**//seminar 3-lista dubla**

void **conversieListaDublaVector**(nodls \*cap, produs \*\*vect, int \*nr)

{

nodls \*temp = cap;

while(temp!=NULL)

{

vect[\*nr] = temp->inf;

(\*nr)++;

nodls \*temp2 = temp->next;

free(temp);

temp = temp2;

}

}

**//sem 4 ciurea- stiva**

void **conversieStivaListaSimpla**(nodStiva\*\* varf, nodLista\*\*cap)

{

carte val;

while(pop(varf, &val)==0)

{

inserareLista(cap, val);

}

}

**//seminar 5 ciurea**

void **conversieCoadaStiva**(nodCoada\*\* prim, nodCoada\*\* ultim, nodStiva \*\*varf)

{

student s;

while(get(prim, ultim, &s)==0)

push(varf, s);

}

void **conversieCoadaVector**(nodCoada\*\* prim, nodCoada\*\* ultim, student \*vect, int \*nr)

{

student s;

while(get(prim, ultim, &s)==0)

{

vect[\*nr] = s;

(\*nr)++;

}

}

**//seminar 9 ciurea- arbori**

void **conversieArboreVector**(nodarb \*rad, student \*vect, int \*nr)

{

if(rad!=NULL)

{

/\*vect[\*nr] = rad->inf;

(\*nr)++;

conversieArboreVector(rad->left, vect, nr);

conversieArboreVector(rad->right, vect, nr);\*/

vect[\*nr].cod = rad->inf.cod;

vect[\*nr].nume = (char\*)malloc((strlen(rad->inf.nume)+1)\*sizeof(char));

strcpy(vect[\*nr].nume, rad->inf.nume);

vect[\*nr].medie = rad->inf.medie;

(\*nr)++;

nodarb \*st = rad->left;

nodarb \*dr = rad->right;

free(rad->inf.nume);

free(rad);

conversieArboreVector(st, vect, nr);

conversieArboreVector(dr, vect, nr);

}

}

**//CONVERSII LISTA SIMPLA**

void **conversieListaVector**(nodls \*cap, produs \*vect, int \*nr)

{

nodls \*temp = cap;

while (temp->next != NULL)

{

vect[\*nr] = temp->inf;

(\*nr)++;

nodls \*temp2 = temp->next;

free(temp);

temp = temp2;

}

vect[\*nr] = temp->inf;

(\*nr)++;

free(temp);

}

void **conversieListaSimplaCoada**(nodls\*cap, nodcoada\*\*capc, nodcoada\*\*coada)

{

nodls\*temp = cap;

while (temp != NULL) {

put(capc, coada, temp->inf);

nodls \*temp2 = temp->next;

free(temp);

temp = temp2;

}

//\*capd = inserarelistadubla(tempd, &tempc, temp->inf);

free(temp);

}

void **conversieListaSimplaDubla**(nodls\*cap, nodld\*\*capd, nodld\*\*coada) {

nodls\*temp = cap;

while (temp != NULL) {

\*capd = inserarelistadubla(\*capd, coada, temp->inf);

nodls \*temp2 = temp->next;

free(temp);

temp = temp2;

}

//\*capd = inserarelistadubla(tempd, &tempc, temp->inf);

free(temp);

}

void **conversieListaSimplaCirculara**(nodls\*cap, nodlc\*\*capc, nodlc\*\*coada) {

nodls\*temp = cap;

while (temp != NULL) {

\*capc = inserarelistacirculara(\*capc, coada, temp->inf);

nodls \*temp2 = temp->next;

free(temp);

temp = temp2;

}

free(temp);

}

void **conversieListaSimplaStiva**(nodls\*cap, nodstiva\*\*caps) {

nodls\*temp = cap;

while (temp != NULL) {

push(caps, temp->inf);

nodls \*temp2 = temp->next;

free(temp);

temp = temp2;

}

free(temp);

}

**//CONVERSII LISTA DUBLA**

void **conversieListaDublaCirculara**(nodld\*cap, nodlc\*\*capc, nodlc\*\*coadac) {

nodld\*temp = cap;

while (temp != NULL) {

\*capc = inserarelistacirculara(\*capc, coadac, temp->inf);

nodld \*temp2 = temp->next;

free(temp);

temp = temp2;

}

free(temp);

}

void **conversieListaDublaSimpla**(nodld\*cap, nodls\*\*caps) {

nodld\*temp = cap;

while (temp != NULL) {

\*caps = inserarelistasimpla(\*caps, temp->inf);

nodld \*temp2 = temp->next;

free(temp);

temp = temp2;

}

free(temp);

}

void **conversieListaDublaVector**(nodld \*cap, produs \*vector, int \*nr) {

nodld \*temp = cap;

while (temp != NULL) {

vector[\*nr] = temp->inf;

(\*nr)++;

nodld \*temp2 = temp->next;

free(temp);

temp = temp2;

}

}

void **conversieListaDublaStiva**(nodld\*cap, nodstiva\*\*caps) {

nodld\*temp = cap;

while (temp != NULL) {

push(caps, temp->inf);

nodld \*temp2 = temp->next;

free(temp);

temp = temp2;

}

free(temp);

}

void **conversieListaSimplaCoada**(nodld\*cap, nodcoada\*\*capc, nodcoada\*\*coada)

{

nodld\*temp = cap;

while (temp != NULL) {

put(capc, coada, temp->inf);

nodld \*temp2 = temp->next;

free(temp);

temp = temp2;

}

//\*capd = inserarelistadubla(tempd, &tempc, temp->inf);

free(temp);

}

**//CONVERSII LISTA CIRCULARA**

void **conversieListaCircularaSimpla**(nodlc\*cap, nodls\*\*caps) {

nodlc\*temp = cap;

while (temp->next != cap) {

\*caps = inserarelistasimpla(\*caps, temp->inf);

nodlc \*temp2 = temp->next;

free(temp);

temp = temp2;

}

\*caps = inserarelistasimpla(\*caps, temp->inf);

nodlc \*temp2 = temp->next;

free(temp);

}

void **conversieListaCircularaDubla**(nodlc\*cap, nodld\*\*capd, nodld\*\*coada) {

nodlc\*temp = cap;

while (temp->next != cap) {

\*capd = inserarelistadubla(\*capd, coada, temp->inf);

nodlc \*temp2 = temp->next;

free(temp);

temp = temp2;

}

\*capd = inserarelistadubla(\*capd, coada, temp->inf);

free(temp);

}

void **conversieListaVecto**r(nodlc \*cap, produs \*vector, int \*nr) {

nodlc \*temp = cap;

while (temp->next != cap) {

vector[\*nr] = temp->inf;

(\*nr)++;

nodlc \*temp2 = temp->next;

free(temp);

temp = temp2;

}

vector[\*nr] = temp->inf;

(\*nr)++;

free(temp);

}

void conversieListaCircularaCoada(nodlc\*cap, nodcoada\*\*capc, nodcoada\*\*coada)

{

nodlc\*temp = cap;

while (temp->next != cap) {

put(capc, coada, temp->inf);

nodlc \*temp2 = temp->next;

free(temp);

temp = temp2;

}

put(capc, coada, temp->inf);

free(temp);

}

void **conversieListaCircularaStiva**(nodlc\*cap, nodstiva\*\*caps) {

nodlc\*temp = cap;

while (temp->next != cap) {

push(caps, temp->inf);

nodlc \*temp2 = temp->next;

free(temp);

temp = temp2;

}

push(caps, temp->inf);

free(temp);

}

**//CONVERSII STIVA**

void **conversieStivaVector**(nodstiva \*\*varf, produs \*vect, int \*nr)

{

produs val;

while (pop(varf, &val) == 0) //atat timp cat se extrage cu succes din stiva se insereaza in vector

{

vect[\*nr] = val;

(\*nr)++;

}

}

void **conversieStivaListaSimpla**(nodstiva\*\* varf, nodlista\*\*cap)

{

produs val;

while (pop(varf, &val) == 0)

{

\*cap=inserareLista(\*cap, val);

}

}

void **conversieStivaListaDubla**(nodstiva\*\* varf, nodld\*\*cap,nodld\*\*coada)

{

produs val;

while (pop(varf, &val) == 0)

{

\*cap = inserarelistadubla(\*cap,coada,val);

}

}

void **conversieStivaListaCirculara**(nodstiva\*\* varf, nodlc\*\*cap, nodlc\*\*coada)

{

produs val;

while (pop(varf, &val) == 0)

{

\*cap = inserarelistacirculara(\*cap, coada, val);

}

}

void **conversieStivaCoada**(nodstiva\*\* varf, nodcoada\*\*cap,nodcoada\*\*coada)

{

produs val;

while (pop(varf, &val) == 0)

{

put(cap, coada, val);

}

}

**//CONVERSII COADA**

void **conversieCoadaVector**(nodcoada \*\*prim, nodcoada \*\*ultim, produs \*vect, int \*nr)

{

produs s;

while (get(prim, ultim, &s) == 0)

{

vect[\*nr] = s;

(\*nr)++;

}

}

void **conversieCoadaListaSimpla**(nodcoada\*\*varf,nodcoada \*\*ultim, nodls\*\*cap) {

produs s;

while (get(varf,ultim, &s) == 0) {

\*cap = inserareListaSimpla(\*cap, s);

}

}

void **conversieCoadaListaDubla**(nodcoada\*\*varf, nodcoada \*\*ultim, nodld\*\*cap,nodld\*\*coada) {

produs s;

while (get(varf, ultim, &s) == 0) {

\*cap = inserarelistadubla(\*cap,coada, s);

}

}

void **convesieCoadaListaCirculara**(nodcoada\*\*varf, nodcoada \*\*ultim, nodlc\*\*cap, nodlc\*\*coada) {

produs s;

while (get(varf, ultim, &s) == 0) {

\*cap = inserarelistacirculara(\*cap, coada, s);

}

}

void conversieCoadaStiva(nodcoada\*\*varf, nodcoada\*\*ultim, nodstiva\*\*cap) {

produs s;

while (get(varf, ultim, &s) == 0) {

push(cap, s);

}

}

**//CONVERSII ARBORE**

void **conversieArboreListaSimpla**(nodarb \*rad, nodListaSimpla \*\*cap) {

if (rad != NULL) {

nodListaSimpla \*nou = (nodListaSimpla\*)malloc(sizeof(nodListaSimpla));

nou->inf.cod = rad->inf.cod;

nou->inf.nume = (char\*)malloc((strlen(rad->inf.nume) + 1) \* sizeof(char));

strcpy(nou->inf.nume, rad->inf.nume);

nou->inf.medie = rad->inf.medie;

nou->next = NULL;

if (\*cap == NULL)

\*cap = nou;

else {

nodListaSimpla \*temp = \*cap;

while (temp->next)

temp = temp->next;

temp->next = nou;

}

nodarb \*st = rad->left;

nodarb \*dr = rad->right;

free(rad->inf.nume);

free(rad);

conversieArboreListaSimpla(st, cap);

conversieArboreListaSimpla(dr, cap);

}

}

void **conversieArboreListaDubla**(nodarb \*rad, nodListaDubla \*\*cap, nodListaDubla \*\*coada) {

if (rad != NULL) {

nodListaDubla \*nou = (nodListaDubla\*)malloc(sizeof(nodListaDubla));

nou->inf.cod = rad->inf.cod;

nou->inf.nume = (char\*)malloc((strlen(rad->inf.nume) + 1) \* sizeof(char));

strcpy(nou->inf.nume, rad->inf.nume);

nou->inf.medie = rad->inf.medie;

nou->next = NULL;

nou->prev = NULL;

if (\*cap == NULL) {

\*cap = nou;

\*coada = nou;

}

else {

nodListaDubla \*temp = \*cap;

while (temp->next)

temp = temp->next;

temp->next = nou;

nou->prev = temp;

\*coada = nou;

}

nodarb \*st = rad->left;

nodarb \*dr = rad->right;

free(rad->inf.nume);

free(rad);

conversieArboreListaDubla(st, cap, coada);

conversieArboreListaDubla(dr, cap, coada);

}

}

void **conversieArboreVector**(nodarb \*rad, student \*vect, int \*nr)

{

if (rad != NULL)

{

/\*vect[\*nr] = rad->inf;

(\*nr)++;

conversieArboreVector(rad->left, vect, nr);

conversieArboreVector(rad->right, vect, nr);\*/

vect[\*nr].cod = rad->inf.cod;

vect[\*nr].nume = (char\*)malloc((strlen(rad->inf.nume) + 1) \* sizeof(char));

strcpy(vect[\*nr].nume, rad->inf.nume);

vect[\*nr].medie = rad->inf.medie;

(\*nr)++;

nodarb \*st = rad->left;

nodarb \*dr = rad->right;

free(rad->inf.nume);

free(rad);

conversieArboreVector(st, vect, nr);

conversieArboreVector(dr, vect, nr);

}

}

**//dezalocare arbore**

Nod\* dezalocaArbore(Nod\* radacina) {

if (radacina == NULL) {

return NULL;

}

else {

if (radacina->fiuD == NULL && radacina->fiuS == NULL) {

free(radacina->info.denumire);

free(radacina);

return NULL;

}

else {

if (radacina->fiuS == NULL) {

//avem fiu pe dreapta

Nod\* temp = radacina->fiuD;

free(radacina->info.denumire);

free(radacina);

return dezalocaArbore(temp);

}

else if (radacina->fiuD == NULL) {

//avem fiu pe stanga

Nod\* temp = radacina->fiuS;

free(radacina->info.denumire);

free(radacina);

return dezalocaArbore(temp);

}

else {

radacina->fiuD = dezalocaArbore(radacina->fiuD);

radacina->fiuS = dezalocaArbore(radacina->fiuS);

radacina = dezalocaArbore(radacina);

}

}

}

return radacina;

}