napszéltől. Az Ariane-5-ben az Ariane-4 vezérlőszoftverét használták fel (mivel az már bizonyított), de mivel az új hajtómű gyorsabb volt elődjénél, a számításoknál aritmetikai hiba lépett fel, ami miatt a hajtóművek túlterhelődtek és működésképtelenné váltak.

- 1999-ben a Mars Climate Orbite űrszonda esett áldozatul szoftverhibának, így nem maradt esélye a Mars légkörének vizsgálatára. A légkörbe ugyan belépett, de sajnos azonnal el is égett. A problémát az okozta, hogy a szoftvermodulokat külön fejlesztőcsoportok készítették, és nem állapodtak meg abban, hogy milyen mértékegységeket (metrikus vagy angolszász) használnak a számításoknál.
- 1991. február 25-én Irakban egy eltévedt Scud rakéta ölt meg 28 amerikai katonát. Ez azért történhetett meg, mert a Patriot légelhárító rendszer és a radar rendszerórájának értéke kis mértékben eltért (0.3 mp-el), amely a mozgó SCUD rakéta esetén 600 méteres különbséget jelent a célzásnál. A rakétarendszer készítői tudtak a hibáról, de a javított szoftverváltozatot a tragédia után csak 1 nappal tudták biztosítani.
- 1988-ban az első internetes féreg (Morris) kevesebb, mint egy nap leforgása alatt 2000–6000 SUN és VAX számítógépet fertőzött meg, kihasználva az operációs rendszer egyik hibáját. A féreg kifejlesztője (elmondása szerint) nem akart kárt okozni, csak meg szerette volna becsülni az internetre kapcsolt számítógépek számát. A féreg működéséből adódóan azonban a szerverek nagymértékben lelassultak, illetve elérhetetlenné váltak, nem kis riadalmat keltve akkoriban. A féreg kifejlesztője felfüggesztett börtönbüntetést és pénzbüntetést kapott. Azóta megszerezte a doktori címét, kutatási témája a számítógépes hálózatok felépítésével kapcsolatos.

Egyszerű programok kezdőknek

V. rész

Láncolt listák *C++*-ban

A mutató olyan típus, amely változói értékként egy címet tartalmaz. A *C++* nyelvben nagyon jól ki van dolgozva a *pointeraritmetika*.

A mutató (pointer) típust *C++*-ban a `*` unáris operátor jelzi. Például, ha egy egész számra mutató pointert akarunk deklarálni, akkor ezt az `int *p;` változódeklarációval tehetjük meg. A `p` pointer típusú változó és az a cím, amely a `p` értéke, egy egész számot tartalmaz. Ha egy pointernek egy változó címét akarjuk megfeleltetni, akkor a cím (unáris `&`) operátort kell használnunk: `p = &i;`.

A `void *` deklaráció explicit típuskonverziót követel, és ezt bármilyen mutatótípusra használhatjuk.

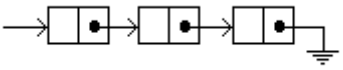
A *C++* a **`NULL`** fenntartott szóval jelöli azokat a mutatókat, amelyek semmilyen zónára sem referálnak. Ha tehát azt mondjuk egy mutatóra, hogy `p = NULL;` akkor ez a mutató üres. A **`NULL`** szimbólumot grafikusan így ábrázoljuk:

Ha mutatót definiálunk egy memóriacímen lévő értékre, és ha ez az érték egy másik memóriacímet is tartalmaz, akkor memórialáncolatokat, listákat hozhatunk létre. A lista dinamikus változók sorozata. A listák elemei két fontos, elkülöníthető részt tartalmaz-

nak, az *adat-* és a *címzónát*. Az adatzóna az elemek adatait, míg a címzóna a következő elem címét tartalmazza.

A *láncolt* (*egyszeresen* vagy *szimplán láncolt*) *lista* egy dinamikus adatsorozat, amelyet egy **NULL** mutató zár le. A listaelemet grafikusan így ábrázoljuk:

A ponttal jelölt rész a címzóna, a másik az adatzóna. A lista tehát a következőképpen ábrázolható:



típusként pedig a következőképpen hozható létre:

```
struct lista
{
  int adat;
  struct lista* kovetkezo;
};
```

A fenti deklarációban a `lista` egy struktúra, amely az adatzónát (`adat`) és a címzónát (`kovetkezo`) tartalmazza. A címzóna egy **`struct lista`** típusú mutató.

Listákra a következő műveletek értelmezettek: *hozzáillesztés, beszúrás, törlés, a lista bejárása, egy elem keresése.*

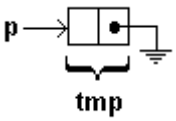
Hozzáillesztés

Legyen `p` egy **`struct lista`** típusú lista, a `p` értéke először **`NULL`**:
Ehhez a `p` hez hozzáilleszthetünk egy lista elemet a következőképpen:

Legyen `tmp` a hozzáillesztendő **`struct lista`** típusú elem. Először helyet foglalunk le az új elem számára: `tmp = new struct lista;`, majd a következő műveleteket végezzük:

```
tmp->adat = adat;
tmp->kovetkezo = p;
p = tmp;
```

Így a `p` most az új elemre, vagyis a lista végére fog mutatni, a `tmp->kovetkezo`-je pedig a régi `p`-re vagyis **`NULL`**-ra. A `tmp` az új elem:



Az eljárást folytatva új elemeket illeszthetünk a listához, a `p` mindig a lista végére fog mutatni.

A listát egymásbaágyazott függvényhívás segítségével is létre tudjuk hozni. Ekkor olyan függvényt kell írjunk, amelynek visszatérő értéke egy PList típusú mutató, így ez a következő paraméter bemeneteli értéke lehet:

```
struct lista *Lista(int adat, struct lista
*kovetkezo)
{
  struct lista *tmp;
  tmp = new struct lista;
  tmp->adat = adat;
  tmp->kovetkezo = kovetkezo;
  return tmp;
```

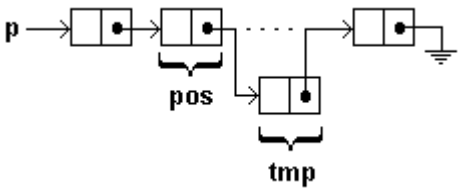
```
}
```

A listát pedig így hozhatjuk létre: `p = Lista(1, Lista(2, Lista(3, Lista(4, Lista(5, NULL)))));`

Beszúrás

Beszúrásról akkor beszélünk, amikor egy, már meglévő lista két eleme közé be akarunk illeszteni egy új elemet. Legyen `tmp` az új elem, `pos` pedig a beillesztési pozícióra mutató `struct lista` típusú pointer. Itt is először helyet kell lefoglaljunk az új elem számára, majd a következő műveleteket végezzük:

```
tmp->adat = adat;
tmp->kovetkezo = pos->kovetkezo;
pos->kovetkezo = tmp;
```

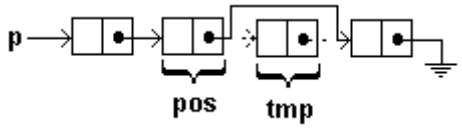


Törlés

Ha egy, már meglévő listából ki akarunk törölni egy elemet, akkor a következőképpen járunk el: Legyen `tmp` a kitörlendő elem, `pos` pedig a törlés pozíciója. Elvégezzük a törlési műveletet:

```
tmp = pos->kovetkezo;
pos->kovetkezo = pos->kovetkezo->kovetkezo;
```

majd felszabadítjuk a helyet: `delete tmp;`



A lista bejárása

Ha már létrehoztunk egy listát és van egy `p` pointerünk, amely a lista végére mutat, akkor végigjárhatjuk a listát a következőképpen:

```
while(p!=NULL)
{
  cout<<p->adat<<endl;
  p = p->kovetkezo;
}
```

A **`while`** ciklus egy üres listát is végig tud járni, a ciklusba tetszőleges műveletek írhatók, és a ciklus végrehajtása után a `p` mutató a lista elejére fog mutatni. Ha meg akarjuk őrizni a végét, akkor ezt egy más változóba kell hogy elmentsük, vagy eljárásnál érték szerinti paraméterátadást használjunk.

Egy elem keresése a listában

Ha van egy létrehozott listánk és egy `p`, a lista végére mutató pointerünk, akkor egy elemet a következő eljárással kereshetünk meg:

```
while((p!=NULL)&&(p->adat!=adat))
  if(p->adat!=adat) p = p->kovetkezo;
```

A keresett adatra, ha benne van a listában, a `p` mutató fog mutatni, különben ez **`NULL`** lesz.

A teljes program:

```
#include<iostream>
#include<stdlib.h>

using namespace std;

struct lista
{
  int adat;
  struct lista *kovetkezo;
};

struct lista *Lista(int adat, struct lista
*kovetkezo)
{
  struct lista *tmp;
  tmp = new struct lista;
  tmp->adat = adat;
  tmp->kovetkezo = kovetkezo;
  return tmp;
}

void Beszur(struct lista *pos, int adat)
{
  struct lista *tmp;
  tmp = new struct lista;
  tmp->adat = adat;
  tmp->kovetkezo = pos->kovetkezo;
  pos->kovetkezo = tmp;
}

void Torol(struct lista *pos)
{
  struct lista *tmp;
  if(pos!=NULL)
  {
    tmp = pos->kovetkezo;
    pos->kovetkezo = pos->kovetkezo->kovetkezo;
    delete tmp;
  }
}

void Bejar(struct lista *p)
{
   while(p!=NULL)
   {
      cout<<p->adat<<endl;
      p = p->kovetkezo;
   }
}

struct lista *Keres(int adat, struct lista *p)
{
   while((p!=NULL)&&(p->adat!=adat))
     if(p->adat!=adat) p = p->kovetkezo;
```

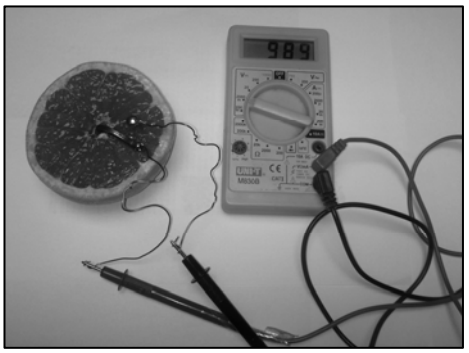
```
   return p;
}

int main()
{
    struct lista *p, *tmp;
    p = Lista(1, Lista(2, Lista(3, Lista(4, Lista(5,
NULL)))));
    tmp = Keres(3, p);
    if (tmp!=NULL) Beszur(tmp, 6);
    Bejar(p);
    Torol(tmp);
    Bejar(p);
    system("pause");
    return 0;
}
```

Kovács Lehel István

kísérlet, labor

Látványosak, érdekesek, hasznosak

Energia termelés, energia átalakítás



A kísérlethez szükséges anyagok, eszközök: gyümölcsök: citrom, narancs (más citrikus gyümölcs), alma; zöldségek: krumpli, paradicsom, murok, vöröscékla, fekete retek, hagyma; drótok: réz, alumínium, vas; pénzérmék (50, 10, 5-banis értékűek), ezüst és esetleg arany gyűrű, kés, univerzális árammérő műszer (millivolt értékekre is használható).

A gyümölcsöket keresztbe vágjátok kettőbe. Két különböző minőségű drótot, vagy lemezkét egymástól 1cm távolságra szúrjatok be a gyümölcs vágási felületébe. A drótok másik végét csatlakoztassátok a mérőműszerhez (lényeges, hogy az elektromos vezetés biztosított legyen, ezért a drótok végeit előzetesen alaposan tisztítsátok meg).

Figyeljétek a műszer jelzését (áramfeszültség, áramerősség). Amikor a mutatott érték állandósult, azt jegyezzétek fel az előre elkészített táblázatba.

Mérési eredmények

Gyümölcs	Elektródok	Feszültség (V)	Áramerősség (A)