# S07 – SD

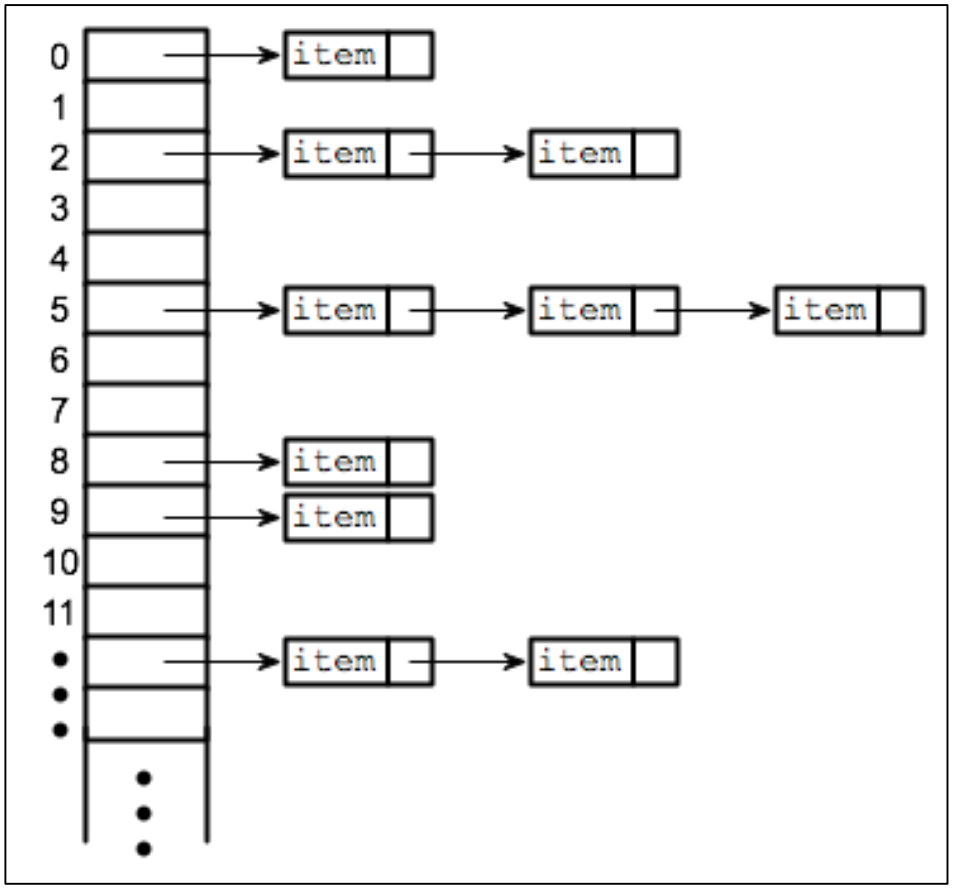
Alin Zamfiroiu
alin.zamfiroiu@csie.ase.ro

# Structuri create

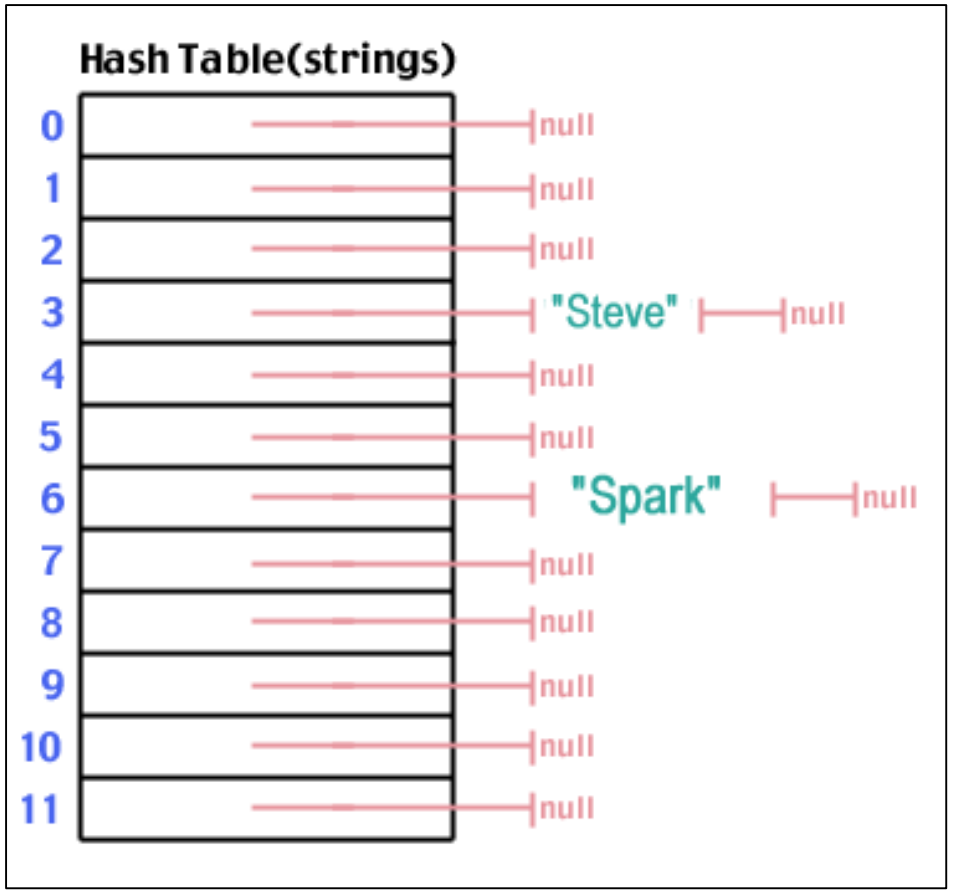
- Se creeaza o structura pentru un vagon cu cod si tipul vagonlui.
- Informatia de tip vagon este informatia dintr-un nod de tip lista.
- Tabelele de dispersie sunt reprezentate de un vector (pointer) de liste (nod*).

```
struct vagon{
    int cod;
    char* tip;
};

struct nod{
    vagon info;
    nod* next;
};

struct HashStruct{
    int dimensiune;
    nod** elemente;
};
```

# Tabele de dispersie




http://4spills.blogspot.ro/




http://www.sparknotes.com/

# Tabele de dispersie

- Sa se scrie functia care creaza si returneaza un HashStruct:
- Se seteaza dimensiunea acestuia;
- Se aloca spatiu pentru vectorul de liste;
- Se initializeaza toate listele cu NULL;
- Se returneaza noua structura creata.

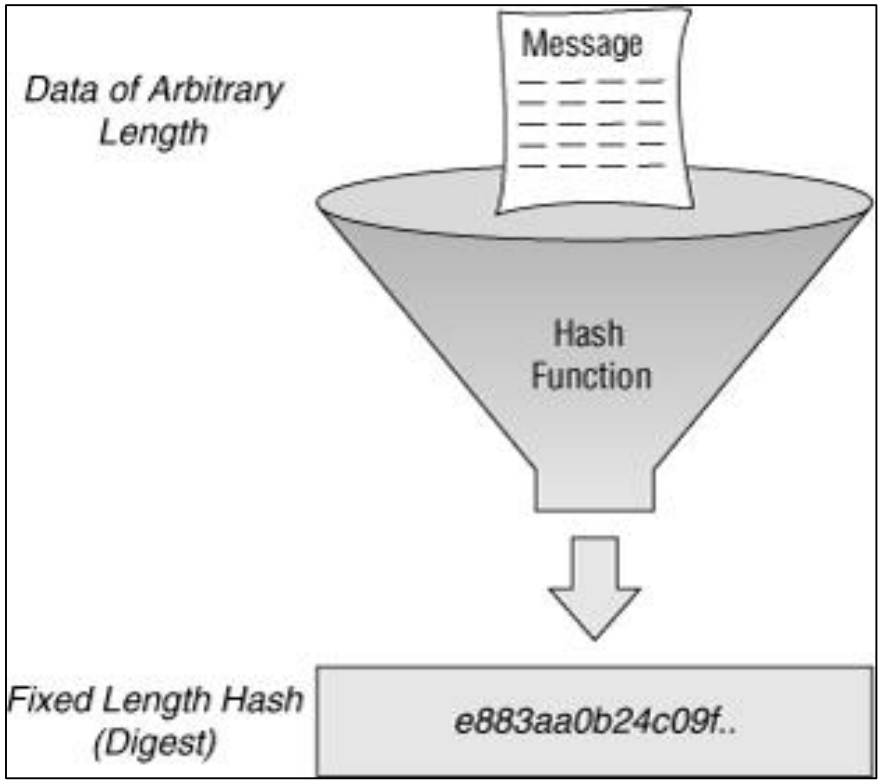
# Alocarea de memorie

```
HashStruct AlocaMemorie()
{
    HashStruct hs;
    hs.dimensiune=50;
    hs.elemente=(nod**)malloc(sizeof(nod*)*hs.dimensiune);
    //hs.elemente=new nod*[hs.dimensiune];
    for(int i=0;i<hs.dimensiune;i++)
    {
        hs.elemente[i]=NULL;
    }
    return hs;
}
```

- Sa se scrie functia de generare a codului hash.
- Functia primeste codul pentru care trebuie sa genereze hash-ul si structura de tip HashStruct pentru obtinerea dimensiunii.

# Functia Hash

```
int functieHash(int cod, HashStruct hs)
{
    return cod%hs.dimensiune;
}
```




http://ciscodocuments.blogspot.ro/

# Functia de inserare

- Functia de inserare primeste ca parametrii structura hash in care se insereaza si elementul de tip vagon care urmeaza a fi inserat.
- Aceasta returneaza pozitia pe care este inserat elementul.
- Pozitia se determina cu ajutorul functiei hash.
- Se creeaza noul nod, si se insereaza la pozitia determinata. Daca exista deja cel putin un nod la acea pozitie vom realiza inserare la sfarsitul listei.

# Functia de inserare

```
int insereazaVagon(HashStruct hs, vagon v)
{
    int pozitie=-1;

    if(v.cod<0)
    {
        return pozitie;
    }
    if(hs.elemente!=NULL)
    {
        pozitie=functieHash(v.cod, hs);
        nod* nod_nou=(nod*)malloc(sizeof(nod));
        nod_nou->next=NULL;
        nod_nou->info=v;
        if(hs.elemente[pozitie]==NULL)
            hs.elemente[pozitie]=nod_nou;
        else
        {
            nod* temp=hs.elemente[pozitie];
            while(temp->next)
                temp=temp->next;
            temp->next=nod_nou;
        }
    }
    return pozitie;
}
```

Ce problema are acest cod?

# Citire din fisier

- Ca sa avem ce sa inseram, implementam functia de citire a unui vagon dintr-un fisier.
- Functia primeste ca si parametru fisierul din care citeste si returneaza un vagon.
- Pentru citire din fisier se foloseste functia fscanf(File file, char* format,…), caruia i se specifica fisierul din care citeste.

# Citire din fisier

```
vagon citesteVagonDinFisier(FILE*f)
{
    char aux[20];
    vagon v;
    fscanf(f, "%d", &v.cod);
    fscanf(f, "%s", &aux);
    v.tip = (char*)malloc(sizeof(char)*(strlen(aux)+1));
    strcpy(v.tip, aux);
    return v;
}
```

```
FILE*f;
f = fopen("vagoane.txt", "r+");

insereazaVagon(hs, citesteVagonDinFisier(f));
```

# Parcurgere HashTables

- Pentru parcurgerea intregii structuri, se parcurge mai intai vectorul si apoi in cazul in care exista lista la pozitia curenta, atunci se parcurge lista si se afiseaza.
- Afisarea se poate face inline sau prin apelul unei functii care va afisa la consola.
- Se poate face si scrierea in fisier. Fisierul se deschide cu “w” pentru scriere si se scrie cu functia fprintf(File file, char* format,…).

# Parcurgere HashTables

```
void afiseazaVagon(vagon v)
{
    printf("%d. %s\n",v.cod,v.tip);
}
```

```
void parcurgere(HashStruct hs)
{
    if(hs.elemente)
    {
        for(int i=0;i<hs.dimensiune;i++)
        {
            nod* temp=hs.elemente[i];
            while(temp)
            {
                afiseazaVagon(temp->info);
                temp=temp->next;
            }
        }
    }
}
```

# Scriere in fisier

```
void AfisareInFisier(HashStruct hs)
{
    FILE *f=fopen("test.txt","w");

    if(hs.elemente)
    {
        for(int i=0;i<hs.dimensiune;i++)
        {
            nod* temp=hs.elemente[i];
            while(temp)
            {
                //afiseazaVagon(temp->info);
                fprintf(f,"%d. %s.\n",temp->info.cod,temp->info.tip);
                temp=temp->next;
            }
        }
    }
    fclose(f);
}
```

# Cautare dupa cod

- Functia pentru cautare in structura dupa cod.
- Functia primeste ca parametri codul vagonului de cautat si structura in care face cautarea.
- Asemanator procesului de inserare se determina valoarea hash a codului de cautat si apoi se cauta la pozitia indicata de pozitia obtinuta din codul hash.
- Daca nu exista se returneaza un vagon cu codul egal cu -1.

# Cautare dupa cod

```
vagon CautaDupaCod(int cod, HashStruct hs)
{
    vagon v;
    v.cod=-1;
    if(cod<0)
    {
        return v;
    }
    if(hs.elemente!=NULL)
    {
        int pozitie=functieHash(cod, hs);
        if(hs.elemente[pozitie]==NULL)
        {
            return v;
        }
        if(hs.elemente[pozitie]->info.cod==cod)
        {
            return hs.elemente[pozitie]->info;
        }
        else
        {
            nod*temp=hs.elemente[pozitie];
            while(temp&& temp->info.cod!=cod)
            {
                temp=temp->next;
            }
            if(temp!=NULL)
                return temp->info;
            else
            {
                return v;
            }
        }
    }
    return v;
}
```

# Stergere structura

- Pentru stergere se parcurge vectorul si se sterg mai intai nodurile, respectiv listele.
- Dupa stergerea tuturor nodurilor se sterge si vectorul de elemente din cadrul structurii.
- Pentru stererea nodurilor se face o functie care sterge recursiv nodurile listei.

# Stergere structura

```
void stergereHash(HashStruct hs)
{
    if(hs.elemente!=NULL)
    {
        for(int i=0;i<hs.dimensiune;i++)
            hs.elemente[i]=stergeLista(hs.elemente[i]);
        free(hs.elemente);
    }
}
```

```
nod* stergeLista(nod* cap)
{
    if(cap)
    {
        cap->next=stergeLista(cap->next);
        free(cap);
        return NULL;
    }
    else
        return NULL;
}
```

# Tema

- Să se realizeze o structură HashTable în care evitarea coliziunilor se realizează prin Linear Probing.