Динамични структури данни

Динамична структура данни е структура с елемент, който сочи към структура от същия тип.

- **използва се указател;**
- **отделя се памет по време на изпълнение на програмата – функции** malloc(), calloc() **от библиотеката** stdlib.h**;**
- **освобождава се паметта след приключване на работата – функция** free() **от библиотеката** stdlib.h**.**

Възел е структура, която включва указател към друга структура от същия тип.

Връзка е указателят, който свързва една структура със следващата структура.

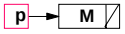
Дефиниране на възел и връзка

```
typedef <тип> DATA;          // тип на данните
struct node                  // възел
{
        DATA data;           // данни
        struct node *next;   // връзка
};
typedef struct node NODE;
typedef NODE *LINK;
```

Пример:

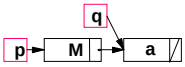
```
typedef char DATA;
LINK p, q;

// Отделяне на памет за възел p
p = (LINK)malloc(sizeof(NODE));
p->data = 'M';
p->next = NULL;
```

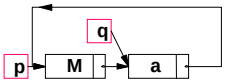



```
// Отделяне на памет за възел q
q = (LINK)malloc(sizeof(NODE));
q->data = 'a';
q->next = NULL;
```


```
// Свързан списък
p->next = q;
```




```
// Цикличен свързан списък
q->next = p;
```



Свързан списък

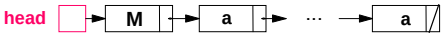
Свързан списък е множество от възли, в което всеки възел е свързан с друг възел след него и указател към първия възел в списъка.

Приложение

- **списък от приоритети на заданията, очакващи обработка от операционната система;**
- **синтактически анализ при транслация.**

Дефиниране

```
struct list                  // свързан списък
{
        LINK head;           // начало на свързан списък
};
typedef struct list LIST;
```



Характеристики

Подредба на възлите – според логическите връзки между тях, а не в зависимост от физическото им разположение в паметта (както при масив).

Възел в списъка – има две компоненти (данни data **и указател** next **към следващия елемент в списъка).**

Първи елемент в списъка – към него сочи указателят head**.**

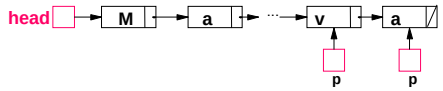
Последен елемент в списъка – указателят next **има стойност** NULL**.**

Операции

1. **Създаване на свързан списък**
2. **Отпечатване на свързан списък**
3. **Включване на възел след даден указател**
4. **Изтриване на възел след даден указател**
5. **Освобождаване на заетата памет от свързан списък**

```
#define FAILURE 0                // неуспешна операция
#define SUCCESS 1                // успешна операция
```

1. Създаване на свързан списък



Алгоритъм

начало ← ПРАЗНО
насочи текущата връзка към начало
докато искаме да въвеждаме данни повтаряй
  въведи данни
  ако списъкът е бил празен
    отдели памет за нов възел като първи елемент
    ако отделянето на памет не е успешно
      неуспешна операция
    в противен случай
      запази данните във възела
      маркирай край на списъка
      насочи началото на списъка към първия елемент
  в противен случай
    отдели памет за нов възел в края на списък
    ако отделянето на памет не е успешно
      неуспешна операция
    в противен случай
      придвижи текущата връзка към възела
      запази данните във възела
      маркирай край на списъка
успешна операция

```
int create (LIST *l)
{  LINK p;                       // текуща връзка
   DATA x;                       // данни
   l->head = NULL;
   p = l->head;
   printf("Въведи низ -> ");
   for(x = getchar(); x != '\n' && x != EOF; x = getchar())
   {  if (p == NULL)             // първи елемент в списъка
      {  p = (LINK)malloc(sizeof(NODE));
         if (p == NULL)
         {  printf("Няма достатъчно памет!\n");
            return FAILURE;
         }
         else
         {  p->data = x;
            p->next = NULL;
            l->head = p;
         }
      }
```

```
      else                       // добавяне в края на списъка
      {
         p->next = (LINK)malloc(sizeof(NODE));
         if(p->next == NULL)
         {
            printf("Няма достатъчно памет!\n");
            return FAILURE;
         }
         else
         {
            p = p->next;
            p->data = x;
            p->next = NULL;
         }
      }
   }
   return SUCCESS;
}
```

2. Отпечатване на свързан списък

Алгоритъм

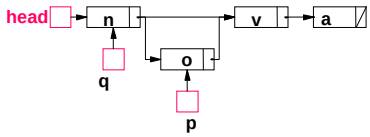
насочи текущата връзка към начало
ако текущата връзка е ПРАЗНО
  празен списък
в противен случай
  докато текущата връзка не е ПРАЗНО повтаряй
    отпечатай данните на текущия възел
    придвижи текущата връзка към следващия възел

```
void print (LIST l)
{
  LINK p;                              // текуща връзка
  p = l.head;
  if (p == NULL)
    printf("Празен списък!");
  else
    while (p! = NULL)
    {
      putchar(p->data);
      p = p->next;
    }
    putchar('\n');
}
```

3. Включване на възел след дадена връзка

Алгоритъм

отдели памет за временна връзка
ако отделянето на памет не е успешно
  неуспешна операция
в противен случай
  запази данните във временната връзка
  насочи следващата компонента на временната връзка към следващата компонента на дадения възел
  насочи следващата компонента на дадения възел към временната връзка
  успешна операция

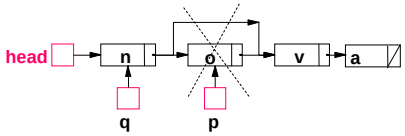



```
int insert (LINK q, DATA key)
{
  LINK p;                              // временна връзка
  p=(LINK)malloc(sizeof(NODE));
  if(p==NULL)
  {
    printf("Няма достатъчно памет!\n");
    return FAILURE;
  }
  else
  {
    p->data=key;
    p->next=q->next;
    q->next=p;
  }
  return SUCCESS;
}
```

4. Изтриване на възел след дадена връзка

Алгоритъм

насочи временната връзка към следващата компонента на дадения възел
насочи следващата компонента на дадения възел към следващата компонента на временния възел
освободи паметта, заета от временната връзка

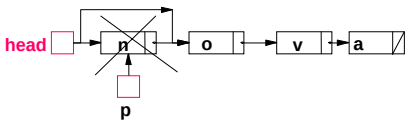



```
int del (LINK q)
{
  LINK p;                              // временна връзка
  if (q == NULL)
    return FAILURE;
  else
  {
    p = q->next;
    q->next = p->next;
    free(p);
    return SUCCESS;
  }
}
```

5. Освобождаване на заетата памет от свързан списък

Алгоритъм

насочи временната връзка към начало на списъка
докато начало не е ПРАЗНО
  придвижи начало на списъка
  освободи паметта, заета от предишния възел
  насочи временната връзка към начало на списъка



```
void freelist(LIST *l)
{
  LINK p;                              // временна връзка
  for (p = l->head; l->head != NULL; p = l->head)
  {
    l->head = l->head->next;
    free (p);
  }
  printf("Освободена е заетата памет от списъка!\n");
}
```

Задача: **Свързан списък от символи**

1. **Напишете функция** count()**, която изчислява броя на елементите в свързан списък.**

Алгоритъм

брояч ← ПРАЗНО
насочи временната връзка към начало на списъка
докато временната връзка не е ПРАЗНО
  увеличи брояча
  придвижи временната връзка

2. **Използвайте функцията** create() **и създайте свързан списък от символи, в който въведете вашето собствено и фамилно име.**
3. **Използвайте функцията** count() **и изчислете броя на символите във вашето име.**

4. **Използвайте многократно функцията** insert() **и добавете: инициал на бащиното ви име, точка и празна позиция.**
5. **Използвайте функцията** del() **и изтрийте точката след инициала на бащиното ви име.**
6. **Използвайте функцията** print() **и разпечатайте получения свързан списък след стъпки 2, 3, 4 и 5.**