CODE ACADEMY

ОБУЧЕНИЕ ПО ПРОГРАМИРАНЕ

Динамични структури. Linked List

CODE ACADEMY

Динамични структури

- многообразни структури от данни
- всички тези многообразни структури от данни могат да бъдат сведени до списък
- динамичните структури от данни отразяват предметната област, която автоматизираме
- динамичните структури от данни се променят по време на изпълнението и се адаптират в съответствие с потока от данните
- за някои динамични структури, като дървета и графи, има теория и решени математически проблеми, като например: балансиране на дърво, търсене на път в граф, задача за търговския пътник, оцветяване на върховете на граф и други.

Елемент на динамична структура

- елементът на динамична структура съдържа данни и връзки с други елементи
- връзките с другите елементи могат да бъдат указатели, индекс в таблица с връзки или друг начин за описание на връзка между един елемент и друг
- връзките могат да имат посока и тежест; пример: пътища в град, еднопосочни и двупосочни, тежестта е разстоянието

```
struct TDynamicData {
    TData m_data;
    TLink m_link[N];
};
```

CODE ACADEMY

Рекурсивна структура

- структура, в която има указател / указатели, които сочат структура от същия тип
- при добавяне на данни между два възела, връзката между тях се прекъсва и новият възел се "свързва" между тях
- при изтриването на елемент, трябва да се възстанови връзката между неговите съседи

```
struct TRecData {
    TData m_data;
    struct TRecData* m_pLink[N];
};
```

CODE ACADEMY

Списък

- динамична структура, в която елементите са свързани последователно
- достъпът до елементите е последователен, ако търсим даден елемент, в най-лошия случай ще обходим всички елементи
- при добавяне на нов елемент между два елемент прекъсваме връзката между тях и ги свързваме към новия елемент
- при изтриване правим обратно, премахваме елемента и правим връзката между съседите му

```
X1 ---> X2 ---> X3 --->… ---> XN
```

```
X1 ---> X2 ---> X23 ---> X3 --->… ---> XN
```

CODE ACADEMY

Стек и опашка

- стек - използваме списък като добавяме и изтриваме само от началото на списъка (LIFO - Last In First Out)

  `push/pop <=> X1 ---> X2 ---> X3 --->… ---> XN`

- опашка - използваме списък като добавяме елементи в началото на списъка, а ги изтриваме от края на списъка (FIFO - First In First Out)

  `write => X1 ---> X2 ---> X3 --->… ---> XN => read`

CODE ACADEMY

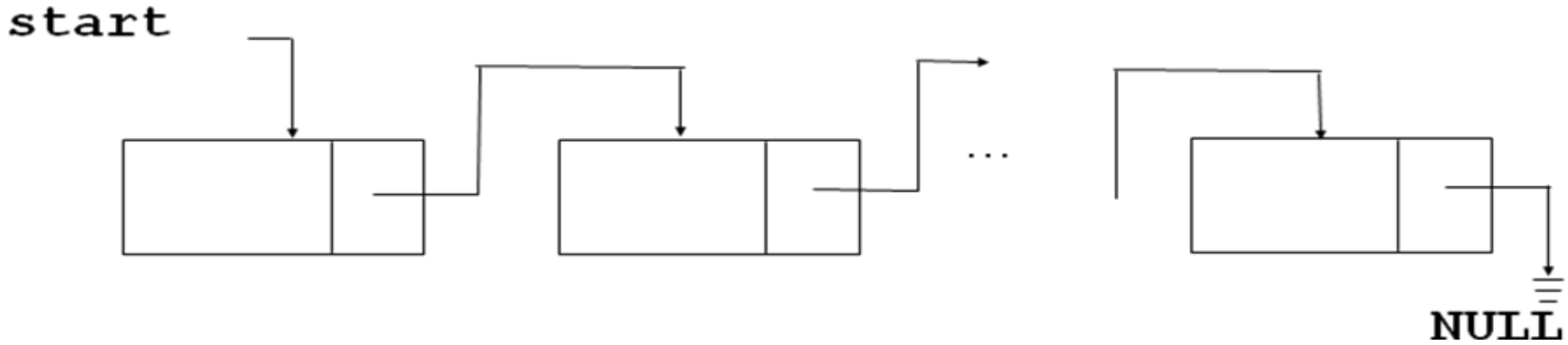
Добри практики

- динамичните структури изискват по-сериозен анализ от статичните такива, преди да започнем да ги използваме трябва да проиграем различни сценарии
- има дебели книги с алгоритми за динамични структури от данни, които е добре да разлистим преди да зациклим
- динамичните структури са гъвкави, но имат ограничения и не пасват на всяка задача, има т.н. design pattern
- когато ползваме динамични структури винаги трябва да мислим за тяхната сериализация и десериализация в / от стринг
- хубаво е да знаем, че сложните динамични структури се обработват бавно - обработващите алгоритми имат експоненциална сложност и отнемат много време и ресурси

CODE ACADEMY

Списък (List)

Списъкът е подредена последователност от еднотипни елементи без определена дисциплина за тяхното включване и изключване. Стекът, опашката и декът могат да се разглеждат като частен случай на структурата списък. Най-простият случай на структурата е този, в който всеки елемент на списъка съдържа указател към следващия елемент:



List Списък – Линейна структура

- Създаване на **структура без тяло в .h файла.**

```
typedef struct node_t node_t;
```

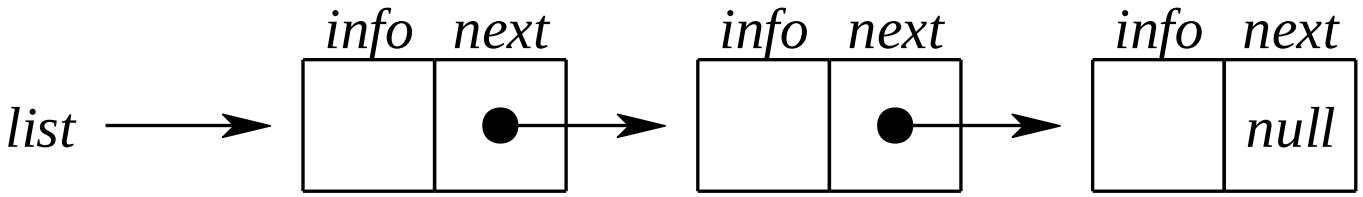
- Разписване **тялото на структурата с данните,** които се съдържат в нея:

```
typedef struct node_t{
        int data;
        node_t *next;
}       node_t;
```

- Създаване на една **глобална променлива от тип node_t Указател към тази структура,която ще е начало на списъка - главата.** Ако тя се загуби, не може да го обхождаме.

```
extern node_t *start;
```

Linked Lists



Linear linked list

Linked list е линейна структура от елементи - данни, наречени възли - **nodes**, където линейната подредба се осъществява с указатели **pointers**.

Всеки **node** се дели на две части:
- информация -info
- адрес към следващия възел – node, т.е. указател

CODE ACADEMY

Свързаните списъци са често срещана алтернатива на масивите при внедряването на структури от данни.

Всеки елемент в свързан списък съдържа елемент от данни от някакъв тип и указател към следващия елемент в списъка.

Лесно е да се вмъкват и изтриват елементи в свързан списък, които не са естествени операции върху масиви, тъй като масивите имат фиксиран размер.

Достъп до елемент в средата на списъка обикновено е O(n), където n е дължината на списъка.

Елемент в свързан списък се състои от структура, съдържаща елемента от данни и указател към друг елемент от свързания списък.

CODE ACADEMY

```
typedef struct list_node {
        typr_elem data;
        struct list_node* next;
} node;
```

Структура от този тип е рекурсивна, тя съдържа указател към друга структура от същия тип и т.н.

Обикновено използваме указател за обхождане на списъка и когато неговата стойност е NULL, това означава, че сме достигнали края на списъка.

Указателят към възел в свързан списък с възли от горния вид е
node*.

Свързаният списък е поредица от възли, започващи с указател head,

сочещ винаги първия възел и завършващ с указател end в края на списъка.

CODE ACADEMY

Създаване на свързан списък

```
#include<stdio.h>
#include<stdlib.h>
struct node{
   int number;
   struct node* next;
};
   struct node * head;
   struct node * current;
   struct node * previous;
int main()
{
   head=malloc(sizeof(struct node));
   head->number=0;
   head->next=NULL;
   current=head;

 for(int i=0;i<3;i++)
   {
      previous=current;
      current=malloc(sizeof(struct node));
      current->number=i+1;
      current->next=NULL;
      previous->next=current;
   }
   for(struct node *p=head; p!=NULL; p=p->next)
   {
      printf("%d\n",p->number);
   }
}
```

CODE ACADEMY

Освобождаване на паметта, заета от свързания списък

```
current=head;
   struct node * tmp;

   while(current){
       tmp=current;
       current=current->next;
       free(tmp);
   }
```

CODE ACADEMY

Задача

Направете свързан списък с информация за 5 човека - име и телефон.

Разпечатайте този списък

Напишете код, който проверява какъв е телефонът на човек с име Ivan

CODE ACADEMY

Примерен свързан списък

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

void create();
void display();
void insert_begin();
void insert_end();
void insert_pos();
void delete_begin();
void delete_end();
void delete_pos();


struct node
{
        int info;
        struct node *next;
};
struct node *start=NULL;
```

```c
int main(){
        int choice;
        while(1){
                printf("\n                MENU                          \n");
                printf("\n 1.Create     \n");
                printf("\n 2.Display    \n");
                printf("\n 3.Insert at the beginning    \n");
                printf("\n 4.Insert at the end  \n");
                printf("\n 5.Insert at specified position       \n");
                printf("\n 6.Delete from beginning      \n");
                printf("\n 7.Delete from the end        \n");
                printf("\n 8.Delete from specified position     \n");
                printf("\n 9.Exit       \n");
                printf("\n--------------------------------------\n");
                printf("Enter your choice:t");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1:       create();      break;
                        case 2:       display();     break;
                        case 3:      insert_begin(); break;
                        case 4:      insert_end();   break;
                        case 5:      insert_pos();   break;
                        case 6:      delete_begin(); break;
                        case 7:      delete_end();   break;
                        case 8:       delete_pos();  break;
                        case 9:       exit(0);  break;
                        default:    printf("\n Wrong Choice:\n");   break;
                }
        }
        return 0;
}
```

CODE ACADEMY

Примерен свързан списък

```
void create()
{
        struct node *temp,*ptr;
        temp=(struct node *)malloc(sizeof(struct n
        if(temp==NULL)
        {
                printf("\nOut of Memory Space:\n")
                exit(0);
        }
        printf("\nEnter the data value for the nod
        scanf("%d",&temp->info);
        temp->next=NULL;
        if(start==NULL)
        {
                start=temp;
        }
        else
        {
                ptr=start;
                while(ptr->next!=NULL)
                {
                        ptr=ptr->next;
                }
                ptr->next=temp;
        }
}
```

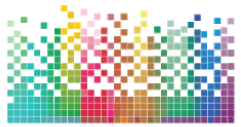
```
void display()
{
        struct node *ptr;
        if(start==NULL)
        {
                printf("\nList is empty:\n");
                return;
        }
        else
        {
                ptr=start;
                printf("\nThe List elements are:\n");
                while(ptr!=NULL)
                {
                        printf("%d\t",ptr->info );
                        ptr=ptr->next ;
                }
        }
}
void insert_begin()
```

CODE ACADEMY

Примерен свързан списък

```
void insert_begin()
{
        struct node *temp;
        temp=(struct node *)malloc(sizeof(struct node));
        if(temp==NULL)
        {
                printf("\nOut of Memory Space:\n");
                return;
        }
        printf("\nEnter the data value for the node:\t" );
        scanf("%d",&temp->info);
        temp->next =NULL;
        if(start==NULL)
        {
                start=temp;
        }
        else
        {
                temp->next=start;
                start=temp;
        }
}
```

```
void insert_end()
{
        struct node *temp,*ptr;
        temp=(struct node *)malloc(sizeof(struct node));
        if(temp==NULL)
        {
                printf("\nOut of Memory Space:\n");
                return;
        }
        printf("\nEnter the data value for the node:\t" );
        scanf("%d",&temp->info );
        temp->next =NULL;
        if(start==NULL)
        {
                start=temp;
        }
        else
        {
                ptr=start;
                while(ptr->next !=NULL)
                {
                        ptr=ptr->next ;
                }
                ptr->next =temp;
        }
}
```



Примерен свързан списък

```
void insert_pos()
{
        struct node *ptr,*temp;
        int i,pos;
        temp=(struct node *)malloc(sizeof(struct node));
        if(temp==NULL)
        {
                printf("nOut of Memory Space:n");
                return;
        }
        printf("\nEnter the position for the new node to be inserted:\t");
        scanf("%d",&pos);
        printf("\nEnter the data value of the node:\t");
        scanf("%d",&temp->info) ;

        temp->next=NULL;
        if(pos==0)
        {
                temp->next=start;
                start=temp;
        }
        else
        {
                for(i=0,ptr=start;i<pos-1;i++) { ptr=ptr->next;
                        if(ptr==NULL)
                        {
                                printf("\nPosition not found:[Handle with care]\n");
                                return;
                        }
                }
                temp->next =ptr->next ;
                ptr->next=temp;
        }
}
```

CODE ACADEMY

Примерен свързан списък

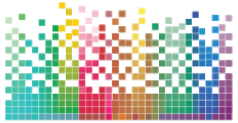
```
void delete_begin()
{
        struct node *ptr;
        if(ptr==NULL)
        {
                printf("\nList is Empty:\n");
                return;
        }
        else
        {
                ptr=start;
                start=start->next ;
                printf("\nThe deleted element is :%d\t",ptr->info);
                free(ptr);
        }
}
```

```
void delete_end()
{
        struct node *temp,*ptr;
        if(start==NULL)
        {
                printf("\nList is Empty:");
                exit(0);
        }
        else if(start->next ==NULL)
        {
                ptr=start;
                start=NULL;
                printf("\nThe deleted element is:%d\t",ptr->info);
                free(ptr);
        }
        else
        {
                ptr=start;
                while(ptr->next!=NULL)
                {
                        temp=ptr;
                        ptr=ptr->next;
                }
                temp->next=NULL;
                printf("\nThe deleted element is:%d\t",ptr->info);
                free(ptr);
        }
}
```

CODE ACADEMY

Примерен свързан списък

```
void delete_pos()
{
        int i,pos;
        struct node *temp,*ptr;
        if(start==NULL)
        {
                printf("\nThe List is Empty:\n");
                exit(0);
        }
        else
        {
                printf("\nEnter the position of the node to be deleted:\t");
                scanf("%d",&pos);
                if(pos==0)
                {
                        ptr=start;
                        start=start->next ;
                        printf("\nThe deleted element is:%d\t",ptr->info  );
                        free(ptr);
                }
                else
                {
                        ptr=start;
                        for(i=0;i<pos;i++) { temp=ptr; ptr=ptr->next ;
                                if(ptr==NULL)
                                {
                                        printf("\nPosition not Found:\n");
                                        return;
                                }
                        }
                        temp->next =ptr->next ;
                        printf("\nThe deleted element is:%d\t",ptr->info );
                        free(ptr);
                }
        }
}
```



CODE ACADEMY

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

/* doubly linked list */
typedef struct TDListNode {
  int m_nValue;
  struct TDListNode* m_pNextNode;
  struct TDListNode* m_pPrevNode;
} TDListNode;

void print(TDListNode * s);

int main()
{
 TDListNode *start;
 TDListNode *current;
for(int i=0;i<3;i++){
 current=(TDListNode *)malloc(sizeof(TDListNode));
 if(current==NULL)
 {
  printf("No memmory");
  return  1;
 }
    if(start==NULL) // empty list
    {
      current->m_nValue=i+5;
      current->m_pNextNode=NULL;
      current->m_pPrevNode=NULL;
      start=current;
    }
    else{
        TDListNode * tmp=start;
        while(tmp->m_pNextNode!=NULL){
            tmp=tmp->m_pNextNode;
        }
       current->m_nValue=i+5;
       current->m_pNextNode=NULL;
       current->m_pPrevNode=tmp;
       tmp->m_pNextNode=current;
    }
    }
     print(start);
  }

   void print(TDListNode * s){

     for(TDListNode * tmp =s; tmp!=NULL; tmp=tmp->m_pNextNode)
     {
         printf("%d\n", tmp->m_nValue);
     }
    }
```

CODE ACADEMY

Домашно:

**Всички задачи които следват +
да си напишете кодовете от
презентацията и да си поиграете
с тях.**

Задачи

Задача 1. Реализация на списък. Напишете програма, която добавя и изтрива елемент от списък по позиция в списъка. Използвайте следния елемент на динамичния списък:

```
typedef struct t_node {
  int m_nValue;
  t_node* m_pNext;
} t_node;
```

CODE ACADEMY

Задачи

Задача 2. Направете едносвързан списък, съдържащ числата 1-14 и принтирайте петия елемент от края му.

Задача 3. Моделиране на играта “Броеница”: N деца застават в кръг и получават номера от 1 до N. Като се започне от дете 1, по часовниковата стрелка се отброяват M деца. Дете с номер M излиза от кръга, след което започва ново броене от следващото дете. Продължава, докато остане само едно дете, чийто номер трябва да се определи.

CODE ACADEMY

Задачи

Задача 4. Двойно-свързан списък. Напишете програма, която добавя и изтрива елемент от списъка по зададена стойност, която се пази в него. Използвайте следния елемент на двойно-свързания списък:

```
/* doubly linked list */
typedef struct TDListNode {
  int m_nValue;
  struct TDListNode* m_pNextNode;
  struct TDListNode* m_pPrevNode;
} TDListNode;
```

CODE ACADEMY

Задачи

Задача 5. Напишете програма за въвеждане на елемент по средата на двойно свързан списък.

Задача 6. Напишете програма за триене на N-тия елемент от края на двойно свързан списък.

Задача 7. Напишете програма за триене на елемент от двойно свързан списък по зададена позиция в него.