Lab\_7

Utwórz nowy projekt, który zawiera:

* Klasę-szablon my\_vect (file my\_vect.h)
* Klasę my\_mess dla obsługi błędów I komunikatów (my\_mess.h, my\_mess.cpp)
* Klasę interfejsu, która łączy pomiędzy sobą szablon i dane oraz udostępnia użytkownikowi interfejs na poziomie aplikacji konsolowej (my\_interf.h, my\_interf.cpp).
* Klasę danych my\_coord (my\_coord.h, my\_coord.cpp)

Klasa my\_vect:

T \*dat; //pointer to the data of generic type, array of container

size\_t ndim; //dimension of array to be allocated dynamically

size\_t last; //index of the first empty position in array

my\_mess msg; //declaration of the message object

public:

my\_vect(size\_t dm); //constructor - allocates memory for array of T type

//on dm elements.

my\_vect() { dat = NULL; ndim = last = 0; }

~my\_vect(); //release of array

void init(size\_t dm); //allocates memory for array with dm elements,

//if dat = NULL (after default constructor)

T \*get\_begin() //returns pointer T \* to the begin of array

T \*get\_end(); //returns pointer T \* to the dat[last]

void disp(); //displays an array to monitor

void push(const T &ob); //add an element of the type T to first empty position

//of array

T \*pop(); //returns pointer to the last element of the filled

//part of array or NULL, if an array is empty.

T & operator [] (const size\_t ind); //0 <= ind < last

void remove(size\_t ind); //erases an element of array with index ind and

//compresses array

private:

bool alloc(); //dynamically allocates of array

bool realloc(); //enlarge the dimension ndim of array and reallocate

//memory.

Dodaj realizację klasy szablonu i umieść funkcję szablonu Find (...) w tym pliku.

Napisz obsługę błędów i komunikatów w plikach my\_mess.cpp i my\_mess.h. Możesz skorzystać się z przykładu do wykładów W29 albo wprowadzić swój własny system obsługi.

my\_mess.h:

class my\_mess

{

public:

enum MY\_MESSAGE

{

ERR\_ALLOC\_MEM, //memory allocation error

WARN\_ARR\_FULL, //array is full (last > ndim)

WARN\_ARR\_EMPT, //array is empty (last = 0)

WARN\_ARR\_UNKN, //invalid operation (interface)

TOT\_NUMB //total number of messages

};

static char \*strtab[TOT\_NUMB]; //array of messages

public:

void mess(enum MY\_MESSAGE ms); // When this function is called, the message

//appears on the screen. If this is an error,

//the calculation must be finished, if not,

//continue. ms - enumeration member that gives

//the message number};

my\_mess.cpp:

char \*my\_mess::strtab[] =

{

"E memory alloc error", //ERR\_ALLOC\_MEM

"W array is full", //WARN\_ARR\_FULL

"W array is empty", //WARN\_ARR\_EMPT

"W invalid operation" //WARN\_ARR\_UNKN

};

void my\_mess::mess(enum MY\_MESSAGE ms)

{

if(strtab[ms][0] == 'E')

{

cout << "ERROR: " << &strtab[ms][1] << endl;

system("pause");

exit(1);

}

else if(strtab[ms][0] == 'W')

{

cout << "WARNING: " << &strtab[ms][1] << endl;

}

}

Przykładowy wariant klasy interfejsu. Konieczne jest użycie przeciążenia operatorów <<, >> przy wprowadzeniu/wyprowadzeniu obiektów danych do/z strumieni cin, cout.

my\_interf.h:

#include "my\_vect.h"

#include "my\_coord.h"

class my\_interf

{

enum MY\_INTERF

{

MY\_INTERF\_PUSH,

MY\_INTERF\_POP,

MY\_INTERF\_DISP,

MY\_INTERF\_FIND,

MY\_INTERF\_FINISH,

MY\_INTERF\_TOT

};

my\_vect<my\_coord> vect;

static char \*str\_interf[]; // messages displayed on the monitor

my\_mess msg;

public:

bool run;

my\_interf();

my\_interf(size\_t dim);

void menu();

void push();

void pop();

void disp();

void find();

void finish();

void defaul();

};

my\_interf.cpp:

my\_interf::my\_interf(size\_t dim)

{

vect.init(dim);

run = true;

}

void my\_interf::menu()

{

int i;

for(i=0; i<MY\_INTERF\_TOT; i++)

{

cout << str\_interf[i] << endl;

}

}

void my\_interf::push()

{

my\_coord ob(0,0);

cin >> ob;

vect.push(ob);

}

void my\_interf::pop()

{

my\_coord \*ptr = NULL;

ptr = vect.pop();

if(ptr)

{

cout << \*ptr;

}

else

{

msg.mess(my\_mess::WARN\_ARR\_EMPT);

}

}

void my\_interf::disp()

{

vect.disp();

}

void my\_interf::find()

{

my\_coord ob(0, 0);

my\_coord \*ptr = NULL;

cout << "input x, y - object for search\n";

cin >> ob;

ptr = vect.get\_begin();

size\_t dist;

while(ptr)

{

ptr = Find(ptr, vect.get\_end(), ob);

if(ptr)

{

dist = ptr-vect.get\_begin();

cout << "it = " << dist << " " << \*ptr;

ptr++;

}

else

cout << "search end\n";

}

}

Klasa danych my\_coord. Tą klasę trzeba uzupełnić.

class my\_coord

{

double \*pcoord; //pcoord[0] – x coordinate; pcoord[1] – y coordinate

my\_mess msg;

public:

my\_coord() { alloc(); pcoord[0] = pcoord[1] = 0; }

my\_coord(double x, double y);

~my\_coord();

private:

void alloc();

};

Funkcja main:

int \_tmain(int argc, \_TCHAR\* argv[])

{

int op; //operacje

my\_interf ifc(1000);

while(ifc.run)

{

ifc.menu();

cin >> op;

switch(op)

{

case MY\_INTERF\_PUSH: //push

ifc.push();

break;

case MY\_INTERF\_POP:

ifc.pop();

break;

case MY\_INTERF\_DISP:

ifc.disp();

break;

case MY\_INTERF\_FIND:

ifc.find();

break;

case MY\_INTERF\_FINISH:

ifc.finish();

break;

default:

ifc.defaul();

};

}

system("pause");

return 0;

}

Jako wzorzec, można użyć przykłady do wykładów W23, W29 (wyjątki, praca z plikiem binarnym).