elementAt(i, j));
}

int TriangularMatrix::f(int i, int j) const {
    Matrix::f(i, j);
    return (i * (i - 1)) / 2 + j - 1;
}

double TriangularMatrix::elementAt(int i, int j) const {
    int k = f(i, j);
    double result = elements[k];
    return j <= i ? result : 0;
}

double TriangularMatrix::setElementAt(int i, int j, double value) {
    string msg("TriangularMatrix: Incorrect value!");
    int k = f(i, j);
    double oldValue = elements[k];
    if (j <= i)
        elements[k] = value;
    else if (value != 0)
        throw msg;
    return oldValue;
}

Matrix& TriangularMatrix::create(int rows, int cols) const {
    cout << "TriangularMatrix::create" << endl;
    return *(new TriangularMatrix(rows));
}

Matrix& TriangularMatrix::operator=(const Matrix &arg) {
    cout << "TriangularMatrix: Operator =" << endl;
    string msg("TriangularMatrix: Illegal matrix dimensions!");
    if (arg.getRows() != n || arg.getColumns() != m)
        throw msg;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= i; j++)
            setElementAt(i, j, arg.elementAt(i, j));
    return *this;
}

```

```
Matrix& TriangularMatrix::operator+(const Matrix &arg) const {
    cout << "TriangularMatrix: Operator +" << endl;
    Matrix &A = *(new OrdinaryMatrix(*this));
    Matrix &B = *(new OrdinaryMatrix(arg));
    return A + B;
}
```

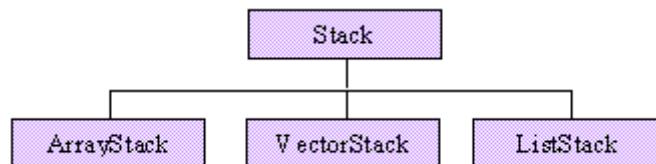
Функцията operator+ е реализирана така, че на текущата триъгълна матрица се присвояват елементите под и по главния диагонал на матрицата arg. Променете функцията operator+ така, че когато arg е от тип **TriangularMatrix** се събират триъгълни матрици и полученият резултат сащо е от тип **TriangularMatrix**.

4. Клас **SparseMatrix**. Реализацията са основава на представянето на ненулевите елементи a_{ij} на „разредена” матрица (матрица, в която повече от елементите са нули) $A(n \times m)$, $i \in [1; n]$, $j \in [1; m]$ чрез динамична структура (вектор, списък и др.)

$$\text{elements} = \{(i, j, a_{ij}) \mid a_{ij} \neq 0\}$$

и включва клас за представяне на тройка от вида (цяло число, цяло число, реално число).

Задача: Дадена е йерархията от класове:



Абстрактният клас **Stack** съдържа операции на *стек с елементи от тип T*, съгласно дадения интерфейс:

```
/*
 * Stack.h
 *
 * Created on: 11.2011
 *
 */

#ifndef STACK_H_
#define STACK_H_

#include "Type.h"
#include <cstddef>

class Stack {
public:
    virtual bool empty() const=0;
    // Returns whether the stack is empty, i.e. whether its size is 0.

    virtual size_t size() const=0;
    // Returns the number of elements in the stack.

    virtual T& top()=0;
    // Returns a reference to the next element in the stack.

    virtual void push(const T& x)=0;
    // Adds a new element at the top of the stack, above its current top element.

    virtual void pop()=0;
    // Removes the element on top of the stack.

    virtual ~Stack() {};
};
```

```
//Destructor  
};  
  
#endif /* STACK_H */
```

Да се реализират:

1. Клас **ArrayStack** – последователен стек с елементи от тип **T** с ограничен капацитет
2. Клас **VectorStack** - последователен стек с елементи от тип **T** без ограничение за капацитета
3. Клас **ListStack** свързан стек с елементи от тип **T** без ограничение за капацитета

Функции шаблони. Класове шаблони. Итератор. Линеен списък. Сортиран линеен списък. Полином на една променлива с реални коефициенти

Итератори в C++ и операции

Операция	Описание на операцията
Входен итератор (Input)	
*p ++p, p++ p1 == p2 p1 != p2	Четене на стойност на елемент, сочен от итератора p, например x = *p Нарастване на итератора p (префиксна и постфиксна форма) Сравняване на итераторите p1 и p2 за равенство Сравняване на итераторите p1 и p2 за неравенство
Изходен итератор (Output)	
*p ++p, p++	Запис на стойност на елемент, сочен от итератора p, например *p = x Нарастване на итератора p (префиксна и постфиксна форма)
Еднопосочен итератор (Forward)	
*p *p ++p, p++ p1 == p2 p1 != p2	Четене на стойност на елемент, сочен от итератора p, например x = *p Запис на стойност на елемент, сочен от итератора p, например *p = x Нарастване на итератора p (префиксна и постфиксна форма) Сравняване на итераторите p1 и p2 за равенство Сравняване на итераторите p1 и p2 за неравенство
Двупосочен итератор (Bidirectional)	
*p *p ++p, p++ --p, p-- p1 == p2 p1 != p2	Четене на стойност на елемент, сочен от итератора p, например x = *p Запис на стойност на елемент, сочен от итератора p, например *p = x Нарастване на итератора p (префиксна и постфиксна форма) Намаляване на итератора p (префиксна и постфиксна форма) Сравняване на итераторите p1 и p2 за равенство Сравняване на итераторите p1 и p2 за неравенство
Итератор с пряк достъп (RandomAccess)	
*p *p ++p, p++ --p, p-- p += i p -= i p + i p - i p1 < p2, p1 <= p2, p1 > p2, p1 >= p2 p[i]	Четене на стойност на елемент, сочен от итератора p, например x = *p Запис на стойност на елемент, сочен от итератора p, например *p = x Нарастване на итератора p (префиксна и постфиксна форма) Намаляване на итератора p (префиксна и постфиксна форма) Преместване на итератора p с i позиции напред Преместване на итератора p с i позиции назад Израз от тип итератор, чиято стойност е i позиции след p, без да премества итератор Израз от тип итератор, чиято стойност е i позиции преди p, без да премества итератор Сравняване на итератори Връща стойността на елемент, който се намира в позиция p + i

На адрес <http://www.cplusplus.com/reference/std/iterator/> може да намерите информация за итераторите в стандартните библиотеки на C++.

Задача: Да се реализира АТД **List** – линеен списък с елементи от тип **T** и операции, съгласно дадената спецификация – виж <http://www.cplusplus.com/reference/stl/list/>:

Member functions

(constructor)	Construct list (public member function)
(destructor)	List destructor (public member function)
operator=	Copy container content (public member function)

Iterators:

begin	Return iterator to beginning (public member function)
-----------------------	---

<u>end</u>	Return iterator to end (public member function)
<u>rbegin</u>	Return reverse iterator to reverse beginning (public member function)
<u>rend</u>	Return reverse iterator to reverse end (public member function)

Capacity:

<u>empty</u>	Test whether container is empty (public member function)
<u>size</u>	Return size (public member function)
<u>max_size</u>	Return maximum size (public member function)
<u>resize</u>	Change size (public member function)

Element access:

<u>front</u>	Access first element (public member function)
<u>back</u>	Access last element (public member function)

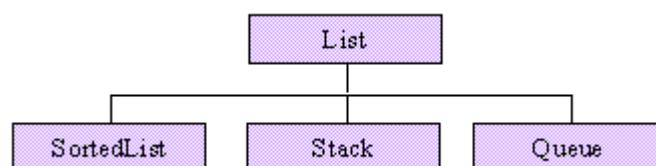
Modifiers:

<u>assign</u>	Assign new content to container (public member function)
<u>push_front</u>	Insert element at beginning (public member function)
<u>pop_front</u>	Delete first element (public member function)
<u>push_back</u>	Add element at the end (public member function)
<u>pop_back</u>	Delete last element (public member function)
<u>insert</u>	Insert elements (public member function)
<u>erase</u>	Erase elements (public member function)
<u>swap</u>	Swap content (public member function)
<u>clear</u>	Clear content (public member function)

Operations:

<u>splice</u>	Move elements from list to list (public member function)
<u>remove</u>	Remove elements with specific value (public member function)
<u>remove_if</u>	Remove elements fulfilling condition (public member function template)
<u>unique</u>	Remove duplicate values (member function)
<u>merge</u>	Merge sorted lists (public member function)
<u>sort</u>	Sort elements in container (public member function)
<u>reverse</u>	Reverse the order of elements (public member function)

ПРОЕКТ: Йерархия от класове за реализация и използване на списък:



ИНТЕРФЕЙСИ

Дефиниция на типа **T** на елементите на списък:

```
/*
 * type.h
 *
 * Created on: 11.2011
 */
```

*/

```
#ifndef TYPE_H_
#define TYPE_H_

typedef int T;

#endif /* TYPE_H_ */
```

Интерфейс на класа List:

```
/*
 * List.h
 *
 * Created on: 11.2011
 */
#ifndef LIST_H_
#define LIST_H_

#include "type.h"
#include <cstddef>
#include <iostream>
using namespace std;

class List {
    friend ostream& operator<<(ostream&, const List&);

protected:
    class Node {
public:
    T data;
    Node *previous;
    Node *next;
    Node(T d = T(), Node *p = 0, Node *n = 0) :
        data(d) {
        previous = p;
        next = n;
    }
};

Node *first; //first element
Node *last; //last element
size_t sz; //elements count

public:
    //Bidirectional list iterator interface
    class Iterator {
        friend class List;
private:
        Node *current;
        Node *first;
        Node *last;
public:
        Iterator();
        Iterator& operator=(const Iterator& x);
        T& operator*() const;
        //const T& operator*() const; //const_iterator
        Iterator& operator++();
        Iterator& operator++(int);
        Iterator& operator--();
        Iterator& operator--(int);
        bool operator==(const Iterator& x) const;
        bool operator!=(const Iterator& x) const;
    };
    //Constructors
```

```

List();
//Default constructor: constructs an empty list, with no content and a
//size of zero.

List(const List& x);
//Copy constructor: The list is initialized to have the same contents
//(copies) as list x.

List& operator=(const List& x);
//Assigns a copy of list x as the new content for the list object. The
//elements contained in the list object before the call are dropped, and
//replaced by copies of those in list x, if any. After a call to this
//member function, both the list object and list x will have the same
//size and compare equal to each other.

//Iterator
Iterator begin() const;
//Returns a bidirectional iterator referring to the first element in the
//list container.

Iterator end() const;
//Returns a bidirectional iterator referring to the past-the-end element
//in the list container.

Iterator rbegin() const;
//Returns a bidirectional iterator referring to the last element in the list
//container. rbegin refers to the element right before the one that would be
//referred to by member end.

Iterator rend() const;
//Returns a bidirectional iterator referring to the element right before the
//first element in the list container, which is considered its reverse end.

//Capacity:
size_t size() const;
//Returns the number of elements in the list.

bool empty() const;
//Returns whether the list is empty, i.e. whether its size is 0.

//Element access:
T& front();
//Returns a reference to the first element in the list.

T& back();
//Returns a reference to the last element in the list.

//Modifiers:
void assign(size_t n, const T& u);
//Assigns new content to the list object, dropping all the elements
//contained in the list before the call and replacing them by those
//specified by the parameters: The new content is the repetition n times of
//copies of element u.

void push_front(const T& x);
//Adds a new element at the beginning of the list, before its current first
//element. The content of this new element is initialized to a copy of x.
//This effectively increases the list size by one.

void pop_front();
//Removes the beginning element in the list, effectively reducing the list
//size by one. This calls the removed element's destructor.

void push_back(const T& x);
//Adds a new element at the end of the list, after its current last
//element. The content of this new element is initialized to a copy of x.

```

```

//This effectively increases the list size by one.

void pop_back();
//Removes the last element in the list, effectively reducing the list
//size by one. This calls the removed element's destructor.

void insert(Iterator position, const T& x);
template<class InputIterator>
void insert(Iterator position, InputIterator first, InputIterator last);
//The list container is extended by inserting either a single element
//or a range of elements [first,last) before the element at position.
//This effectively increases the container size by the amount
//of elements inserted.

Iterator erase(Iterator position);
Iterator erase(Iterator first, Iterator last);
//Removes from the list container either a single element (position) or
//a range of elements ([first,last)). This effectively reduces the list
//size by the number of elements removed, calling each element's
//destructor before.

void clear();
//All the elements of the list are dropped: their destructors are called,
//and then they are removed from the list, leaving the list with a size of 0.

//Operations
void splice(Iterator position, List& x);
void splice(Iterator position, Iterator first, Iterator last);
//Moves all elements from list x or moves the elements in the
//range [first,last) into the list container at the specified
//position, effectively inserting the specified elements into
//the container and removing them from x. This increases the
//container size by the amount of elements inserted, and
//reduces the size of x by the same amount ( whenever x is not
//the same as *this ).

void remove(const T& value);
//Removes from the list all the elements with a specific value
//using ==. This calls the destructor of these objects and
//reduces the list size by the amount of elements removed.

template<class Predicate>
void remove_if(Predicate pred);
//Removes from the list all the elements for which Predicate
//pred returns true. This calls the destructor of these objects
//and reduces the list size by the amount of elements removed.

void unique();
//Removes all but the first element from every consecutive group
//of equal (using ==) elements in the list container.

template<class BinaryPredicate>
void unique(BinaryPredicate binary_pred);
//Template function, accepting a binary predicate, a specific
//comparison function to determine the "uniqueness" of an element
//can be specified.

void merge(List& x);
//Merges x into the list, inserting all the elements of x into
//the list object at their respective ordered positions.
//This empties x and increases the list size.

template<class Compare>
void merge(List& x, Compare comp);
//The second version (template function), has the same behavior,
//but takes a specific function to perform the comparison

```

```

//operation in charge of determining the insertion points.
//The comparison function has to perform weak strict ordering
//(which basically means the comparison operation has to be
//transitive and irreflexive).

void sort();
template<class Compare>
void sort(Compare comp);
//Sorts the elements in the container from lower to higher.
//The sorting is performed by comparing the elements in the
//container in pairs using a sorting algorithm. In the first
//version, taking no parameters, the comparisons are performed
//using the operator< between the elements being compared.
//In the second version, the comparisons are performed using
//function comp, which performs weak strict ordering
//(this basically means the comparison operation has to be
//transitive and irreflexive).

//Destructor
~List();
//Destroys the list object. This calls each of the contained element's
//destructors, and deallocates all the storage capacity allocated by the
//list.
};

#include "ListTemplateFunctionImp.h"

#endif /* LIST_H_ */

```

Интерфейс на класа SortedList:

```

/*
 * SortedList.h
 *
 * Created on: 11.2011
 *
 */

#ifndef SORTEDLIST_H_
#define SORTEDLIST_H_

#include "List.h"

class SortedList: public List {
public:
    SortedList() :
        List() {
    }
    template<class Compare>
    void push_in_sortedList(Compare comp, const T& x);
};

#include "SortedListTemplateFunctionImp.h"

#endif /* SORTEDLIST_H_ */

```

TECTВАНЕ

Интерфейс на класа Tester:

```

/*
 * Tester.h
 *
 * Created on: 11.2011
 */

```

```
/*
#ifndef TESTER_H_
#define TESTER_H_

#include "List.h"
#include "SortedList.h"

class Tester {
private:
    void test1();
    void test2();
    void test3();
    void test4();
    void test5();
    void test6();
public:
    void start();
};

#endif /* TESTER_H_ */
```

Реализация на класа Tester:

```
/*
 * Tester.cpp
 *
 * Created on: 11.2011
 *
 */

#include "Tester.h"
#include <cstdlib>
#include <ctime>

void Tester::start() {
    cout << "      Test 1" << endl;
    test1();

    cout << "      Test 2" << endl;
    test2();

    cout << "      Test 3" << endl;
    test3();

    cout << "      Test 4" << endl;
    test4();

    cout << "      Test 5" << endl;
    test5();

    cout << "      Test 6" << endl;
    test6();
}

//=====TEST 1=====
void Tester::test1() {
    cout << "Start..." << endl;
    List list;
    List::Iterator it;
    int n;

    cout << "\n      Inserting head:" << endl;
    srand(time(0));
    for (int i = 0; i < 5; i++) {
        n = rand();
```

```

cout << "push_front " << n << endl;
list.push_front(n);
}

cout << "\n      Inserting tail:" << endl;
strand(n);
for (int i = 0; i < 5; i++) {
    n = rand();
    cout << "push_back " << n << endl;
    list.push_back(n);
}

cout << "\n      Iterator: head to tail print..." << endl;
for (it = list.begin(); it != list.end(); it++)
    cout << *it << " ";

cout << "\n\n      Removing head:" << endl;
for (int i = 0; i < 5; i++) {
    cout << "pop_front " << list.front() << endl;
    list.pop_front();
}

cout << "\n      Iterator: tail to head print..." << endl;
for (it = list.rbegin(); it != list.rend(); it--)
    cout << *it << " ";

cout << "\n\n      Iterator: change list *it = -100..." << endl;
for (it = list.rbegin(); it != list.rend(); it--)
    *it = -100;

cout << "\n      Iterator: tail to head print..." << endl;
for (it = list.rbegin(); it != list.rend(); it--)
    cout << *it << " ";

cout << "\n\n      Removing tail:" << endl;
while (!list.empty()) {
    cout << "pop_back " << list.back() << endl;
    list.pop_back();
}

cout << "\nDone!" << endl;
}

//====================================================================TEST 2=====
void Tester::test2() {
    List list;
    List::Iterator it;
    int n;

    cout << "\n      Inserting tail:" << endl;
    strand(n);
    for (int i = 0; i < 5; i++) {
        n = rand();
        cout << "push_back " << n << endl;
        list.push_back(n);
    }

    cout << "\n      Iterator: head to tail print..." << endl;
    for (it = list.begin(); it != list.end(); it++)
        cout << *it << " ";

    cout << "\n\n      list.begin(); ++it; insert(it,-100);" << endl;
    it = list.begin();
    ++it;
    list.insert(it, -100);
}

```

```

cout << "\n      Iterator: head to tail print..." << endl;
for (it = list.begin(); it != list.end(); it++)
    cout << *it << " ";

cout << "\n\n      list.rbegin(); --it; --it; erase(it)" << endl;
it = list.rbegin();
--it;
--it;
list.erase(it);

cout << "\n      Iterator: head to tail print..." << endl;
for (it = list.begin(); it != list.end(); it++)
    cout << *it << " ";

cout << "\n\nDone!" << endl;
}

//=====TEST 3=====
void Tester::test3() {
    List list1, list2;
    List::Iterator it;
    int n;

    cout << "\n      Inserting tail:" << endl;
    srand(n);
    for (int i = 0; i < 5; i++) {
        n = rand();
        cout << "push_back " << n << endl;
        list1.push_back(n);
    }

    cout << "\n      Iterator: list1 head to tail print..." << endl;
    for (it = list1.begin(); it != list1.end(); it++)
        cout << *it << " ";

    cout << "\n\n      Inserting tail:" << endl;
    srand(n);
    for (int i = 0; i < 5; i++) {
        n = rand();
        cout << "push_back " << n << endl;
        list2.push_back(n);
    }

    cout << "\n      Iterator: list2 head to tail print..." << endl;
    for (it = list2.begin(); it != list2.end(); it++)
        cout << *it << " ";

    cout << "\n\n      list1.begin(); ++it; ++it; list1.splice(it, list2);"
         << endl;
    it = list1.begin();
    ++it;
    ++it;
    list1.splice(it, list2);

    cout << "\n      Iterator: list1 head to tail print..." << endl;
    for (it = list1.begin(); it != list1.end(); it++)
        cout << *it << " ";

    cout << "\n\nDone!" << endl;
}

//=====TEST 4=====
#include <cmath>

class IntBinPred {
public:

```

```

bool operator()(int first, int second) {
    return fabs(first) == fabs(second);
};

void Tester::test4() {
    List list;
    List::Iterator it;
    int n;

    cout << "\n      Inserting tail:" << endl;
    srand(n);
    for (int i = 0; i < 5; i++) {
        n = rand();
        cout << "push_back " << n << endl;
        list.push_back(n);
        cout << "push_back " << -n << endl;
        list.push_back(-n);
    }

    cout << "\n      Iterator: head to tail print..." << endl;
    for (it = list.begin(); it != list.end(); it++)
        cout << *it << " ";

    cout << "\n\n      list.unique(IntBinPred());" << endl;
    list.unique(IntBinPred());

    cout << "\n      Iterator: head to tail print..." << endl;
    for (it = list.begin(); it != list.end(); it++)
        cout << *it << " ";

    cout << "\n\nDone!" << endl;
}

//====================================================================TEST 5=====
class IntComp {
public:
    bool operator()(int first, int second) {
        return first <= second;
    }
};

void Tester::test5() {
    List list;
    List::Iterator it;
    int n;

    cout << "\n      Inserting tail:" << endl;
    srand(n);
    for (int i = 0; i < 5; i++) {
        n = rand();
        cout << "push_back " << n << endl;
        list.push_back(n);
        cout << "push_back " << -n << endl;
        list.push_back(-n);
    }

    cout << "\n      Iterator: head to tail print..." << endl;
    for (it = list.begin(); it != list.end(); it++)
        cout << *it << " ";

    cout << "\n\n      list.sort(IntComp());" << endl;
    list.sort(IntComp());

    cout << "\n      Iterator: head to tail print..." << endl;
    for (it = list.begin(); it != list.end(); it++)

```

```

cout << *it << " ";
cout << "\n\nDone!" << endl;
}

//=====================================================================
void Tester::test6() {
    SortedList list;
    List::Iterator it;
    int n;

    cout << "\n      Inserting:" << endl;
    strand(n);
    for (int i = 0; i < 5; i++) {
        n = rand();
        cout << "push_in_sortedList " << n << endl;
        list.push_in_sortedList(IntComp(), n);
        cout << "push_in_sortedList " << -n << endl;
        list.push_in_sortedList(IntComp(), -n);
    }

    cout << "\n      Iterator: head to tail print..." << endl;
    for (it = list.begin(); it != list.end(); it++)
        cout << *it << " ";
    cout << "\n\nDone!" << endl;
}

```

Резултати от тестването:

```

//=====================================================================
// Name       : Lists.cpp
// Version    : 11.2011
// Description : Lists
//=====================================================================

#include "Tester.h"
#include <iostream>
using namespace std;

int main() {
    Tester().start();
    return 0;
}

Test 1
Start...

      Inserting head:
push_front 3918
push_front 5840
push_front 11643
push_front 2006
push_front 27611

      Inserting tail:
push_back 24668
push_back 2192
push_back 12172
push_back 10624
push_back 17881

      Iterator: head to tail print...
27611 2006 11643 5840 3918 24668 2192 12172 10624 17881

```

```
    Removing head:  
pop_front 27611  
pop_front 2006  
pop_front 11643  
pop_front 5840  
pop_front 3918  
  
    Iterator: tail to head print...  
17881 10624 12172 2192 24668  
  
    Iterator: change list *it = -100...  
  
    Iterator: tail to head print...  
-100 -100 -100 -100 -100  
  
    Removing tail:  
pop_back -100  
pop_back -100  
pop_back -100  
pop_back -100  
pop_back -100  
  
Done!  
Test 2  
  
    Inserting tail:  
push_back 18214  
push_back 19568  
push_back 12743  
push_back 21903  
push_back 5247  
  
    Iterator: head to tail print...  
18214 19568 12743 21903 5247  
  
    list.begin(); ++it; insert(it,-100);  
  
    Iterator: head to tail print...  
18214 -100 19568 12743 21903 5247  
  
    list.rbegin(); --it; --it; erase(it)  
  
    Iterator: head to tail print...  
18214 -100 19568 21903 5247  
  
Done!  
Test 3  
  
    Inserting tail:  
push_back 22361  
push_back 13613  
push_back 12297  
push_back 9177  
push_back 30446  
  
    Iterator: list1 head to tail print...  
22361 13613 12297 9177 30446  
  
    Inserting tail:  
push_back 1158  
push_back 32452  
push_back 30538  
push_back 7143  
push_back 30644  
  
    Iterator: list2 head to tail print...
```

```
1158 32452 30538 7143 30644
```

```
list1.begin(); ++it; ++it; list1.splice(it, list2);
```

```
Iterator: list1 head to tail print...
22361 13613 1158 32452 30538 7143 30644 12297 9177 30446
```

Done!

Test 4

Inserting tail:

```
push_back 29207
push_back -29207
push_back 24142
push_back -24142
push_back 11138
push_back -11138
push_back 25611
push_back -25611
push_back 6391
push_back -6391
```

Iterator: head to tail print...

```
29207 -29207 24142 -24142 11138 -11138 25611 -25611 6391 -6391
```

```
list.unique(IntBinPred());
```

Iterator: head to tail print...

```
29207 24142 11138 25611 6391
```

Done!

Test 5

Inserting tail:

```
push_back 29207
push_back -29207
push_back 24142
push_back -24142
push_back 11138
push_back -11138
push_back 25611
push_back -25611
push_back 6391
push_back -6391
```

Iterator: head to tail print...

```
29207 -29207 24142 -24142 11138 -11138 25611 -25611 6391 -6391
```

```
list.sort(IntComp());
```

Iterator: head to tail print...

```
-29207 -25611 -24142 -11138 -6391 6391 11138 24142 25611 29207
```

Done!

Test 6

Inserting:

```
push_in_sortedList 35
push_in_sortedList -35
push_in_sortedList 29739
push_in_sortedList -29739
push_in_sortedList 3374
push_in_sortedList -3374
push_in_sortedList 11141
push_in_sortedList -11141
push_in_sortedList 31308
```

```
push_in_sortedList -31308
    Iterator: head to tail print...
-31308 -29739 -11141 -3374 -35 35 3374 11141 29739 31308
Done!
```

РЕАЛИЗАЦИЯ

Реализация на класа List:

```
/*
 * List.cpp
 *
 * Created on: 11.2011
 *
 */

#include "List.h"

// Реализация на функциите на класа Iterator на класа List

List::Iterator::Iterator() {
}

List::Iterator& List::Iterator::operator=(const List::Iterator& x) {
    current = x.current;
    last = x.last;
    return *this;
}

T& List::Iterator::operator*() const {
    return current->data;
}

List::Iterator& List::Iterator::operator++() {
    if (current != 0)
        current = current->next;
    return *this;
}

List::Iterator& List::Iterator::operator++(int) {
    List::Iterator *result = new List::Iterator();
    *result = *this;
    ++(*this);
    return *result;
}

List::Iterator& List::Iterator::operator--() {
    if (current != 0)
        current = current->previous;
    return *this;
}

List::Iterator& List::Iterator::operator--(int) {
    List::Iterator *result = new List::Iterator();
    *result = *this;
    --(*this);
    return *result;
}

bool List::Iterator::operator==(const List::Iterator& x) const {
    return current == x.current;
}
```

```

bool List::Iterator::operator!=(const List::Iterator& x) const {
    return current != x.current;
}

// Реализация на функциите на класа List, които не са шаблони

List::List() {
    first = last = 0;
    sz = 0;
}

List::Iterator List::begin() const {
    List::Iterator it;
    it.current = first;
    it.last = last;
    return it;
}

List::Iterator List::end() const {
    List::Iterator it;
    it.current = 0;
    it.last = last;
    return it;
}

List::Iterator List::rbegin() const {
    List::Iterator it;
    it.current = last;
    it.first = first;
    return it;
}

List::Iterator List::rend() const {
    List::Iterator it;
    it.current = 0;
    it.first = first;
    return it;
}

size_t List::size() const {
    return sz;
}

bool List::empty() const {
    return sz == 0;
}

T& List::front() {
    return first->data;
}

T& List::back() {
    return last->data;
}

void List::push_back(const T& x) {
    Node *newNode = new Node(x);
    sz++;
    if (first == 0) {
        first = newNode;
        last = newNode;
    } else {
        newNode->previous = last;
        last ->next = newNode;
        last = newNode;
    }
}

```

```

    }

}

void List::pop_back() {
    if (first == 0)
        return;
    else if (first->next == 0) {
        delete first;
        first = last = 0;
        sz = 0;
    } else {
        Node *remove = last;
        last->previous->next = 0;
        last = last->previous;
        delete remove;
        sz--;
    }
}

void List::push_front(const T& x) {
    Node *newNode = new Node(x);
    sz++;
    if (first == 0) {
        first = newNode;
        last = newNode;
    } else {
        newNode->next = first;
        first->previous = newNode;
        first = newNode;
    }
}

void List::pop_front() {
    if (first == 0)
        return;
    else if (first->next == 0) {
        delete first;
        first = last = 0;
        sz = 0;
    } else {
        Node *remove = first;
        first->next->previous = 0;
        first = first->next;
        delete remove;
        sz--;
    }
}

void List::insert(List::Iterator position, const T& x) {
    if (position == end())
        push_back(x);
    else if (position == rend())
        push_front(x);
    else {
        Node *after = position.current;
        Node *before = after->previous;
        Node *newNode = new Node(x);
        newNode->previous = before;
        newNode->next = after;
        after->previous = newNode;
        if (before == 0)
            first = newNode;
        else
            before->next = newNode;
    }
}

```

```

}

List::Iterator List::erase(List::Iterator position) {
    if (position == end() || position == rend())
        return position;
    List::Iterator it = position;
    it++;
    Node* remove = position.current;
    Node* before = remove->previous;
    Node* after = remove->next;
    if (remove == first) {
        first = after;
    } else {
        before->next = after;
    }
    if (remove == last) {
        last = before;
    } else {
        after->previous = before;
    }
    delete remove;
    return it;
}

void List::clear() {
    while (!empty())
        pop_front();
}

void List::splice(List::Iterator position, List& x) {
    if (this != &x)
        while (!x.empty()) {
            insert(position, x.front());
            x.pop_front();
        }
}

List::~List() {
    clear();
}

ostream& operator<<(ostream& s, const List& x) {
    for (List::Iterator it = x.begin(); it != x.end(); ++it)
        s << *it << endl;
    return s;
}

```

Реализация на функциите шаблони на класа List:

```

/*
 * ListTemplateFunctionImp.h
 *
 * Created on: 11.2011
 *
 */

#ifndef TEMPLATEFUNCTIONIMP_H_
#define TEMPLATEFUNCTIONIMP_H_

template<class BinaryPredicate>
void List::unique(BinaryPredicate binary_pred) {
    List::Iterator it1, it2;
    for (it1 = begin(); it1 != end(); ++it1) {
        it2 = it1;
        ++it2;
        for (; it2 != end(); ++it2)

```

```

        if (binary_pred(*it1, *it2))
            it2 = erase(it2);
    }

template<class Compare>
void List::sort(Compare comp) {
    List::Iterator it1, it2, it3;
    bool flag = true;
    while (flag) {
        flag = false;
        it1 = it2 = begin();
        ++it2;
        for (; it2 != end(); ++it2) {
            if (!comp(*it1, *it2)) {
                flag = true;
                T temp = *it1;
                *it1 = *it2;
                *it2 = temp;
            }
            ++it1;
        }
    }
}

#endif /* TEMPLATEFUNCTIONIMP_H_ */

```

Реализация на функцията шаблон на класа *SortedList*:

```

/*
 * SortedListTemplateFunctionImp.h
 *
 * Created on: 11.2011
 *
 */

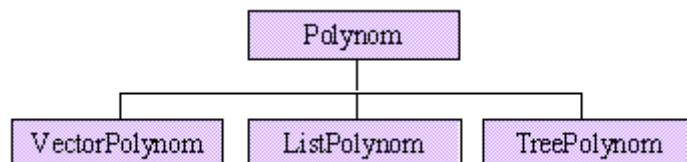
#ifndef SORTEDLISTTEMPLATEFUNCTIONIMP_H_
#define SORTEDLISTTEMPLATEFUNCTIONIMP_H_

template<class Compare>
void SortedList::push_in_sortedList(Compare comp, const T& x) {
    List::Iterator it;
    for (it = begin(); it != end(); ++it)
        if (comp(x, *it))
            break;
    insert(it, x);
}

#endif /* SORTEDLISTTEMPLATEFUNCTIONIMP_H_ */

```

Задача: Дадена е йерархията от класове:



Да се реализират:

1. Абстрактен клас **Polynom**, който представя операции на *полином на една променлива с коефициенти реални числа*
2. Клас **ListPolynom** - представяне на коефициентите на полином $P(x) = a_n x^n + \dots + a_0$ чрез списък **coeffs** = $\{(i, a_i) \mid a_i \neq 0\}$

ИНТЕРФЕЙСИ

*Интерфейс на абстрактния клас **Polynom**:*

```
/*
 * Polynom.h
 *
 * Created on: 11.2011
 *
 */

#ifndef POLYNOM_H_
#define POLYNOM_H_

class Polynom {
protected:
    int n; // max degree

    virtual Polynom& create() const=0;
    //Създава нулев полином
public:
    //Constructor
    Polynom(int = 0);

    //Abstract methods
    virtual double coeff(int degree) const=0;
    virtual double setCoeff(int degree, double value)=0;

    //Public methods
    int degree() const;
    //virtual Polynom& operator=(const Polynom &arg);
    virtual Polynom& operator+(const Polynom &arg) const;
    //virtual Polynom& operator*(const Polynom &arg) const;
    //virtual Polynom& operator*=(double value);
    //virtual Polynom& derive() const;
    virtual double operator()(double value) const; //P(value)

    //Destructor
    virtual ~Polynom();
};

#endif /* POLYNOM_H_ */
```

*Интерфейс на класа **ListPolynom**:* Реализацията на класа **ListPolynom** включва клас **Pair** за представяне на двойка от вида (цияло число, реално число) и използва наличният клас шаблон list - <http://www.cplusplus.com/reference/stl/list/>

```
/*
 * ListPolynom.h
 *
 * Created on: 11.2011
 *
 */

#ifndef LISTPOLYNOM_H_
#define LISTPOLYNOM_H_

#include "Polynom.h"
#include <list>
using namespace std;

class ListPolynom: public Polynom {
private:
    class Pair {
```

```

public:
    int n;
    double coeff;
public:
    Pair(int d, double c) {
        n = d;
        coeff = c;
    }
};

list<Pair> coeffs; //empty list
protected:
    Polynom& create() const;
public:
    //Constructors
    ListPolynom(int = 0);
    ListPolynom(double data[], int n);
    ListPolynom(const Polynom &arg);

    //Public methods
    double coeff(int degree) const;
    double setCoeff(int degree, double value);

    //Public methods - inherited from Polynom
};

#endif /* LISTPOLYNOM_H_ */

```

TECTBAHE

```

//=====================================================================
// Name      : PolynomPrj.cpp
// Version   : 11.2011
// Description : Polynom tests
//=====================================================================

#include "ListPolynom.h"
#include <iostream>
using namespace std;

const int MAX_N = 3;

void show(Polynom &A);

int main() {
    double d[MAX_N] = { 1, 2, 3 };
    ListPolynom *x = new ListPolynom(d, 2);
    Polynom &P = *x;
    show(P);
    cout << "\nP(-1)=" << P(-1) << endl;
    delete x;

    return 0;
}

void show(Polynom &A) {
    cout << "P(x)=" << A.coeff(0);
    for (int i = 1; i <= A.degree(); i++)
        cout << "+" << A.coeff(i) << "x^" << i;
}

```

Резултати от тестването:

P(x)=1+2x^1+3x^2
P(-1)=2

РЕАЛИЗАЦИЯ

Реализация на класа Polynom:

```
/*
 * Polynom.cpp
 *
 * Created on: 11.2011
 *
 */

#include "Polynom.h"

Polynom::Polynom(int x) {
    n = x;
}

int Polynom::degree() const {
    return n;
}

Polynom& Polynom::operator+(const Polynom &arg) const {
    Polynom &result = create();
    int maxDegree = degree() < arg.degree() ? arg.degree() : degree();
    for (int i = 0; i <= maxDegree; i++) {
        result.setCoeff(i, coeff(i) + arg.coeff(i));
    }
    return result;
}

double Polynom::operator()(double value) const {
    double result = 0;
    double p = 1;
    for (int i = 0; i <= degree(); i++) {
        result += coeff(i) * p;
        p *= value;
    }
    return result;
}
Polynom::~Polynom() {
}
```

Реализация на класа ListPolynom:

```
/*
 * ListPolynom.cpp
 *
 * Created on: 11.2011
 *
 */

#include "ListPolynom.h"

ListPolynom::ListPolynom(int n) :
    Polynom(n) {}

ListPolynom::ListPolynom(double data[], int n) :
    Polynom(n) {
    for (int i = 0; i <= n; i++)
        setCoeff(i, data[i]);
}

ListPolynom::ListPolynom(const Polynom &arg) :
    Polynom(arg.degree()) {
```

```

for (int i = 0; i <= n; i++)
    setCoeff(i, arg.coeff(i));
}

double ListPolynom::coeff(int degree) const {
    list<Pair>::const_iterator it;
    for (it = coeffs.begin(); it != coeffs.end(); ++it) {
        if ((*it).n == degree)
            return (*it).coeff;
    }
    return 0;
}

double ListPolynom::setCoeff(int degree, double value) {
    double result = 0;
    list<Pair>::iterator it;
    for (it = coeffs.begin(); it != coeffs.end(); ++it) {
        if ((*it).n == degree) {
            result = (*it).coeff;
            if (value != 0)
                (*it).coeff = value;
            else
                coeffs.erase(it);
            return result;
        }
    }
    if (value != 0)
        coeffs.push_front(Pair(degree, value));
    return result;
}

Polynom& ListPolynom::create() const {
    return *(new ListPolynom());
}

```

Обобщено програмиране. Приоритетна опашка. Асоциативен масив. Множество

ПРОЕКТ Templates:**1. Реализация на клас шаблон****Пример:** Клас шаблон **Vector**:*Интерфейс на класа шаблон Vector:*

```

/*
 * Vector
 *
 * Created on: 09.2011
 *
 */

#ifndef VECTOR_
#define VECTOR_


#include <cstddef>

template<typename T, int initialCapacity = 1>
class Vector {
private:
    T* data;
    size_t c; //capacity
    size_t sz; //elements count
public:
    Vector();
    Vector(size_t n, const T& value = T());
    Vector(const Vector<T, initialCapacity>& x);
    Vector<T, initialCapacity>& operator=(const Vector<T, initialCapacity>& x);
    ~Vector();
    size_t size() const;
    size_t max_size() const;
    void resize(size_t sz, const T& c);
    size_t capacity() const;
    bool empty() const;
    T& operator[](size_t n);
    T& at(size_t n);
    T& front();
    T& back();
    void assign(size_t n, const T& u);
    void push_back(const T& x);
    void pop_back();
    void insert(size_t position, size_t n, const T& x);
    void erase(size_t position);
    void erase(size_t first, size_t last);
    void clear();
};

#include "VectorImp.h"

#endif /* VECTOR_ */

```

Реализация на класа шаблон Vector:

```

/*
 * VectorImp.h
 *
 * Created on: 09.2011
 *

```

```

*/
```

```

#ifndef VECTORIMP_H_
#define VECTORIMP_H_
```

```

#include <string>
#include <exception>
using namespace std;
```

```

template<typename T, int initialCapacity>
Vector<T, initialCapacity>::Vector() {...}
```

```

template<typename T, int initialCapacity>
Vector<T, initialCapacity>::Vector(size_t n, const T& value) {...}
```

```

template<typename T, int initialCapacity>
Vector<T, initialCapacity>::Vector(const Vector<T, initialCapacity>& x) {...}
```

```

template<typename T, int initialCapacity>
Vector<T, initialCapacity>::operator=(const Vector<T, initialCapacity>& x) {...}
```

```

template<typename T, int initialCapacity>
Vector<T, initialCapacity>::~Vector() {...}
```

```

template<typename T, int initialCapacity>
size_t Vector<T, initialCapacity>::size() const {...}
```

```

template<typename T, int initialCapacity>
size_t Vector<T, initialCapacity>::max_size() const {...}
```

```

template<typename T, int initialCapacity>
void Vector<T, initialCapacity>::resize(size_t sz, const T& c) {...}
```

```

template<typename T, int initialCapacity>
size_t Vector<T, initialCapacity>::capacity() const {...}
```

```

template<typename T, int initialCapacity>
bool Vector<T, initialCapacity>::empty() const {...}
```

```

template<typename T, int initialCapacity>
T& Vector<T, initialCapacity>::operator[](size_t n) {...}
```

```

template<typename T, int initialCapacity>
T& Vector<T, initialCapacity>::at(size_t n) {...}
```

```

template<typename T, int initialCapacity>
T& Vector<T, initialCapacity>::front() {...}
```

```

template<typename T, int initialCapacity>
T& Vector<T, initialCapacity>::back() {...}
```

```

template<typename T, int initialCapacity>
void Vector<T, initialCapacity>::assign(size_t n, const T& u) {...}
```

```

template<typename T, int initialCapacity>
void Vector<T, initialCapacity>::push_back(const T& x) {...}
```

```

template<typename T, int initialCapacity>
void Vector<T, initialCapacity>::pop_back() {...}
```

```

template<typename T, int initialCapacity>
void Vector<T, initialCapacity>::insert(size_t position, size_t n, const T& x)
{...}
```

```

template<typename T, int initialCapacity>
```

```

void Vector<T, initialCapacity>::erase(size_t position) {...}

template<typename T, int initialCapacity>
void Vector<T, initialCapacity>::erase(size_t first, size_t last) {...}

template<typename T, int initialCapacity>
void Vector<T, initialCapacity>::clear() {...}

#endif /* VECTORIMP_H_ */

```

2. Наследяване на клас шаблон от обикновен клас

Пример: Определяне на броя на съществуващите обекти на даден клас:

Реализация на класа шаблон Countable:

```

/*
 * Countable.h
 *
 * Created on: 12.2011
 *
 */

#ifndef COUNTABLE_H_
#define COUNTABLE_H_

#include <cstddef>

template<class T>
class Countable {
private:
    static size_t counter;
public:
    static size_t GetInstanceCount() {
        return counter;
    }

    Countable() {
        ++counter;
    }

    ~Countable() {
        --counter;
    }
};

size_t Countable<T>::counter = 0;

#endif /* COUNTABLE_H_ */

```

a) Реализация на класа **SomeClassCountable**, наследник на класа шаблон **Countable**:

```

/*
 * SomeClassCountable.h
 *
 * Created on: 12.2011
 *
 */

#ifndef SOMECLASSCOUNTABLE_H_
#define SOMECLASSCOUNTABLE_H_

#include "Countable.h"

```

```
class SomeClassCountable: public Countable<SomeClassCountable> {
public:
    //...
};

#endif /* SOMECLASSCOUNTABLE_H_ */
```

б) Реализация на класа **VectorOfIntsCountable**, наследник на класа шаблон **Countable** и на класа шаблон **Vector**:

```
/*
 * VectorOfIntsCountable.h
 *
 * Created on: 12.2011
 *
 */

#ifndef VECTOROFINTSCOUNTABLE_H_
#define VECTOROFINTSCOUNTABLE_H_

#include "Vector.h"
#include "Countable.h"

class VectorOfIntsCountable: public Vector<int> , public Countable<
    VectorOfIntsCountable> {

};

#endif /* VECTOROFINTSCOUNTABLE_H_ */
```

в) Реализация на класа шаблон **VectorCountable**, наследник на класа шаблон **Countable** и на класа шаблон **Vector**:

```
/*
 * VectorCountable.h
 *
 * Created on: 12.2011
 *
 */

#ifndef VECTORCOUNTABLE_H_
#define VECTORCOUNTABLE_H_

#include "Vector.h"
#include "Countable.h"

template<typename T, int initialCapacity = 1>
class VectorCountable: public Vector<T, initialCapacity> , public Countable<
    VectorCountable<T, initialCapacity> > {

};

#endif /* VECTORCOUNTABLE_H_ */
```

Тестване:

```
/*
 * Example1.h
 *
 * Created on: 12.2011
 *
 */

#ifndef EXAMPLE1_H_
```

```

#define EXAMPLE1_H_
#include "VectorCountable.h"

class Example1 {
public:
    void start();
};

#endif /* EXAMPLE1_H_ */

/*
 * Example1.cpp
 *
 * Created on: 12.2011
 */
#include "Example1.h"
#include <iostream>
using namespace std;

void Example1::start() {
    cout << "Start..." << endl;
    VectorCountable<int> someObject;
    cout << VectorCountable<int>::GetInstanceCount() << "\n";
{
    VectorCountable<int> someOtherObject;
    cout << VectorCountable<int>::GetInstanceCount() << "\n";
}
    cout << VectorCountable<int>::GetInstanceCount() << "\n";
    cout << "\nDone!" << endl;
}

```

Резултати от изпълнението:

```

Example 1
Start...
1
2
1

```

Done!

3. Специализация (specialization) на клас шаблон

Пример: Специализация на класа шаблон **Vector** за типа **void***:

```

/*
 * VectorVoidPtr.h
 *
 * Created on: 12.2011
 */
#ifndef VECTORVOIDPTR_H_
#define VECTORVOIDPTR_H_

#include "Vector.h"

template<> class Vector<void*> {
private:
    void* *ptr;

```

```

size_t c; //capacity
size_t sz; //elements count
public:
Vector() {
    c = 10;
    ptr = new void*[c];
    sz = 0;
}

~Vector() {
    delete[] ptr;
}

size_t size() const {
    return sz;
}

void* operator[](int i) {
    return ptr[i];
}

void push_back(const void* x) {
    ptr[sz] = const_cast<void*>(x);
    sz++;
}
//...
};

#endif /* VECTORVOIDPTR_H_ */

```

Тестване:

```

/*
 * Example2.h
 *
 * Created on: 12.2011
 *
 */

#ifndef EXAMPLE2_H_
#define EXAMPLE2_H_

#include "VectorVoidPtr.h"

class Example2 {
public:
    void start();
};

#endif /* EXAMPLE2_H_ */

/*
 * Example2.cpp
 *
 * Created on: 12.2011
 *
 */

#include "Example2.h"
#include <string>
#include <typeinfo>
#include <iostream>
#include <cstddef>
using namespace std;

```

```

void Example2::start() {
    cout << "Start..." << endl;
    Vector<void*> v;
    int n1 = 100;
    int n2 = 200;
    int n3 = 300;
    string s1("Word1");
    string s2("Word2");

    v.push_back(&n1);
    v.push_back(&s1);
    v.push_back(&n2);
    v.push_back(&s2);
    v.push_back(&n3);

    cout << "Size =" << v.size() << endl;
    for (size_t i = 0; i < v.size(); i++)
        if (i % 2 == 0)
            cout << "v[" << i << "] = " << *(static_cast<int*> (v[i])) << endl;
        else
            cout << "v[" << i << "] = " << *(static_cast<string*> (v[i]))
                << endl;

    cout << "\nDone!" << endl;
}

```

Виж <http://www.cplusplus.com/doc/tutorial/typecasting/> за повече подробности.

Резултати от тестването:

```

Example 2
Start...
Size =5
v[0] = 100
v[1] = Word1
v[2] = 200
v[3] = Word2
v[4] = 300

```

Done!

4. Клас шаблон, като наследник на обикновен клас

Пример: Клас шаблон **Vector<T*>** – специализация на класа шаблон **Vector** за произволни указатели, наследник на класа **Vector<void*>**:

```

/*
 * VectorPtr.h
 *
 * Created on: 12.2011
 *
 */
#ifndef VECTORPTR_H_
#define VECTORPTR_H_

#include "VectorVoidPtr.h"

template<class T> class Vector<T*> : private Vector<void*> {
    //...
};

#endif /* VECTORPTR_H_ */

```

5. Клас шаблон, като наследник на клас шаблон

Пример: Клас шаблон сортиран вектор **SortedVector**, наследник на класа шаблон **Vector**:

```
/*
 * SortedVector.h
 *
 * Created on: 12.2011
 *
 */

#ifndef SORTEDVECTOR_H_
#define SORTEDVECTOR_H_


#include "Vector.h"

template<typename T, int initialCapacity = 1>
class SortedVector: public Vector<T, initialCapacity> {
public:
    template<class Compare>
    void push_in_sortedVector(Compare comp, const T& x) {
        size_t i = 0;
        for (; i < Vector<T, initialCapacity>::size(); i++)
            if (comp(x, Vector<T, initialCapacity>::operator[](i)))
                break;
        insert(i, 1, x);
    }
};

#endif /* SORTEDVECTOR_H_ */
```

Тестване:

```
/*
 * Example3.h
 *
 * Created on: 12.2011
 *
 */

#ifndef EXAMPLE3_H_
#define EXAMPLE3_H_


#include "SortedVector.h"

class Example3 {
public:
    void start();
};

#endif /* EXAMPLE3_H_ */


/*
 * Example3.cpp
 *
 * Created on: 12.2011
 *
 */

#include "Example3.h"
#include <cstdlib>
#include <ctime>
#include <iostream>
using namespace std;
```

```

class IntComp {
public:
    bool operator()(int first, int second) {
        return first <= second;
    }
};

void Example3::start() {
    cout << "Start..." << endl;
    SortedVector<int> v;
    int n;

    cout << "\n      Inserting:" << endl;
    srand(n);
    for (int i = 0; i < 5; i++) {
        n = rand();
        cout << "push_in_sortedLVector " << n << endl;
        v.push_in_sortedVector(IntComp(), n);
        cout << "push_in_sortedVector " << -n << endl;
        v.push_in_sortedVector(IntComp(), -n);
    }

    cout << "Size =" << v.size() << endl;
    for (size_t i = 0; i < v.size(); i++)
        cout << "v[" << i << "] = " << v[i] << endl;

    cout << "\nDone!" << endl;
}

```

Резултати от изпълнението:

```

Example 3
Start...

      Inserting:
push_in_sortedLVector 35
push_in_sortedVector -35
push_in_sortedLVector 29739
push_in_sortedVector -29739
push_in_sortedLVector 3374
push_in_sortedVector -3374
push_in_sortedLVector 11141
push_in_sortedVector -11141
push_in_sortedLVector 31308
push_in_sortedVector -31308
Size =10
v[0] = -31308
v[1] = -29739
v[2] = -11141
v[3] = -3374
v[4] = -35
v[5] = 35
v[6] = 3374
v[7] = 11141
v[8] = 29739
v[9] = 31308

Done!

```

6. **Обобщен алгоритъм** – функция шаблон, която изпълнява дадена операция над последователен контейнер от произволен тип. Параметър на обобщен алгоритъм се явяват итератори на контейнера. Примерите са от стандартната библиотека на C++ – виж <http://www.cplusplus.com/reference/algorithms/>.

Пример 1: Функция шаблон за сортиране:

```
template<class RandomAccessIterator>
void sort(RandomAccessIterator first, RandomAccessIterator last);

template<class RandomAccessIterator, class Compare>
void sort(RandomAccessIterator first, RandomAccessIterator last, Compare comp);
//Sorts the elements in the range [first,last) into ascending order.
//The elements are compared using operator< for the first version,
//and comp for the second.
```

Пример 2: Функция шаблон за копиране:

```
template<class InputIterator, class OutputIterator>
OutputIterator copy(InputIterator first, InputIterator last,
                    OutputIterator result);
//Copies the elements in the range [first,last) into a range beginning at
//result. Returns an iterator to the end of the destination range
//(which points to the element following the copy of last).
```

Пример 3: Функция шаблон, която се изпълнява за всеки елемент на контейнер в интервала [first,last):

```
template<class InputIterator, class Function>
Function for_each(InputIterator first, InputIterator last, Function f) {
    for (; first != last; ++first)
        f(*first);
    return f;
}
//Applies function f to each of the elements in the range [first,last).
```

7. **Функционален обект (функтор)** – обект с поведение на функция. Клас, в който е предефинирана операцията operator() се нарича *клас-функтор*. Много често параметри на алгоритмите са явяват функторите.

Пример 1: Клас-функтор за проверка дали цяло число е нечетено:

```
class Odd {
public:
    bool operator()(const int &d) {
        return d % 2 == 1;
    }
};
```

Пример 2: Клас-функтор за сравнение:

```
template<class T>
class GT {
public:
    bool operator()(const T &first, const T &second) {
        return first > second;
    }
};
```

Тестване:

```
/*
 * Example5.h
 *
 * Created on: 12.2011
 */
*/
```

```

#ifndef EXAMPLE5_H_
#define EXAMPLE5_H_

#include "Algorithms.h"

class Example5 {
public:
    void start();
};

#endif /* EXAMPLE5_H_ */

/*
 * Example5.cpp
 *
 * Created on: 12.2011
 *
 */

#include "Example5.h"
#include <iostream>
using namespace std;

void Example5::start() {
    cout << "Start..." << endl;
    cout << "    Odd isOdd:" << endl;
    Odd isOdd;
    cout << "isOdd(5) " << (isOdd(5) ? "true" : "false") << endl;
    cout << "isOdd(12) " << (isOdd(12) ? "true" : "false") << endl;

    cout << "    GT<int> comp:" << endl;
    GT<int> comp;
    cout << "comp(3,5) = " << (comp(3, 5) ? "true" : "false") << endl;
    cout << "comp(5,3) = " << (comp(5, 3) ? "true" : "false") << endl;

    cout << "\nDone!" << endl;
}

```

Резултати от тестването:

```

Example 5
Start...
    Odd isOdd:
isOdd(5)true
isOdd(12)false
    GT<int> comp:
comp(3,5) = false
comp(5,3) = true

Done!

```

Пример 3: Клас-функция за определяне на минимум:

```

template<class T>
class Min {
private:
    T min;
public:
    Min(T x) :
        min(x) {
    }
    void operator()(const T &x) {
        if (x < min)
            min = x;
    }
}

```

```

    T result() const {
        return min;
    }
};

```

Тестване:

```

/*
 * Example6.h
 *
 * Created on: 12.2011
 *
 */

#ifndef EXAMPLE6_H_
#define EXAMPLE6_H_


#include "Algorithms.h"

class Example6 {
public:
    void start();
};

#endif /* EXAMPLE6_H_ */


/*
 * Example6.cpp
 *
 * Created on: 12.2011
 *
 */

#include "Example6.h"
#include <iostream>
#include <vector>
using namespace std;

void printFunction(int i) {
    cout << " " << i;
}

struct PrintFunctor {
    void operator()(int i) {
        cout << " " << i;
    }
};

void Example6::start() {
    cout << "Start..." << endl;
    vector<int> v;
    v.push_back(10);
    v.push_back(20);
    v.push_back(30);

    cout << "\nVector contains: ";
    for_each(v.begin(), v.end(), printFunction);

    // or:
    cout << "\nVector contains: ";
    for_each(v.begin(), v.end(), PrintFunctor());

    cout << "\nMin value: ";
    vector<int>::iterator it = v.begin();
    Min<int> min(*it);
}

```

```

for_each(++it, v.end(), min);
cout << min.result() << endl;

cout << "\nDone!" << endl;
}

```

Резултати от тестването:

```

Example 6
Start...

Vector contains: 10 20 30
Vector contains: 10 20 30
Min value: 10

Done!

```

Задача: Да се реализира АТД *приоритетна опашка с елементи от тип T* – тип опашка, такава, че на всеки елемент е съпоставен приоритет, който определя редът на достъп до елементите: първи елемент в опашката е този с най-голям приоритет, с множество от операции – http://www.cplusplus.com/reference/stl/priority_queue:

Member functions

<u>(constructor)</u>	Construct priority queue (public member function)
<u>empty</u>	Test whether container is empty (public member function)
<u>size</u>	Return size (public member function)
<u>top</u>	Access top element (public member function)
<u>push</u>	Insert element (public member function)
<u>pop</u>	Remove top element (public member function)

Спецификация на класа шаблон PriorityQueue:

```

/*
 * PriorityQueue.h
 *
 * Created on: 11.2011
 *
 */

#ifndef PRIORITQUEUE_H_
#define PRIORITQUEUE_H_

#include <cstddef>
#include <vector>
using namespace std;

template<class T>
class less {
public:
    bool operator()(T first, T second) {
        return first < second;
    }
};

template<class T>
class gt {
public:
    bool operator()(T first, T second) {
        return first > second;
    }
};

```

```

template<class T, class Container = vector<T>, class Compare = less<T> >

//Where the template parameters have the following meanings:
//T: Type of the elements.
//Container: Type of the underlying container object used to store and access
//the elements.
//Compare: Comparison class: A class such that the expression comp(a,b),
//where comp is an object of this class and a and b are elements of the
//container, returns true if a is to be placed earlier than b in a strict
//weak ordering operation. This can either be a class implementing a function
//call operator or a pointer to a function. This defaults to less<T>, which
//returns the same as applying the less-than operator (a<b). The priority_queue
//object uses this expression when an element is inserted or removed from it
//(using push or pop, respectively) to grant that the element popped is always
//the greater in the priority queue.

class PriorityQueue {
protected:
    Container c;
    Compare comp;
public:
    explicit PriorityQueue(const Compare& x = Compare(), const Container& y = Container());

    template<class InputIterator>
    PriorityQueue(InputIterator first, InputIterator last, const Compare& x =
        Compare(), const Container& y = Container());

    bool empty() const;
    //Returns whether the priority_queue is empty, i.e. whether its size is 0.

    size_t size() const;
    //Returns the number of elements in the priority_queue.

    const T& top() const;
    //Returns a constant reference to the top element
    //in the priority_queue. The top element is the element
    //that compares higher in the priority_queue, and the next
    //that is removed from the container when
    //PriorityQueue::pop is called.

    void push(const T& x);
    //Inserts a new element in the priority_queue. The content
    //of this new element is initialized to a copy of x.

    void pop();
    //Removes the element on top of the priority_queue,
    //effectively reducing its size by one. The value of this
    //element can be retrieved before being popped by calling
    //member priority_queue::top.
    //This calls the removed element's destructor.
};

#include "PriorityQueueImp.h"

#endif /* PRIORITYQUEUE_H_ */

```

Реализация на класа шаблон PriorityQueue:

```

/*
 * PriorityQueueImp.h
 *
 * Created on: 12.2011
 */

```

```

#ifndef PRIORITYQUEUEIMP_H_
#define PRIORITYQUEUEIMP_H_

template<class T, class Container, class Compare>
PriorityQueue<T, Container, Compare>::PriorityQueue(const Compare& x,
    const Container& y) :
    c(y), comp(x) {
    PriorityQueue<T, Container, Compare> temp(y.begin(), y.end(), x, y);
    (*this) = temp;
}

template<class T, class Container, class Compare>
template<class InputIterator>
PriorityQueue<T, Container, Compare>::PriorityQueue(InputIterator first,
    InputIterator last, const Compare& x, const Container& y) {
    while (first != last) {
        push(*first);
        ++first;
    }
}

template<class T, class Container, class Compare>
bool PriorityQueue<T, Container, Compare>::empty() const {
    return c.empty();
}

template<class T, class Container, class Compare>
size_t PriorityQueue<T, Container, Compare>::size() const {
    return c.size();
}

template<class T, class Container, class Compare>
const T& PriorityQueue<T, Container, Compare>::top() const {
    return c.back();
}

template<class T, class Container, class Compare>
void PriorityQueue<T, Container, Compare>::pop() {
    c.pop_back();
}

template<class T, class Container, class Compare>
void PriorityQueue<T, Container, Compare>::push(const T& x) {
    typename Container::iterator it;
    for (it = c.begin(); it != c.end(); ++it)
        if (comp(x, *it))
            break;
    c.insert(it, x);
}

#endif /* PRIORITYQUEUEIMP_H_ */

```

Тестване:

```

//=====
// Name      : PriorityQueues.cpp
// Version   : 11.2011
// Description : PriorityQueues
//=====

#include "PriorityQueue.h"
#include <cstdlib>
#include <ctime>
#include <list>
#include <iostream>
using namespace std;

```

```

int main() {
    int n;
    cout << "\n      PriorityQueue<int> queue1;" << endl;
    PriorityQueue<int> queue1;
    cout << "  Inserting:" << endl;
    rand(n);
    for (int i = 0; i < 5; i++) {
        n = rand();
        cout << "push " << n << endl;
        queue1.push(n);
        cout << "push " << -n << endl;
        queue1.push(-n);
    }

    cout << "\n  Print queue1 ..." << endl;
    while (!queue1.empty()) {
        cout << queue1.top() << " ";
        queue1.pop();
    }

    cout << "\n\n      PriorityQueue<int,list<int>,gt<int> > queue2;" << endl;
    PriorityQueue<int, list<int> , gt<int> > queue2;
    cout << "  Inserting:" << endl;
    rand(n);
    for (int i = 0; i < 5; i++) {
        n = rand();
        cout << "push " << n << endl;
        queue2.push(n);
        cout << "push " << -n << endl;
        queue2.push(-n);
    }

    cout << "\n  Print queue2 ..." << endl;
    while (!queue2.empty()) {
        cout << queue2.top() << " ";
        queue2.pop();
    }

    cout << "\n\n      list<int> c;" << endl;
    list<int> c;
    cout << "  Inserting:" << endl;
    rand(n);
    for (int i = 0; i < 3; i++) {
        n = rand();
        cout << "push " << n << endl;
        c.push_front(n);
        cout << "push " << -n << endl;
        c.push_front(-n);
        cout << "push " << n << endl;
        c.push_front(n);
    }

    cout << "\n      PriorityQueue<int,list<int> > queue3(c.begin(), c.end());"
         << endl;
    PriorityQueue<int, list<int> > queue3(c.begin(), c.end());
    cout << "  Print queue3 ..." << endl;
    while (!queue3.empty()) {
        cout << queue3.top() << " ";
        queue3.pop();
    }

    cout << "\n\nDone!" << endl;

    return 0;
}

```

Резултати от тестването:

```

PriorityQueue<int> queue1;
Inserting:
push 25646
push -25646
push 10508
push -10508
push 645
push -645
push 27280
push -27280
push 20221
push -20221

Print queue1 ...
27280 25646 20221 10508 645 -645 -10508 -20221 -25646 -27280

PriorityQueue<int, list<int>, gt<int> > queue2;
Inserting:
push 535
push -535
push 1101
push -1101
push 18217
push -18217
push 14672
push -14672
push 11544
push -11544

Print queue2 ...
-18217 -14672 -11544 -1101 -535 535 1101 11544 14672 18217

list<int> c;
Inserting:
push 4968
push -4968
push 4968
push 27674
push -27674
push 27674
push 3441
push -3441
push 3441

PriorityQueue<int, list<int> > queue3(c.begin(), c.end());
Print queue3 ...
27674 27674 4968 4968 3441 3441 -3441 -4968 -27674

```

Done!

Задача: Нека **Key** и **T** са типове и D_{Key} и D_T са съответните им множества от стойности. Елементите на D_{Key} ще наричаме *ключове*, а тези на D_T – *стойности*. Нека **Pair** е тип с множество от стойности $D_{Pair} = D_{Key} \times D_T$. Да се реализира АТД *ассоциативен масив* (*таблица, изображение (map), речник (dictionary)*) с елементи от тип **Pair**, като за всеки два различни елемента $(key_1, value_1)$ и $(key_2, value_2)$ е изпълнено $key_1 \neq key_2$ и операции, съгласно дадената спецификация – виж <http://www.cplusplus.com/reference/stl/map/>:

Member functions

[\(constructor\)](#)

Construct map (public member function)

<u>(destructor)</u>	Map destructor (public member function)
<u>operator=</u>	Copy container content (public member function)

Iterators:

<u>begin</u>	Return iterator to beginning (public member function)
<u>end</u>	Return iterator to end (public member function)
<u>rbegin</u>	Return reverse iterator to reverse beginning (public member function)
<u>rend</u>	Return reverse iterator to reverse end (public member function)

Capacity:

<u>empty</u>	Test whether container is empty (public member function)
<u>size</u>	Return container size (public member function)
<u>max_size</u>	Return maximum size (public member function)

Element access:

<u>operator[]</u>	Access element (public member function)
-----------------------------------	--

Modifiers:

<u>insert</u>	Insert element (public member function)
<u>erase</u>	Erase elements (public member function)
<u>swap</u>	Swap content (public member function)
<u>clear</u>	Clear content (public member function)

Observers:

<u>key_comp</u>	Return key comparison object (public member function)
<u>value_comp</u>	Return value comparison object (public member function)

Operations:

<u>find</u>	Get iterator to element (public member function)
<u>count</u>	Count elements with a specific key (public member function)
<u>lower_bound</u>	Return iterator to lower bound (public member function)
<u>upper_bound</u>	Return iterator to upper bound (public member function)
<u>equal_range</u>	Get range of equal elements (public member function)

ПРОЕКТ: Реализация и използване на таблици

Класът шаблон **Pair** представя наредена двойка от обекти:

```
/*
 * Pair.h
 *
 * Created on: 12.2011
 *
 */
#ifndef PAIR_H_
#define PAIR_H_

template <class T1, class T2> struct Pair {
    typedef T1 first_type;
    typedef T2 second_type;

    T1 first;
    T2 second;
};

#endif // PAIR_H_
```

```

T2 second;
Pair() :
    first(T1()), second(T2()) {
}
Pair(const T1& x, const T2& y) :
    first(x), second(y) {
}
};

#endif /* PAIR_H_ */

```

Да се реализират:

- Клас шаблон **ListTable** - представяне на елементите на таблица чрез линеен списък:
`template < class Key, class T > class ListTable;`

- Клас шаблон **HashTable** - представяне на елементите на таблица чрез масив array с фиксиран размер M от списъци с използване на хеш функция $h: D_{Key} \rightarrow \{0, 1, \dots, M-1\}$, като списъкът array[i] съдържа всеки елемент (key,value) на таблицата, за който $h(key) = i$:
`template < class Key, class T, class H = Hash<Key>, class EQ = equalTo<Key> >
class HashTable;`

- Клас шаблон **TreeTable** - представяне на елементите на таблица чрез наредено по ключовете двоично дърво:
`template < class Key, class T, class Compare = less<Key> > class TreeTable;`

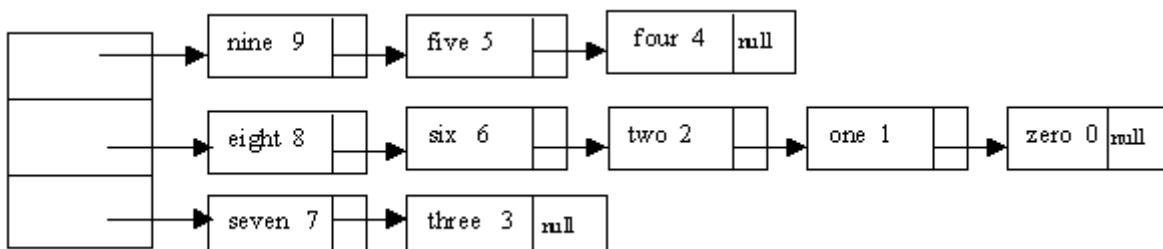
Пример: Нека table е таблица с ключове - обекти на класа **string** и целочислени стойности. Следващата таблица съдържа операции за включване в table, които се изпълняват в реда от таблицата. Хешкодът на името $w = c_0c_1\dots c_{n-1}$ (обект на класа **string**), се определя по следния начин: $(c_0 + c_1 + \dots + c_{n-1}) \bmod 3$:

Име на цифра на английски език	Цифра	Операция	Сума на буквите в името	Хешкод на името
zero	0	table["zero"] = 0	448	1
one	1	table["one"] = 1	322	1
two	2	table["two"] = 2	346	1
three	3	table["three"] = 3	536	2
four	4	table["four"] = 4	444	0
five	5	table["five"] = 5	426	0
six	6	table["six"] = 6	340	1
seven	7	table["seven"] = 7	545	2
eight	8	table["eight"] = 8	529	1
nine	9	table["nine"] = 9	426	0

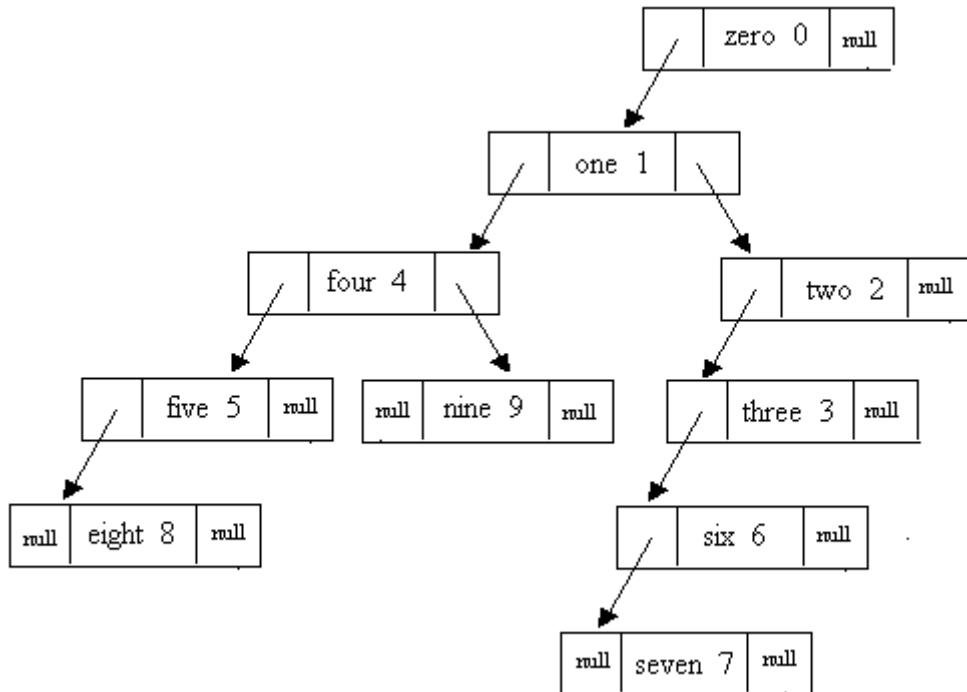
- table – представяне на елементите чрез линеен едносвързан списък



- table – представяне на елементите чрез масив от три линейни едносвързани списъка



3. table – представяне на елементите чрез наредено по ключовете двоично дърво



ИНТЕРФЕЙСИ

Интерфейс на класа шаблон *ListTable*:

```

/*
 * ListTable.h
 *
 * Created on: 12.2011
 *
 */

#ifndef LISTTABLE_H_
#define LISTTABLE_H_

#include "Pair.h"
#include <cstddef>
#include <list>
using namespace std;

template<class Key, class T>
class ListTable {
private:
    list<Pair<Key, T>> elements;
    typedef list<Pair<Key, T>> container;
public:
    typedef typename container::iterator iterator;
    typedef typename container::const_iterator const_iterator;
    typedef typename container::reverse_iterator reverse_iterator;
    typedef typename container::const_reverse_iterator const_reverse_iterator;

    //Constructors
    ListTable();

    template<class InputIterator>
    ListTable(InputIterator first, InputIterator last);

    ListTable(const ListTable<Key, T> &x);
}
  
```

```

//Destructor
~ListTable();

ListTable<Key, T>& operator=(const ListTable<Key, T>& x);
//Assigns a copy of the elements in x as the new content for the container.
//The elements contained in the object before the call are dropped, and
//replaced by copies of those in map x, if any. After a call to this member
//function, both the map object and x will have the same size and
//compare equal to each other.

//Iterators
iterator begin();
const_iterator begin() const;
iterator end();
const_iterator end() const;
reverse_iterator rbegin();
const_reverse_iterator rbegin() const;
reverse_iterator rend();
const_reverse_iterator rend() const;

//Capacity:
size_t size() const;
//Returns the number of elements in the map.

bool empty() const;
//Returns whether the map is empty, i.e. whether its size is 0.

//Element access:
T& operator[](const Key& x);
//If x matches the key of an element in the container, the function
//returns a reference to its mapped value. If x does not match
//the key of any element in the container, the function inserts
//a new element with that key and returns a reference to its mapped value.
//Notice that this always increases the map size by one, even if no
//mapped value is assigned to the element (the element is constructed
//using its default constructor).

//Modifiers
iterator insert(const Pair<Key, T>& x);
template<class InputIterator>
void insert(InputIterator first, InputIterator last);
//The map container is extended by inserting a single new element
//(if parameter x is used) or a sequence of elements (if input
//iterators are used). This effectively increases the container
//size by the amount of elements inserted. Because map containers
//do not allow for duplicate key values, the insertion operation
//checks for each element inserted whether another element exists
//already in the container with the same key value, if so,
//the element is not inserted and its mapped value is not changed
//in any way. The first version returns an iterator pointing to
//either the newly inserted element or to the element that already
//had its same value in the map.

void erase(iterator position);
size_t erase(const Key& x);
void erase(iterator first, iterator last);
//Removes from the map container either a single element or a range
//of elements ([first,last)). This effectively reduces the container
//size by the number of elements removed, calling each element's
//destructor.

void swap(ListTable<Key, T>& mp);
//Exchanges the content of the container with the content of mp,
//which is another map object containing elements of the same type.
//Sizes may differ.

```

```

void clear();
//All the elements in the container are dropped: their destructors
//are called, and then they are removed from the container, leaving
//it with a size of 0.

//Operations:
iterator find(const Key& x);
const_iterator find(const Key& x) const;
//Searches the container for an element with x as key and returns
//an iterator to it if found, otherwise it returns an iterator to
//map::end (the element past the end of the container).
};

#include "ListTableImp.h"

#endif /* LISTTABLE_H_ */

```

ТЕСТВАНЕ

Интерфейс на класа Tester:

```

/*
 * Tester.h
 *
 * Created on: 12.2011
 *
 */

#ifndef TESTER_H_
#define TESTER_H_

#include "ListTable.h"

class Tester {
private:
    void test1();
    void test2();
    void test3();
public:
    void start();
};

#endif /* TESTER_H_ */

```

Реализация на класа Tester:

```

/*
 * Tester.cpp
 *
 * Created on: 12.2011
 *
 */

#include "Tester.h"
#include <string>
#include <iostream>
using namespace std;

void Tester::start() {
    cout << "      Test 1" << endl;
    test1();

    cout << "      Test 2" << endl;
    test2();
}

```

```

cout << "      Test 3" << endl;
test3();
}

//================================================================TEST 1=====
void Tester::test1() {
    cout << "Start..." << endl;

    ListTable<char, int> mymap;
    ListTable<char, int>::iterator it;

    // first insert function version (single parameter):
    mymap.insert(Pair<char, int> ('a', 100));
    mymap.insert(Pair<char, int> ('z', 200));
    mymap.insert(Pair<char, int> ('z', 500));

    // showing content:
    cout << "mymap contains:\n";
    for (it = mymap.begin(); it != mymap.end(); it++)
        cout << (*it).first << " => " << (*it).second << endl;

    cout << "\nDone!" << endl;
}

//================================================================TEST 2=====
void Tester::test2() {
    cout << "Start..." << endl;

    ListTable<char, int> mymap;
    ListTable<char, int>::iterator it;

    // insert some values:
    mymap['a'] = 10;
    mymap['b'] = 20;
    mymap['c'] = 30;
    mymap['d'] = 40;
    mymap['e'] = 50;
    mymap['f'] = 60;

    it = mymap.find('b');
    mymap.erase(it); // erasing by iterator

    mymap.erase('c'); // erasing by key

    // show content:
    for (it = mymap.begin(); it != mymap.end(); it++)
        cout << (*it).first << " => " << (*it).second << endl;

    cout << "\n\nDone!" << endl;
}

//================================================================TEST 3=====
void Tester::test3() {
    cout << "Start..." << endl;

    ListTable<string, int> table;
    ListTable<string, int>::iterator it;

    // insert some values:
    table["zero"] = 0;
    table["one"] = 1;
    table["two"] = 2;
    table["three"] = 3;
    table["four"] = 4;
    table["five"] = 5;
}

```

```

table["six"] = 6;
table["seven"] = 7;
table["eight"] = 8;
table["nine"] = 9;

// show content:
for (it = table.begin(); it != table.end(); it++)
    cout << (*it).first << " => " << (*it).second << endl;

cout << "\n\nDone!" << endl;
}

```

Резултати от тестването:

```

//=====
// Name      : Tables.cpp
// Version   : 12.2011
// Description : Tables
//=====

#include "Tester.h"

int main() {
    Tester().start();
    return 0;
}

Test 1
Start...
mymap contains:
z => 200
a => 100

Done!
Test 2
Start...
f => 60
e => 50
d => 40
a => 10

Done!
Test 3
Start...
nine => 9
eight => 8
seven => 7
six => 6
five => 5
four => 4
three => 3
two => 2
one => 1
zero => 0

Done!

```

РЕАЛИЗАЦИЯ

*Реализация на класа шаблон **ListTable**:*

```

/*
 * ListTableImp.h

```

```

/*
 * Created on: 12.2011
 */
*/
#ifndef LISTTABLEIMP_H_
#define LISTTABLEIMP_H_

template<class Key, class T>
ListTable<Key, T>::ListTable() {
}

template<class Key, class T>
ListTable<Key, T>::~ListTable() {
}

template<class Key, class T>
typename ListTable<Key, T>::iterator ListTable<Key, T>::begin() {
    return elements.begin();
}

template<class Key, class T>
typename ListTable<Key, T>::iterator ListTable<Key, T>::end() {
    return elements.end();
}

template<class Key, class T>
T& ListTable<Key, T>::operator[](const Key& x) {
    ListTable<Key, T>::iterator it = find(x);
    if (it == end())
        it = insert(Pair<Key, T> (x, T()));
    return (*it).second;
}

template<class Key, class T>
typename ListTable<Key, T>::iterator ListTable<Key, T>::insert(const Pair<Key,
    T>& x) {
    ListTable<Key, T>::iterator it = find(x.first);
    if (it == end()) {
        elements.push_front(x);
        it = begin();
    }
    return it;
}

template<class Key, class T>
void ListTable<Key, T>::erase(ListTable<Key, T>::iterator position) {
    elements.erase(position);
}

template<class Key, class T>
size_t ListTable<Key, T>::erase(const Key& x) {
    ListTable<Key, T>::iterator it = find(x);
    if (it != end())
        erase(it);
    return 0;
}

template<class Key, class T>
typename ListTable<Key, T>::iterator ListTable<Key, T>::find(const Key& x) {
    ListTable<Key, T>::iterator it;
    for (it = begin(); it != end(); it++)
        if ((*it).first == x)
            break;
    return it;
}

```

```
#endif /* LISTTABLEIMP_H_ */
```

Задача: Да се реализира АТД множество с елементи от тип *T* и операции, съгласно дадената спецификация – виж <http://www.cplusplus.com/reference/stl/set/>:

Member functions

<u>(constructor)</u>	Construct set (public member function)
<u>(destructor)</u>	Set destructor (public member function)
<u>operator=</u>	Copy container content (public member function)

Iterators:

<u>begin</u>	Return iterator to beginning (public member function)
<u>end</u>	Return iterator to end (public member function)
<u>rbegin</u>	Return reverse iterator to reverse beginning (public member function)
<u>rend</u>	Return reverse iterator to reverse end (public member function)

Capacity:

<u>empty</u>	Test whether container is empty (public member function)
<u>size</u>	Return container size (public member function)
<u>max_size</u>	Return maximum size (public member function)

Modifiers:

<u>insert</u>	Insert element (public member function)
<u>erase</u>	Erase elements (public member function)
<u>swap</u>	Swap content (public member function)
<u>clear</u>	Clear content (public member function)

Observers:

<u>key_comp</u>	Return comparison object (public member function)
---------------------------------	---

Operations:

<u>find</u>	Get iterator to element (public member function)
<u>count</u>	Count elements with a specific key (public member function)
<u>lower_bound</u>	Return iterator to lower bound (public member function)
<u>upper_bound</u>	Return iterator to upper bound (public member function)
<u>equal_range</u>	Get range of equal elements (public member function)

Упътване: Реализация на клас шаблон:

1. Вариант 1 – несортирано множество

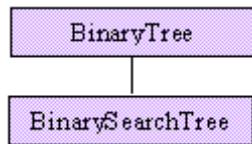
```
template < class T > class Set;
```

2. Вариант 2 – сортирано множество

```
template < class T, class Compare = less<T> > class Set;
```

Двоично дърво

ПРОЕКТ: Йерархия от класове шаблони за реализация и използване на двоично дърво:



ИНТЕРФЕЙСИ

*Интерфейс на класа шаблон **BinaryTree**:*

```

/*
 * BinaryTree.h
 *
 * Created on: 01.2012
 *
 */

#ifndef BINARYTREE_H_
#define BINARYTREE_H_


#include <cstddef>
#include <list>
using namespace std;

template<class T>
class Node {
public:
    T data;
    Node *left;
    Node *right;
    Node(T d) :
        data(d) {
        left = 0;
        right = 0;
    }
};

template<class T>
class BinaryTree {
private:
    list<T> elements;
    typedef list<T> container;
    void inorder(Node<T> *root);
    void clear(Node<T> *root);
    void copy(Node<T>* &newRoot, const Node<T> *root);
    void insert(Node<T>* &root, const T& x);
    void remove(Node<T> *root, const T& x);
    Node<T>* find(Node<T> *root, const T& x) const;
    size_t size(Node<T> *root) const;
    size_t height(Node<T> *root) const;
protected:
    Node<T> *root; //root element
public:
    typedef typename container::iterator inorderIterator;
    typedef typename container::const_iterator const_inorderIterator;

    //Constructors
    BinaryTree();
    BinaryTree(const BinaryTree<T> &x);
    template<class InputIterator>
  
```

```

BinaryTree(InputIterator first, InputIterator last);

//Iterators
inorderIterator beginInorder();
inorderIterator endInorder();
const_inorderIterator beginInorder() const;
const_inorderIterator endInorder() const;

//Capacity:
size_t size() const;
//Returns the number of elements in the tree.

bool empty() const;
//Returns whether the tree is empty, i.e. whether its size is 0.

//Modifiers:
BinaryTree<T>& operator=(const BinaryTree<T>& x);
//Assigns a copy of the elements in x as the new content for the container.
//The elements contained in the object before the call are dropped, and
//replaced by copies of those in map x, if any. After a call to this member
//function, both the map object and x will have the same size and
//compare equal to each other.

virtual void insert(const T& x);
//The tree container is extended by inserting an element. This effectively
//increases the container size by 1.

virtual void remove(const T& value);
//Removes from the tree an element with a specific value
//using ==. This calls the destructor of the object and
//reduces the tree size by 1.

void clear();
//All the elements of the tree are dropped: their destructors
//are called, and then they are removed from the tree, leaving
//the tree with a size of 0.

//Operations:
virtual bool find(const T& x) const;
//Searches the tree for an element x and returns true if found,
//otherwise it returns false.

size_t height() const;
//Returns the height of the tree.

size_t numberofLiefs() const;
//Returns the number of liefs in the tree.

//Destructor
~BinaryTree();
//Destructs the tree object. This calls each of the contained
//element's destructors, and deallocates all the storage
//capacity allocated by the tree.
};

#include "BinaryTreeImp.h"

#endif /* BINARYTREE_H_ */

```

*Интерфейс на класа шаблон **BinarySearchTree**:*

```

/*
 * BinarySearchTree.h
 *
 * Created on: 01.2012
 *
```

```

*/
```

```

#ifndef BINARYSEARCHTREE_H_
#define BINARYSEARCHTREE_H_
```

```

#include "BinaryTree.h"

template<class T>
class BinarySearchTree: public BinaryTree<T> {
private:
    void add(Node<T>* &root, const T& x);
    void erase(Node<T> *root, const T& x);
    Node<T>* search(Node<T> *root, const T& x) const;
public:
    //Constructor
    BinarySearchTree();
    BinarySearchTree(const BinaryTree<T> &x);
    template<class InputIterator>
    BinarySearchTree(InputIterator first, InputIterator last);

    //Operations:
    void insert(const T& x);
    void remove(const T& x);
    bool find(const T& x) const;
};
```

```

#include "BinarySearchTreeImp.h"

#endif /* BINARYSEARCHTREE_H_ */
```

РЕАЛИЗАЦИЯ

Реализация на класа шаблон **BinaryTree**:

```

/*
 * BinaryTreeImp.h
 *
 * Created on: 01.2012
 *
 */

#ifndef BINARYTREEIMP_H_
#define BINARYTREEIMP_H_
```

```

#include <algorithm>
#include <cstdlib>
#include <ctime>
#include <iostream>
using namespace std;
```

```

template<class T>
BinaryTree<T>::BinaryTree() {
    root = 0;
}
```

```

template<class T>
BinaryTree<T>::BinaryTree(const BinaryTree<T> &x) {
    cout << "BTree: Copy constructor" << endl;
    root = 0;
    copy(root, x.root);
}
```

```

template<class T>
void BinaryTree<T>::inorder(Node<T> *root) {
    if (root != 0) {
```

```

        inorder(root->left);
        elements.push_back(root->data);
        inorder(root->right);
    }
}

template<class T>
typename BinaryTree<T>::inorderIterator BinaryTree<T>::beginInorder() {
    elements.clear();
    inorder(root);
    return elements.begin();
}

template<class T>
typename BinaryTree<T>::inorderIterator BinaryTree<T>::endInorder() {
    return elements.end();
}

template<class T>
typename BinaryTree<T>::const_inorderIterator BinaryTree<T>::beginInorder()
const {
    elements.clear();
    inorder(root);
    return elements.begin();
}

template<class T>
typename BinaryTree<T>::const_inorderIterator BinaryTree<T>::endInorder() const
{
    return elements.end();
}

template<class T>
size_t BinaryTree<T>::size(Node<T> *root) const {
    if (root == 0)
        return 0;
    else
        return size(root->left) + 1 + size(root->right);
}

template<class T>
size_t BinaryTree<T>::size() const {
    return size(root);
}

template<class T>
bool BinaryTree<T>::empty() const {
    return root == 0;
}

template<class T>
BinaryTree<T>& BinaryTree<T>::operator=(const BinaryTree<T>& x) {
    cout << "BTree: Assignment operator" << endl;
    if (this != &x) {
        clear();
        copy(root, x.root);
    }
    return *this;
}

template<class T>
void BinaryTree<T>::insert(Node<T>* &root, const T& x) {
    if (root == 0)
        root = new Node<T> (x);
    else {
        int n = (rand() / 100) % 2;

```

```

    if (n == 0)
        insert(root->left, x);
    else
        insert(root->right, x);
}

template<class T>
void BinaryTree<T>::insert(const T& x) {
    insert(root, x);
}

template<class T>
void BinaryTree<T>::remove(Node<T> *root, const T& value) {
    cout << "BTree: Remove not implemented!" << endl;
}

template<class T>
void BinaryTree<T>::remove(const T& value) {
    remove(root, value);
}

template<class T>
void BinaryTree<T>::copy(Node<T>* &newRoot, const Node<T> *root) {
    if (root == 0)
        newRoot = 0;
    else {
        newRoot = new Node<T> (root->data);
        copy(newRoot->left, root->left);
        copy(newRoot->right, root->right);
    }
}

template<class T>
void BinaryTree<T>::clear(Node<T> *root) {
    if (root != 0) {
        clear(root->left);
        clear(root->right);
        delete root;
    }
}

template<class T>
void BinaryTree<T>::clear() {
    clear(root);
}

template<class T>
Node<T>* BinaryTree<T>::find(Node<T> *root, const T& value) const {
    cout << "BTree: Find not implemented!" << endl;
    return 0;
}

template<class T>
bool BinaryTree<T>::find(const T& value) const {
    return find(root, value) != 0;
}

template<class T>
size_t BinaryTree<T>::height(Node<T> *root) const {
    if (root == 0)
        return 0;
    else
        return 1 + max(height(root->left), height(root->right));
}

```

```

template<class T>
size_t BinaryTree<T>::height() const {
    return height(root);
}

template<class T>
BinaryTree<T>::~BinaryTree() {
    clear();
}

#endif /* BINARYTREEIMP_H_ */

```

Реализация на класа шаблон **BinarySearchTree**:

```

/*
 * BinarySearchTreeImp.h
 *
 * Created on: 01.2012
 *
 */

#ifndef BINARYSEARCHTREEIMP_H_
#define BINARYSEARCHTREEIMP_H_

template<class T>
BinarySearchTree<T>::BinarySearchTree() :
    BinaryTree<T>()
}

template<class T>
BinarySearchTree<T>::BinarySearchTree(const BinaryTree<T> &x) :
    BinaryTree<T> (x)
}

template<class T>
void BinarySearchTree<T>::add(Node<T>* &root, const T& x) {
    if (root == 0)
        root = new Node<T> (x);
    else if (x < root->data)
        add(root->left, x);
    else
        add(root->right, x);
}

template<class T>
void BinarySearchTree<T>::insert(const T& x) {
    add(BinarySearchTree<T>::root, x);
}

template<class T>
Node<T>* BinarySearchTree<T>::search(Node<T> *root, const T& x) const {
    if (root == 0)
        return 0;
    else if (x < root->data)
        return search(root->left, x);
    else if (root->data < x)
        return search(root->right, x);
    else
        return root;
}

template<class T>
bool BinarySearchTree<T>::find(const T& x) const {
    return search(BinaryTree<T>::root, x) != 0;
}

```

```

template<class T>
void BinarySearchTree<T>::erase(Node<T> *root, const T& value) {
    cout << "BSTree: Remove not implemented!" << endl;
}

template<class T>
void BinarySearchTree<T>::remove(const T& value) {
    erase(BinaryTree<T>::root, value);
}

#endif /* BINARYSEARCHTREEIMP_H_ */

```

TECTВАНЕ

*Интерфейс и реализация на класа **Pair**:*

```

/*
 * Pair.h
 *
 * Created on: 12.2011
 *
 */

#ifndef PAIR_H_
#define PAIR_H_

#include <iostream>
using namespace std;

template<class T1, class T2> struct Pair {
    friend ostream& operator<<(ostream &s, const Pair<T1, T2> &n) {
        return s << "(" << n.first << "," << n.second << ")";
    }

    T1 first;
    T2 second;
    Pair() :
        first(T1()), second(T2()) {
    }
    Pair(const T1& x, const T2& y) :
        first(x), second(y) {
    }
    bool operator<(const Pair<T1, T2> &x) const {
        return first.compare(x.first) < 0;
    }
};

#endif /* PAIR_H_ */

```

*Интерфейс на класа **BinaryTreeTester**:*

```

/*
 * BinaryTreeTester.h
 *
 * Created on: 01.2012
 *
 */

#ifndef BINARYTREETESTER_H_
#define BINARYTREETESTER_H_

#include "BinaryTree.h"
#include "BinarySearchTree.h"
#include <string>
using namespace std;

```

```

class BinaryTreeTester {
protected:
    template<class T>
    void treeOfIntegers(BinaryTree<T> &tree);

    template<class T>
    void treeOfStrings(BinaryTree<T> &tree);

    template<class T>
    void treeOfPairs(BinaryTree<T> &tree);

    template<class T>
    void treeTraversal(BinaryTree<T> &tree, ostream &output);
public:
    void testBTree();
    void testBSTree();
};

#endif /* BINARYTREETESTER_H_ */

```

*Реализация на класа **BinaryTreeTester**:*

```

/*
 * BinaryTreeTester.cpp
 *
 * Created on: 01.2012
 *
 */

#include "BinaryTreeTester.h"
#include <cstddef>
#include <cstdlib>
#include <ctime>
#include "Pair.h"
#include <iostream>
#include <fstream>
using namespace std;

void BinaryTreeTester::testBTree() {
    cout << "      BinaryTree: Test 1" << endl;
    BinaryTree<int> tree1;
    treeOfIntegers(tree1);

    cout << "      BinaryTree: Test 2" << endl;
    BinaryTree<string> tree2;
    treeOfStrings(tree2);

    cout << "      BinaryTree: Test 3" << endl;
    BinaryTree<Pair<string, string> > tree3;
    treeOfPairs(tree3);
}

void BinaryTreeTester::testBSTree() {
    cout << "      BinarySearchTree: Test 1" << endl;
    BinarySearchTree<int> tree1;
    treeOfIntegers(tree1);

    cout << "      BinarySearchTree: Test 2" << endl;
    BinarySearchTree<string> tree2;
    treeOfStrings(tree2);

    cout << "      BinarySearchTree: Test 3" << endl;
    BinarySearchTree<Pair<string, string> > tree3;
    treeOfPairs(tree3);
}

```

```

template<class T>
void BinaryTreeTester::treeOfIntegers(BinaryTree<T> &tree) {
    cout << "      Create tree of integers" << endl;
    int n;
    srand(time(0));
    for (int i = 0; i < 3; i++) {
        n = rand();
        cout << "insert " << n << endl;
        tree.insert(n);
        cout << "insert " << -n << endl;
        tree.insert(-n);
    }

    cout << "tree contains " << n << endl;
    string f = tree.find(n) ? ": true" : ": false";
    cout << "the result is " << f << endl;

    cout << "tree remove " << n << endl;
    tree.remove(n);

    cout << "Test tree and write results to the file cout" << endl;
    treeTraversal(tree, cout);

    cout << "BinaryTree<T> newTree1=tree;" << endl;
    BinaryTree<T> newTree1 = tree;
    cout << "Test newTree1 and write results to the file cout" << endl;
    treeTraversal(newTree1, cout);

    cout << "BinaryTree<T> newTree2; newTree2=tree;" << endl;
    BinaryTree<T> newTree2;
    newTree2 = tree;
    cout << "Test newTree2 and write results to the file cout" << endl;
    treeTraversal(newTree2, cout);
}

template<class T>
void BinaryTreeTester::treeOfStrings(BinaryTree<T> &tree) {
    string path("D:\\UserFolder\\CPPProjects2\\Trees\\src\\");
    string inFile(path + "testTreeData1.txt");
    ifstream input(inFile.c_str());
    if (!input.is_open()) {
        cerr << "ERROR: cannot open file " << inFile << endl;
        exit(1);
    }

    string outFile(path + "testTreeResult1.txt");
    ofstream output;
    output.open(outFile.c_str());
    if (!output.is_open()) {
        cerr << "ERROR: cannot open file " << outFile << endl;
        exit(1);
    }

    cout << "      Create tree of strings" << endl;
    cout << "Read text from the file " << inFile << endl;
    string line;
    getline(input, line);
    while (!input.eof()) {
        tree.insert(line);
        getline(input, line);
    }
    input.close();

    cout << "Test and write results to the file " << outFile << endl;
}

```

```

treeTraversal(tree, output);
output.close();
}

template<class T>
void BinaryTreeTester::treeOfPairs(BinaryTree<T> &tree) {
    string path("D:\\UserFolder\\CPPProjects2\\Trees\\src\\");
    string inFile(path + "testTreeData2.txt");
    ifstream input(inFile.c_str());
    if (!input.is_open()) {
        cerr << "ERROR: cannot open file " << inFile << endl;
        exit(1);
    }

    string outFile(path + "testTreeResult2.txt");
    ofstream output;
    output.open(outFile.c_str());
    if (!output.is_open()) {
        cerr << "ERROR: cannot open file " << outFile << endl;
        exit(1);
    }

    cout << "      Create tree of Pairs<string,string>" << endl;
    cout << "Read text from the file " << inFile << endl;
    string line;
    for (int i = 1; i <= 10; i++) {
        getline(input, line);
        size_t index = line.find_first_of('@');
        string word = line.substr(0, index);
        string digit = line.substr(index + 1);
        tree.insert(Pair<string, string> (word, digit));
    }
    input.close();

    cout << "Test and write results to the file " << outFile << endl;
    treeTraversal(tree, output);
    output.close();
}

template<class T>
void BinaryTreeTester::treeTraversal(BinaryTree<T> &tree, ostream &output) {
    output << "      Testing..." << endl;
    output << "Size of the tree: " << tree.size() << endl;
    output << "Height of the tree: " << tree.height() << endl;

    output << "      Tree inorder traversal:" << endl;
    typename BinaryTree<T>::inorderIterator it;
    for (it = tree.beginInorder(); it != tree.endInorder(); ++it)
        output << (*it) << endl;

    output << "\nDone!" << endl;
}

```

*Резултати от тестването на класа **BinaryTree**:*

```

//=====
// Name      : Trees.cpp
// Version   : 01.2012
// Description : Binary trees
//=====

#include "BinaryTreeTester.h"

int main() {
    BinaryTreeTester().testBTree();
}

```

```
//BinaryTreeTester().testBSTree();
return 0;
}

BinaryTree: Test 1
Create tree of integers
insert 2655
insert -2655
insert 1528
insert -1528
insert 23889
insert -23889
tree contains 23889
BTree: Find not implemented!
the result is : false
tree remove 23889
BTree: Remove not implemented!
Test tree and write results to the file cout
    Testing...
Size of the tree: 6
Height of the tree: 4
    Tree inorder traversal:
23889
-1528
2655
-2655
-23889
1528

Done!
BinaryTree<T> newTree1=tree;
BTree: Copy constructor
Test newTree1 and write results to the file cout
    Testing...
Size of the tree: 6
Height of the tree: 4
    Tree inorder traversal:
23889
-1528
2655
-2655
-23889
1528

Done!
BinaryTree<T> newTree2; newTree2=tree;
BTree: Assignment operator
Test newTree2 and write results to the file cout
    Testing...
Size of the tree: 6
Height of the tree: 4
    Tree inorder traversal:
23889
-1528
2655
-2655
-23889
1528

Done!
    BinaryTree: Test 2
    Create tree of strings
Read text from the file D:\UserFolder\CPPProjects2\Trees\src\testTreeData1.txt
Test and write results to the file
D:\UserFolder\CPPProjects2\Trees\src\testTreeResult1.txt
    BinaryTree: Test 3
```

```
Create tree of Pairs<string, string>
Read text from the file D:\UserFolder\CPPProjects2\Trees\src\testTreeData2.txt
Test and write results to the file
D:\UserFolder\CPPProjects2\Trees\src\testTreeResult2.txt
```

Файл testTreeData1.txt	Файл testTreeResult1.txt
Word2 Word10 Word7 Word1 Word5 Word7 Word2 Word9 Word8 Word7	Testing... Size of the tree: 10 Height of the tree: 5 Tree inorder traversal: Word2 Word7 Word7 Word2 Word5 Word9 Word10 Word8 Word1 Word7
Файл testTreeData2.txt	Файл testTreeResult2.txt
zero@0 one@1 two@2 three@3 four@4 five@5 six@6 seven@7 eight@8 nine@9	Testing... Size of the tree: 10 Height of the tree: 5 Tree inorder traversal: (five,5) (seven,7) (two,2) (zero,0) (six,6) (nine,9) (three,3) (one,1) (eight,8) (four,4)

Виж <http://www.cplusplus.com/doc/tutorial/files/> за повече подробности.

Резултати от тестването на класа *BinarySearchTree*:

```
=====

// Name      : Trees.cpp
// Version   : 01.2012
// Description : Binary trees
=====

#include "BinaryTreeTester.h"

int main() {
    //BinaryTreeTester().testBTree();
    BinaryTreeTester().testBSTree();
    return 0;
}

BinarySearchTree: Test 1
Create tree of integers
insert 3377
insert -3377
insert 15713
```

```
insert -15713
insert 17350
insert -17350
tree contains 17350
the result is : true
tree remove 17350
BSTree: Remove not implemented!
Test tree and write results to the file cout
    Testing...
Size of the tree: 6
Height of the tree: 4
    Tree inorder traversal:
-17350
-15713
-3377
3377
15713
17350

Done!
BinaryTree<T> newTree1=tree;
BTree: Copy constructor
Test newTree1 and write results to the file cout
    Testing...
Size of the tree: 6
Height of the tree: 4
    Tree inorder traversal:
-17350
-15713
-3377
3377
15713
17350

Done!
BinaryTree<T> newTree2; newTree2=tree;
BTree: Assignment operator
Test newTree2 and write results to the file cout
    Testing...
Size of the tree: 6
Height of the tree: 4
    Tree inorder traversal:
-17350
-15713
-3377
3377
15713
17350

Done!
BinarySearchTree: Test 2
Create tree of strings
Read text from the file D:\UserFolder\CPPProjects2\Trees\src\testTreeData1.txt
Test and write results to the file
D:\UserFolder\CPPProjects2\Trees\src\testTreeResult1.txt
BinarySearchTree: Test 3
Create tree of Pairs<string,string>
Read text from the file D:\UserFolder\CPPProjects2\Trees\src\testTreeData2.txt
Test and write results to the file
D:\UserFolder\CPPProjects2\Trees\src\testTreeResult2.txt
```

Файл testTreeData1.txt

Word2
Word10
Word7
Word1

Файл testTreeResult1.txt

Testing...
Size of the tree: 10
Height of the tree: 6
Tree inorder traversal:

Word5
Word7
Word2
Word9
Word8
Word7

Word1
Word10
Word2
Word2
Word5
Word7
Word7
Word8
Word9

Done!

Файл testTreeData2.txt

zero@0
one@1
two@2
three@3
four@4
five@5
six@6
seven@7
eight@8
nine@9

Файл testTreeResult2.txt

Testing...
Size of the tree: 10
Height of the tree: 6
Tree inorder traversal:
(eight,8)
(five,5)
(four,4)
(nine,9)
(one,1)
(seven,7)
(six,6)
(three,3)
(two,2)
(zero,0)

Done!

Литература

1. Deitel H, C++ How to Program, Fifth Edition, Prentice Hall, 2005
2. Harbison S., Guy L. Steele, C: A Reference Manual, Fifth Edition, Prentice Hall, 2002
3. Huss E. , The C Library Reference Guide, 1997

http://www.acm.uiuc.edu/webmonkeys/book/c_guide/

4. Plauger P., The Standard C Library, Prentice Hall, 1992
5. Preiss B., Data Structures and Algorithms with Object-Oriented Design Patterns in C++, Wiley, 1998
6. Stroustrup B., The C++ programming Language, Addison-Wesley, 1988
7. Wiener R., L. Pinson, An Introduction to Object-Oriented Programming and C++, Addison-Wesley, 1988
8. Азълов П., Обектно-ориентирано програмиране. Структури от данни и STL, Сиела, 2008
9. Богданов Д., И. Мустакеров, Език за програмиране С, Техника, 1991
10. Керниган Б., Ритчи Д., Д. Фьюэр, Язык программирования Си. Задачи по языку Си, Пер. с англ., М.: Финансы и статистика, 1985
11. Кнут Д., Искусство программирования для ЭВМ, т. 1, Основные алгоритмы, Пер. с англ., М.: Мир, 1976
12. Кнут Д., Искусство программирования для ЭВМ, т. 2, Получисленные алгоритмы, Пер. с англ., М.: Мир, 1977
13. Кнут Д., Искусство программирования для ЭВМ, т. 3, Сортировка и поиск, Пер. с англ., М.: Мир, 1978
14. Липман Ст., Езикът C++ в примери, Пер. с англ., Колхида Трейд, София, 1993
15. Стандартната библиотека на C++ <http://www.cplusplus.com/reference/>
16. Страуstrup Б. (создатель C++), Язык программирования C++. Специальное издание, Пер. с англ., М.: БИНОМ, 2011
17. Транбле Ж., П. Соренсон, Введение в структуры данных, Пер. с англ., М.: Машиностроение, 1982
18. Хорстман К., Принципи на програмирането със C++, Пер. с англ., ИК Софтех, София 2000