**Príklad 9.2**

*Zadanie (na samostatnú prácu)*

Rozšírte zdrojový kód riešiaci zadanie príkladu 9.1 tak, aby program vytvorený

z tohto rozšíreného zdrojového kódu dokázal vytvoriť z existujúceho zoznamu

reverzný (otočený) zoznam a zobraziť takýto zoznam.

Pre takéto rozšírenie funkcionality Vášho programu potrebujete pridať verejnú

členskú funkciu s nasledovnou deklaráciou

void reverse();

do triedy *IntList* a následne vytvoriť jej definíciu.

Príklad vstupu a výstupu takéhoto programu

#include "triedy.h"

// definicia parametrickeho konstruktora triedy 'Item'

Item::Item(int value, Item \*item) : m\_value(value)

{

if (!item) //ak je hodnota vazobn. clena prvku zoznamu (ukazovatela na nasledujuci prvok) == NULL,

m\_next = NULL; //tak potom inicializujeme instancnu premennu objektu tohto prvku zoznamu,

//ukazovatel na nasledujuci prvok 'm\_next', na NULL

else

{

m\_next = item->m\_next;

item->m\_next = this;

}

}

// definicia clenskej funkcie, ktora vlozi na koniec zoznamu 1 prvok s hodnotou 'value'

void IntList::insert\_end(int value)

{

if (!m\_end) //ak je zoznam prazdny,

m\_end = m\_front = new Item(value); //tak novy prvok vkladame do zoznamu ako prvy a zaroven

//aj ako posledny,

else

m\_end = new Item(value, m\_end); //inak vlozime novy prvok na koniec zoznamu

bump\_up();

}

// definicia clenskej funkcie, ktora vlozi na zaciatok zoznamu 1 prvok s hodnotou 'value'

void IntList::insert\_front(int value)

{

Item \*ptr = new Item(value); //vytvorenie ukazovatela 'ptr' na novy prvok zoznamu s hodnotou

//datoveho clena 'value', ktory vkladame do tohto zoznamu

if (!m\_front) //ak je zoznam prazdny,

m\_front = m\_end = ptr; //tak novy prvok vkladame ako prvy (a zaroven aj ako posledny) do

//zoznamu

else //inak

{

ptr->next(m\_front); //novy prvok vkladame pred prvy prvok zoznamu

m\_front = ptr;

}

bump\_up();

}

// definicia clenskej funkcie, ktora vlozi 1 prvok s hodnotou 'value' do zoznamu USPORIADANE

void IntList::insert\_order(int value)

{

Item \*ptr = new Item(value); //vytvorenie ukazovatela 'ptr' na novy prvok zoznamu s hodnotou

//datoveho clena 'value', ktory vkladame do tohto zoznamu USPORIADANE

if (!m\_front) //ak je zoznam prazdny,

m\_front = m\_end = ptr; //tak novy prvok vkladame ako prvy do zoznamu,

else //inak hladame vhodne miesto pre vlozenie prvku

{

Item \*pred = NULL, \*po = m\_front;

enum { HLADAJ, UKONCI } stav = HLADAJ; // premenna 'stav' je vymenovaneho typu

while ((stav == HLADAJ) && (po != 0))

if (po->value() >= value)

stav = UKONCI; // Vhodne miesto najdene!

else // ak sa vhodne miesto pre vlozenie prvku nenaslo, tak sa presuvame dalej v zozname pri

// jeho hladani

{

pred = po; // Ukazovatele "pred" a "po"

po = po->next(); // si zapamataju miesto vlozenia.

}

if (pred == NULL) // Vlozenie noveho prvku na zaciatok zoznamu.

{

m\_front = ptr;

ptr->next(po);

}

else if (po == NULL) // Vlozenie noveho prvku na koniec zoznamu.

{

m\_end->next(ptr);

ptr->next(NULL);

m\_end = ptr;

}

else // Vlozenie noveho prvku medzi prvky zoznamu,

{ // na ktore teraz ukazuju ukazovatele 'pred' a 'po'

pred->next(ptr);

ptr->next(po);

}

}

bump\_up();

}

// definicia clenskej funkcie, ktora spoji 2 zoznamy 'za' a 'zb' NEusporiadanie

IntList& IntList::join(IntList &za, IntList &zb)

{

IntList \*z = new IntList; // vytvorenie ukazovatela 'z' na novy objekt triedy 'IntList'. Do tohto noveho

// objektu sa spajaju 2 zoznamy 'za' a 'zb'

Item \*p1 = za.front(); //vytvorenie pomocneho ukazovatela 'p1' na dat. typ 'Item', do ktoreho si ulozime

//ukazovatel na 1. prvok zoznamu 'za'

Item \*p2 = zb.front(); //vytvorenie pomocneho ukazovatela 'p2' na dat. typ 'Item', do ktoreho si ulozime

//ukazovatel na 1. prvok zoznamu 'zb'

//NEusporiadane vkladanie (kopirovanie) prvkov zoznamu 'za' do noveho zoznamu s ukazovatelom 'z'

while (p1 != NULL)

{

z->insert\_end(p1->value());

p1 = p1->next();

bump\_up();

}

//NEusporiadane vkladanie (kopirovanie) prvkov zoznamu 'zb' do noveho zoznamu s ukazovatelom 'z'

while (p2 != NULL)

{

z->insert\_end(p2->value());

p2 = p2->next();

bump\_up();

}

return (\*z);

}// definicia clenskej funkcie, ktora spoji 2 zoznamy 'za' a 'zb' usporiadanie

IntList& IntList::join\_ordered(IntList &za, IntList &zb)

{

IntList \*z = new IntList;;

Item \*p1 = za.front();

Item \*p2 = zb.front();

// usporiadane vkladanie (kopirovanie) prvkov zoznamu 'za' do noveho zoznamu s ukazovatelom 'z'

while (p1 != NULL)

{

z->insert\_order(p1->value());

p1 = p1->next();

bump\_up();

}

// usporiadane vkladanie (kopirovanie) prvkov zoznamu 'zb' do noveho zoznamu s ukazovatelom 'z'

while (p2 != NULL)

{

z->insert\_order(p2->value());

p2 = p2->next();

bump\_up();

}

return (\*z);

}

// Operatorova funkcia prepisaneho operatora '+' pre triedu 'IntList', ktora vykona neusporiadane

// spojenie dvoch zoznamov do vysledneho zoznamu pomocou operatora '+'

IntList& operator+(IntList &za, IntList &zb)

{

IntList \*z = new IntList;

Item \*p1 = za.front();

Item \*p2 = zb.front();

//NEusporiadane vkladanie (kopirovanie) prvkov zoznamu 'za' do noveho zoznamu s ukazovatelom 'z'

while (p1 != NULL)

{

z->insert\_end(p1->value());

p1 = p1->next();

}

//NEusporiadane vkladanie (kopirovanie) prvkov zoznamu 'zb' do noveho zoznamu s ukazovatelom 'z'

while (p2 != NULL)

{

z->insert\_end(p2->value());

p2 = p2->next();

}

return (\*z);

}

// definicia clenskej funkcie, ktora zmaze prvy prvok zoznamu

void IntList::remove\_front()

{

if (m\_front) // ak zoznam nie je prazdny, tzn. 'm\_front != NULL',

{

Item \*ptr = m\_front; // tak vytvorime pomocny ukazovatel 'ptr' na datov. typ 'Item' a naplnime ho

// ukazovatelom na 1. prvok zoznamu 'm\_front'

m\_front = m\_front->next(); //ukazovatel na 1. prvok zoznamu 'm\_front' prepiseme ukazovatelom na

//dalsi prvok zoznamu, pretoze 1. prvok zoznamu chceme zmazat

bump\_down(); delete ptr; // mazeme 1. prvok zoznamu, na ktory ukazuje ukazovatel 'ptr'

}

}

// definicia clenskej funkcie, ktora zmaze vsetky prvky zoznamu

void IntList::remove\_all()

{

while (m\_front)

remove\_front();

m\_size = 0;

m\_front = m\_end = NULL;

}

// definicia clenskej funkcie, ktora zisti pocet vyskytov prvku s datovou hodnotou 'value' v zozname

int IntList::numb\_of\_occurr(int value)

{

int numb\_occurr = 0;

Item \*ptr = m\_front;

while (ptr)

{

if (ptr->value() == value)

numb\_occurr++;

ptr = ptr->next();

}

return numb\_occurr;

}

// definicia clenskej funkcie, ktora zobrazi datove hodnoty prvkov zoznamu na konzolu

void IntList::display()

{

cout << " (velkost " << m\_size << ") ( ";

Item \*ptr = m\_front;

while (ptr)

{

cout << ptr->value() << " ";

ptr = ptr->next();

}

cout << ")\n";

}

void IntList::reverse() {

Item \*currently = m\_front, \*previous = NULL, \*next = NULL;

while (currently != NULL)

{

next = currently->next();//dalsi ukazuje nato naco ukazuje m\_next prvka na ktory ukazuje aktualny

currently->next(previous);//m\_next prvku na ktory ukazuje aktualny bude ukazovat na prvok na ktory ukazuje predchadzajuci

previous = currently; //bude ukazovat na prvok na ktory ukazuje aktualny

currently = next; //dalsi bude ukazovat na prvok na ktory ukazuje aktualny

}

m\_front = previous;

}

vlozte zakladnu velkost zoznamov, ktore chcete vytvarat: 5

vlozte 5 prvkov, ktore budu neusporiadane vlozene do zoznamu Z1: 9 8 7 10 3

Zoznam Z1 = (velkost 5) ( 9 8 7 10 3 )

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Vlozte 5 prvkov, ktore budu USPORIADANE vlozene do zoznamu Z2:

-9 4 12 -11 0

Zoznam Z2 = (velkost 5) ( -11 -9 0 4 12 )

Vlozte 3 cisla, ktore budu USPORIADANE vlozene do zoznamu Z2:

-12 -10 5

Zoznam Z2 s USPORIADANE vlozenymi cislami -12, -10 a 5

Zoznam Z2 = (velkost 8) ( -12 -11 -10 -9 0 4 5 12 )

Vlozte cislo, ktore sa ma hladat v zozname Z2: -11

Pocet vyskytov cisla -11 v zozname Z2: 1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

prvky LEN spojeneho zoznamu Z = Z1 + Z2 prepisanym operatorom '+':

(velkost 13) ( 9 8 7 10 3 -12 -11 -10 -9 0 4 5 12 )

prvky LEN spojeneho zoznamu Z (= Z + Z1) instancnou funkciou 'join':

(velkost 18) ( 9 8 7 10 3 -12 -11 -10 -9 0 4 5 12 9 8 7 10 3 )

prvky USPORIADANE spojeneho zoznamu Z\_usp (= Z + Z1) instancnou funkciou

'join\_ordered':

(velkost 23) ( -12 -11 -10 -9 0 3 3 3 4 5 7 7 7 8 8 8 9 9 9 10 10 10 12 )

Reverzovany (otoceny) zoznam Z:

(velkost 18) ( 3 10 7 8 9 12 5 4 0 -9 -10 -11 -12 3 10 7 8 9 )

Reverzovany (otoceny) zoznam Z\_usp:

(velkost 23) ( 12 10 10 10 9 9 9 8 8 8 7 7 7 5 4 3 3 3 0 -9 -10 -11 -12 )