Univerzális programozás

Írd meg a saját programozás tankönyvedet!

Ed. BHAX, DEBRECEN,
2019. február 19, v. 0.0.4

COLLABORATORS

	TITLE : Univerzális programozás	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Bátfai, Norbert	2019. április 23.	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2019-02-12	Az iniciális dokumentum szerkezetének kialakítása.	nbatfai
0.0.2	2019-02-14	Inciális feladatlisták összeállítása.	nbatfai
0.0.3	2019-02-16	Feladatlisták folytatása. Feltöltés a BHAX csatorna https://gitlab.com/nbatfai/bhax repójába.	nbatfai
0.0.4	2019-02-19	Aktualizálás, javítások.	nbatfai

Ajánlás

„To me, you understand something only if you can program it. (You, not someone else!) Otherwise you don't really understand it, you only think you understand it."

—Gregory Chaitin, *META MATH! The Quest for Omega*, [METAMATH]

Tartalomjegyzék

Előszó

Amikor programozónak terveztem állni, ellenezték a környezetemben, mondván, hogy kell szövegszerkesztő meg táblázatkezelő, de az már van... nem lesz programozói munka.

Tévedtek. Hogy egy generáció múlva kell-e még tömegesen hús-vér programozó vagy olcsóbb lesz allokálni igény szerint pár robot programozót a felhőből? A programozók dolgozók lesznek vagy papok? Ki tudhatná ma.

Mindenesetre a programozás a teoretikus kultúra csúcsa. A GNU mozgalomban látom annak garanciáját, hogy ebben a szellemi kalandban a gyerekeim is részt vehessenek majd. Ezért programozunk.

Hogyan forgasd

A könyv célja egy stabil programozási szemlélet kialakítása az olvasóban. Módszere, hogy hetekre bontva ad egy tematikus feladatcsokrot. Minden feladathoz megadja a megoldás forráskódját és forrásokat feldolgozó videókat. Az olvasó feladata, hogy ezek tanulmányozása után maga adja meg a feladat megoldásának lényegi magyarázatát, avagy írja meg a könyvet.

Miért univerzális? Mert az olvasótól (kvázi az írótól) függ, hogy kinek szól a könyv. Alapértelmezésben gyerekeknek, mert velük készítem az iniciális változatot. Ám tervezem felhasználását az egyetemi programozás oktatásban is. Ahogy szélesedni tudna a felhasználók köre, akkor lehetne kiadása különböző korosztályú gyerekeknek, családoknak, szakköröknek, programozás kurzusoknak, felnőtt és továbbképzési műhelyeknek és sorolhatnánk...

Milyen nyelven nyomjuk?

C (mutatók), C++ (másoló és mozgató szemantika) és Java (lebutított C++) nyelvekből kell egy jó alap, ezt kell kiegészíteni pár R (vektoros szemlélet), Python (gépi tanulás bevezető), Lisp és Prolog (hogy lássuk mást is) példával.

Hogyan nyomjuk?

Rántsd le a https://gitlab.com/nbatfai/bhax git repót, vagy méginkább forkolj belőle magadnak egy sajátot a GitLabon, ha már saját könyvön dolgozol!

Ha megvannak a könyv DocBook XML forrásai, akkor az alább látható **make** parancs ellenőrzi, hogy „jól formázottak” és „érvényesek-e” ezek az XML források, majd elkészíti a dblatex programmal a könyved pdf változatát, íme:

```
batfai@entropy:~$ cd glrepos/bhax/thematic_tutorials/bhax_textbook/
batfai@entropy:~/glrepos/bhax/thematic_tutorials/bhax_textbook$ make
rm -f bhax-textbook-fdl.pdf
xmllint --xinclude bhax-textbook-fdl.xml --output output.xml
xmllint --relaxng http://docbook.org/xml/5.0/rng/docbookxi.rng output.xml  ↩
   --noout
output.xml validates
rm -f output.xml
dblatex bhax-textbook-fdl.xml -p bhax-textbook.xls
Build the book set list...
Build the listings...
XSLT stylesheets DocBook - LaTeX 2e (0.3.10)
====================================================
Stripping NS from DocBook 5/NG document.
Processing stripped document.
Image 'dblatex' not found
Build bhax-textbook-fdl.pdf
'bhax-textbook-fdl.pdf' successfully built
```

Ha minden igaz, akkor most éppen ezt a legenerált `bhax-textbook-fdl.pdf` fájlt olvasod.

A DocBook XML 5.1 új neked?
Ez esetben forgasd a https://tdg.docbook.org/tdg/5.1/ könyvet, a végén találod az informatikai szövegek jelölésére használható gazdag „API” elemenkénti bemutatását.

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

1.2. Milyen doksikat olvassak el?

- Olvasgasd a kézikönyv lapjait, kezd a **man man** parancs kiadásával. A C programozásban a 3-as szintű lapokat fogod nézegetni, például az első feladat kapcsán ezt a **man 3 sleep** lapot
- [KERNIGHANRITCHIE]
- [BMECPP]
- Az igazi kockák persze csemegéznek a C nyelvi szabvány ISO/IEC 9899:2017 kódcsipeteiből is.

1.3. Milyen filmeket nézzek meg?

- 21 - Las Vegas ostroma, https://www.imdb.com/title/tt0478087/, benne a Monty Hall probléma bemutatása.

II. rész

Tematikus feladatok

Bátf41 Haxor Stream
A feladatokkal kapcsolatos élő adásokat sugároz a https://www.twitch.tv/nbatfai csatorna, melynek permanens archívuma a https://www.youtube.com/c/nbatfai csatornán található.

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

Írj olyan C végtelen ciklusokat, amelyek 0 illetve 100 százalékban dolgoztatnak egy magot és egy olyat, amely 100 százalékban minden magot!

Megoldás forrása ha a magot 100%-on akajruk dolgoztatni:https://sourceforge.net/p/udprog/code/ci/master/-tree/source/kezdo/elsoc/vegtelen/v.c

```
int main ()
{
  for (;;)

  return 0;
}
```

Megoldás forrása ha a magot 0%-on akarjuk dolgoztatni: https://sourceforge.net/p/udprog/code/ci/master/-tree/source/kezdo/elsoc/vegtelen/vs.c

```
#include <unistd.h>
int main ()
{
  for (;;)
    sleep (1);
  return 0;
}
```

Megoldás ha minden magot 100%-on akarunk dolgoztatni:

```
#include <omp.h>
int main()
{
    #pragma omp parallel
    for(;;){}
}
```

A sleep függvény alvó/várakozó állapotba küldi a magot ezért a terhelés 0%-os lesz. A sleep elhagyása nélkül a program egy magot 100%-on dolgoztat. Ha az első példát parallelel minden magon futtatjuk akkor minden mag 100%-osan le lesz terhelve.

2.2. Lefagyott, nem fagyott, akkor most mi van?

Mutasd meg, hogy nem lehet olyan programot írni, amely bármely más programról eldönti, hogy le fog-e fagyni vagy sem!

Megoldás forrása: tegyük fel, hogy akkora haxorok vagyunk, hogy meg tudjuk írni a `Lefagy` függvényt, amely tetszőleges programról el tudja dönteni, hogy van-e benne vlgtelen ciklus:

```
Program T100
{

  boolean Lefagy(Program P)
  {
     if(P-ben van végtelen ciklus)
      return true;
     else
      return false;
  }

  main(Input Q)
  {
    Lefagy(Q)
  }
}
```

A program futtatása, például akár az előző `v.c` ilyen pszeudókódjára:

```
T100(t.c.pseudo)
true
```

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

A T100-as programot felhasználva készítsük most el az alábbi T1000-set, amelyben a Lefagy-ra épőlő Lefagy2 már nem tartalmaz feltételezett, csak csak konkrét kódot:

```
Program T1000
{

  boolean Lefagy(Program P)
  {
     if(P-ben van végtelen ciklus)
      return true;
```

```
      else
       return false;
  }

  boolean Lefagy2(Program P)
  {
     if(Lefagy(P))
      return true;
     else
      for(;;);
  }

  main(Input Q)
  {
    Lefagy2(Q)
  }

}
```

Mit for kiírni erre a `T1000(T1000)` futtatásra?

- Ha T1000 lefagyó, akkor nem fog lefagyni, kiírja, hogy true
- Ha T1000 nem fagyó, akkor pedig le fog fagyni...

akkor most hogy fog működni? Sehogy, mert ilyen `Lefagy` függvényt, azaz a T100 program nem is létezik.

Tanulságok, tapasztalatok, magyarázat...

2.3. Változók értékének felcserélése

Írj olyan C programot, amely felcseréli két változó értékét, bármiféle logikai utasítás vagy kifejezés használata nélkül!

Megoldás videó: https://bhaxor.blog.hu/2018/08/28/10_begin_goto_20_avagy_elindulunk

Megoldás forrása segédváltozó nélkül: https://sourceforge.net/p/udprog/code/ci/master/tree/source/kezdo/elsoc/valtozocsere/csere.c

```
    #include<stdio.h>

    int main()
    {
      int valtozo_1 = 2,valtozo_2 = 4;

      printf("valtozo_1=%d valtozo_2=%d\n",valtozo_1,valtozo_2);
```

```
      valtozo_1 = ( valtozo_1 - valtozo_2 );
      valtozo_2 = ( valtozo_1 + valtozo_2 );
      valtozo_1 = ( valtozo_2 - valtozo_1 );

      printf("valtozo_1=%d valtozo_2=%d\n",valtozo_1, valtozo_2);

      return 0;
    }
```

Megoldás forrása segédváltozóval:

```
    int main()
    {
      int v1=1, v2=2, v3;
      v3=v1;
      v1=v2;
      v2=v3;
      return 0;
    }
```

A csere segédváltozóval bevezet egy új ideiglenes változót, amiben az eslő változó értékét tároljuk amíg abba belereakjuk a 2. változó értékét mert ekkor a v1 értéke elvész. Ezután a második változóba belerakjuk a segédváltozóban eltárolt eslő változó értékét.

Segédváltozó nélküli csere csak egyszerű kivonás meg összeadás.

2.4. Labdapattogás

Először if-ekkel, majd bármiféle logikai utasítás vagy kifejezés nasználata nélkül írj egy olyan programot, ami egy labdát pattogtat a karakteres konzolon! (Hogy mit értek pattogtatás alatt, alább láthatod a videókon.)

Megoldás videó: https://bhaxor.blog.hu/2018/08/28/labdapattogas

Megoldás forrása: https://sourceforge.net/p/udprog/code/ci/master/tree/source/kezdo/elsoc/pattog/pattog.c

```
    /* Készítette: 'Kojak' felhasználó
    Dátum: 2014-02-24

   A megoldás hibái javítva Józan Csaba splint ellenőrzése alapján.(Sipos ↩
      Ádám)
    Dátum: 2015-02-08
*/


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

```
static void gotoxy(int x, int y)                    //kurzor pozicionálása
{
  int i;
  for(i=0; i<y; i++) printf("\n");              //lefelé tolás
  for(i=0; i<x; i++) printf(" ");               //jobbra tolás
  printf("o\n");    //labda ikonja

}

void usleep(int);
int main(void)
{

  int egyx=1;
  int egyy=-1;
  int i;
  int x=10;   //a labda kezdeti pozíciója
  int y=20;
  int ty[23];//magasság     // a pálya mérete
  int tx[80];//szélesség

  //pálya széleinek meghatározás

  for(i=0; i<23; i++)
       ty[i]=1;

  ty[1]=-1;
  ty[22]=-1;

  for(i=0; i<79; i++)
       tx[i]=1;

  tx[1]=-1;
  tx[79]=-1;



  for(;;)
  {

    //címsor és pozíció kijelzése
    for(i=0; i<36; i++)
       printf("_");

    printf("x=%2d", x);
    printf("y=%2d", y);

    for(i=0; i<=35; i++)
         printf("_");
```

```
    (void)gotoxy(x,y);
    //printf("o\n"); Áthelyezve a gotoxy függvényve

    x+=egyx;
    y+=egyy;

    egyx*=tx[x];
    egyy*=ty[y];

    usleep (200000);
    (void)system("clear");
  }

}
```

Tanulságok, tapasztalatok, magyarázat...

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

Írj egy programot, ami megnézi, hogy hány bites a szó a gépeden, azaz mekkora az int mérete. Használd ugyanazt a while ciklus fejet, amit Linus Torvalds a BogoMIPS rutinjában!

Megoldás forrása: https://sourceforge.net/p/udprog/code/ci/master/tree/source/kezdo/elsoc/szohossz/szohossz.c#l

```
    #include <stdio.h>
    int
    main (void)
    {
     int h = 0;
     int n = 0x01;
     do
       ++h;
     while (n <<= 1);
     printf ("A szohossz ezen a gepen: %d bites\n", h);
     return 0;
    }
```

A változó értékét egyre allítjuk, létrehozunk egy számlálót, és a változót 1-el balra shifteljük , miközben a számlálót is növeljük ciklusonként. A ciklus addig megy amíg 2 számrendszerbeli számban van egyes. (1 bájt = 00000000, az 1 egy bájton ábrázolva 00000001 ezt az egyest sifteljük balra eggyel 00000010. Ha kiesik az egyes akkor a while-ban a logikai érték hamis lesz mert az n=0. Ezért a számláló megmutatja ogy hány bites egy int.)

2.6. Helló, Google!

Írj olyan C programot, amely egy 4 honlapból álló hálózatra kiszámolja a négy lap Page-Rank értékét!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

2.7. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

Megoldás videó: https://youtu.be/xbYhp9G6VqQ

Megoldás forrása: https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/Primek_R

2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

Megoldás videó: https://bhaxor.blog.hu/2019/01/03/erdos_pal_mit_keresett_a_nagykonyvben_a_monty_hall-paradoxon_kapcsan

Megoldás forrása: https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/MontyHall_R

Tanulságok, tapasztalatok, magyarázat...

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás forrása:

```
#include <stdio.h>
int main()
{
  int x;
  printf("Tizes szamrendszerbeli szam:");
  scanf("%d",&x);
  printf("\nUnaris alakja:");
  for(int i = 0; i < x; i++)
    {
      printf("/");
    }
  printf("\n");
  return 0;
}
```

Decimálisból unáris (egyes számrendszer) számrendszerbe való átváltás során csak a 10-es számrenderben megadott számú tetszőleges karaktert kell leírni. A fenti példában ez egy for ciklussal van megoldva de bármilyen más megoldás is jó ha az megfelelő számú tetszőleges karakter ad vissza eredményül.

3.2. Az $a^nb^nc^n$ nyelv nem környezetfüggetlen

Mutass be legalább két környezetfüggő generatív grammatikát, amely ezt a nyelvet generálja!

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

3.3. Hivatkozási nyelv

A [KERNIGHANRITCHIE] könyv C referencia-kézikönyv/Utasítások melléklete alapján definiáld BNF-ben a C utasítás fogalmát! Majd mutass be olyan kódcsipeteket, amelyek adott szabvánnyal nem fordulnak (például C89), mással (például C99) igen.

```
<nem terminális> ::= konkatenációja terminálisoknak, nem terminálisoknak,
illetve {iteráció}, [opcionális], alter|natíva
<egész szám> ::= <előjel><szám>
<előjel> ::= [-|+]
<szám> ::= <számjegy>{<számjegy>}
<számjegy> ::= 0|1|2|3|4|5|6|7|8|9
```

```
<címkézett_utasítás> ::= <azonosító> | case | default
<kifejezésutasítás> ::= <kifejezés>
<deklarációs_lista> ::= <deklaráció>
<utasítás_lista> ::= <utasítás>
<összetett_utasítás> ::= <deklarációs_lista> | <utasítás_lista>
<kiválasztó_utasítás> ::= if | if else | switch
<iterációs_utasítás> ::= while | do while | for
<vezérlésátadó_ utasítás> ::= goto | continue | break | return
```

Példák a szabványok közötti különbségekre:

```
int main(){
//int a;
  return 0;
}
```

A fenti példa C89-es szabvánnyal nem fordul le mí C99-es szabvánnyal le fordul mert abban már megengedett a //-el való kommentelés, míg a C89-es szabványban csak a /* és a */ voltak kommentet jelző szimbólumok. Szóval a fenti példa C89-es szabványban így nézne ki:

```
int main(){
/*int a;*/
  return 0;
}
```

Ez mind a 2 szabványban le fog fordulni.

3.4. Saját lexikális elemző

Írj olyan programot, ami számolja a bemenetén megjelenő valós számokat! Nem elfogadható olyan megoldás, amely maga olvassa betűnként a bemenetet, a feladat lényege, hogy lexert használjunk, azaz óriások vállán álljunk és ne kispályázzunk!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

3.5. l33t.l

Lexelj össze egy l33t ciphert!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

3.6. A források olvasása

Hogyan olvasod, hogyan értelmezed természetes nyelven az alábbi kódcsipeteket? Például

```
if(signal(SIGINT, jelkezelo)==SIG_IGN)
    signal(SIGINT, SIG_IGN);
```

Ha a SIGINT jel kezelése figyelmen kívül volt hagyva, akkor ezen túl is legyen figyelmen kívül hagyva, ha nem volt figyelmen kívül hagyva, akkor a jelkezelo függvény kezelje. (Miután a **man 7 signal** lapon megismertem a SIGINT jelet, a **man 2 signal** lapon pedig a használt rendszerhívást.)

Bugok
Vigyázz, sok csipet kerülendő, mert bugokat visz a kódba! Melyek ezek és miért? Ha nem megy ránézésre, elkapja valamelyiket esetleg a splint vagy a frama?

---

i.
```
if(signal(SIGINT, SIG_IGN)!=SIG_IGN)
    signal(SIGINT, jelkezelo);
```

Ha a SIGINT nem volt figyelmen kívül hagyva akkor a jelkezelő kezelje. Ha figyelmen kívül volt hagyva továbbra is maradjon úgy.

ii.
```
for(i=0; i<5; ++i)
```

A függvény 5x fog lefutni, viszont figyelni kell arra hogy az i változót még a for ciklus ellőtt deklarálni kell, és ha az i-t ++i-vel növeltük akkor lehetőleg továbbra is úgy használjuk hogy könnyebben érthető maradjon a kód.

iii.
```
for(i=0; i<5; i++)
```

Itt is ugyanaz igaz mint az előzőnél. A cuklus 5x le fog futni ha a ciklusváltozót deklarátuk a ciklus előtt.

iv.
```
for(i=0; i<5; tomb[i] = i++)
```

Nyelvtani hiba nincs ha már létrehoztuk a tomb nevű tömböt és az i-t. Le is fog fordulni, viszont az eredmény bugos lesz.

v.
```
for(i=0; i<n && (*d++ = *s++); ++i)
```

A kód hibamentes ha már létrehoztuk a látható változókat és mutatókat.

vi.
```
printf("%d %d", f(a, ++a), f(++a, a));
```

A printf függvény ki fog írni 2 decimális számot ha már megvan az f függvény, az a változó, és ha az a változó megfelelő típusú az f függvényhez. Arra kell figyelni hogy ha az f függvény visszatérési értéke nem int akkor a kiírt értékek nem biztos hogy pontosak lesznek.

vii.
```
printf("%d %d", f(a), a);
```

A printf ki fogja írni az f függvény visszatérési értékét a-ra decimális alakban, és a értékét decimális alakban. Itt is ugyanarra kell figyelni mint az előbb. Nyelvtani hiba nincs a kódrészletben.

viii.
```
printf("%d %d", f(&a), a);
```

A kiiratás megtörténik viszont az f függvény most az a változó memória címével fog dolgozni nem az a értékével ha ezt akarjuk akkor nincs semmi gond.

Tanulságok, tapasztalatok, magyarázat...: Figyeljünk hogy hogyan haszáljuk az operátorokat, figyeljünk a kiértékelés és az értékadás sorrendjére (i++, ++i), és ismerjük a függvényeket amiket használunk.

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

```
$(\forall x \exists y ((x<y)\wedge(y \text{ prím})))$
```

Végtelen sok prím van.

```
$(\forall x \exists y ((x<y)\wedge(y \text{ prím})\wedge(SSy \text{ prím})) ↩
   )$
```

Végtelen sok ikerprím van.

```
$(\exists y \forall x (x \text{ prím}) \supset (x<y)) $
```

Véges sok prím van.

```
$(\exists y \forall x (y<x) \supset \neg (x \text{ prím}))$
```

Véges sok prím van.

Megoldás forrása: https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/MatLog_LaTeX

Megoldás videó: https://youtu.be/ZexiPy3ZxsA, https://youtu.be/AJSXOQFF_wk

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

- egész

```
int a;
```

- egészre mutató mutató

```
int *b;
```

- egész referenciája

```
int &r;
```

- egészek tömbje

```
int t[5];
```

- egészek tömbjének referenciája (nem az első elemé)

```
int (&tr)[5] = t;
```

- egészre mutató mutatók tömbje

```
int *d[5];
```

- egészre mutató mutatót visszaadó függvény

```
int *h();
```

- egészre mutató mutatót visszaadó függvényre mutató mutató

```
int *(*h) ();
```

- egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvény

```
int (*v (int c)) (int a, int b);
```

- függvénymutató egy egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvényre

```
int (*(*z) (int)) (int, int);
```

Mit vezetnek be a programba a következő nevek?

-

```
int a;
```

- Egy egész típusú változót.

```
int *b = &a;
```

- Egy egész típusú mutatót ami a-ra mutat.

```
int &r = a;
```

- a változónak a referenciája.

```
int c[5];
```

- Egy 5 elemű egész típusú tömböt.

```
int (&tr)[5] = c;
```

- Egészek tömbjének referenciáját.

```
int *d[5];
```

- 5 elemű int-re mutató mutatók tömbjét.

```
int *h ();
```

- Egy függvényt ami int-re mutató mutatót ad vissza.

```
int *(*l) ();
```

- Egy int-re mutató mutatót visszaadó függvényre mutató mutatót.(pl. az előző függvényre)

```
int (*v (int c)) (int a, int b)
```

- int-et visszaadó, két intet kapó függvényre mutató mutatót visszaadó egészet kapó függvényt.

```
int (*(*z) (int)) (int, int);
```

- int-et visszaadó, két intet kapó függvényre mutató mutatót visszaadó egészet kapó függvényre mutató mutatót.

4. fejezet

Helló, Caesar!

4.1. int *** háromszögmátrix

Megoldás forrása:

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    int nr = 3;
    double **tm;

    printf("%p\n", &tm);
    if ((tm = (double **) malloc (nr * sizeof (double *))) == NULL)
    {
  return -1;
    }
    printf("%p\n", tm);
    for (int i = 0; i < nr; ++i)
    {
  if ((tm[i] = (double *) malloc ((i + 1) * sizeof (double))) == NULL)
  {
      return -1;
  }
    }
    printf("%p\n", tm[0]);

    for (int i = 0; i < nr; ++i)
  for (int j = 0; j < i + 1; ++j)
      tm[i][j] = i * (i + 1) / 2 + j;

    for (int i = 0; i < nr; ++i)
    {
  for (int j = 0; j < i + 1; ++j)
      printf ("%f, ", tm[i][j]);
```

```
    printf ("\n");
      }
      for (int i = 0; i < nr; ++i)
      {
    for (int j = 0; j < i + 1; ++j)
        printf ("%f, ", tm[i][j]);
    printf ("\n");
      }
      for (int i = 0; i < nr; ++i)
    free (tm[i]);
      free (tm);
      return 0;
}
```

Ha a négyzetes mátrix főátlója alatt vagy felett minden elem 0, a mátrx háromszögmátrix. Az alsó háromszögmátrix elemeit sorfolytonosan bejárva el tudjuk helyezni egy tömbben, az így kapott elemek száma n(n+1)/2 lesz. A `malloc` függvény képes lefoglalni a dinamikus területen egy, a paramétereként kapott méretű területet. A `free` függvény felszabadítja a lefoglalt területet.

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás forrás:

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#define MAX_KULCS 100
#define BUFFER_MERET 256
int main (int argc, char **argv)
{
  char kulcs[MAX_KULCS];
  char buffer[BUFFER_MERET];
  int kulcs_index = 0;
  int olvasott_bajtok = 0;
  int kulcs_meret = strlen (argv[1]);
  strncpy (kulcs, argv[1], MAX_KULCS);
  while ((olvasott_bajtok = read (0, (void *) buffer, BUFFER_MERET)))
    {
      for (int i = 0; i < olvasott_bajtok; ++i)
  {
    buffer[i] = buffer[i] ^ kulcs[kulcs_index];
    kulcs_index = (kulcs_index + 1) % kulcs_meret;
  }
      write (1, buffer, olvasott_bajtok);
    }
}
```

Ez a titkosítási módszer a xor művelet azonosságain alapul: A XOR B = C és C XOR B = A (A=tiszta bemeneti adat, B=titkosító kulcs, C=kimenő titkosított adat)

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

```
           public class Titkosito {
    public Titkosito(String kulcsSzöveg,
            java.io.InputStream bejövőCsatorna,
            java.io.OutputStream kimenőCsatorna)
            throws java.io.IOException {
        byte [] kulcs = kulcsSzöveg.getBytes();
        byte [] buffer = new byte[256];
        int kulcsIndex = 0;
        int olvasottBájtok = 0;
        while((olvasottBájtok =
                bejövőCsatorna.read(buffer)) != -1) {
            for(int i=0; i<olvasottBájtok; ++i) {
                buffer[i] = (byte)(buffer[i] ^ kulcs[kulcsIndex]);
                kulcsIndex = (kulcsIndex+1) % kulcs.length;
            }
            kimenőCsatorna.write(buffer, 0, olvasottBájtok);
        }
    }
    public static void main(String[] args) {
        try {
            new Titkosito(args[0], System.in, System.out);
        } catch(java.io.IOException e) {
            e.printStackTrace();
        }
    }
}
```

Itt csak nyelvi különbségek vannak a feladatok között.

4.4. C EXOR törő

Írj egy olyan C programot, amely megtöri az első feladatban előállított titkos szövegeket!

```
#define MAX_TITKOS 4096
#define OLVASAS_BUFFER 256
#define KULCS_MERET 8
#define _GNU_SOURCE
#include <stdio.h>
```

```
#include <unistd.h>
#include <string.h>

double
atlagos_szohossz (const char *titkos, int titkos_meret)
{
  int sz = 0;
  for (int i = 0; i < titkos_meret; ++i)
    if (titkos[i] == ' ')
      ++sz;

  return (double) titkos_meret / sz;
}
int tiszta_lehet (const char *titkos, int titkos_meret)
{
  double szohossz = atlagos_szohossz (titkos, titkos_meret);
  return szohossz > 6.0 && szohossz < 9.0
    && strcasestr (titkos, "hogy") && strcasestr (titkos, "nem")
    && strcasestr (titkos, "az") && strcasestr (titkos, "ha");
}

void exor (const char kulcs[], int kulcs_meret, char titkos[], int  ↩
   titkos_meret)
{
  int kulcs_index = 0;
  for (int i = 0; i < titkos_meret; ++i)
    {
      titkos[i] = titkos[i] ^ kulcs[kulcs_index];
      kulcs_index = (kulcs_index + 1) % kulcs_meret;
    }
}
int exor_tores (const char kulcs[], int kulcs_meret, char titkos[],
      int titkos_meret)
{
  exor (kulcs, kulcs_meret, titkos, titkos_meret);
  return tiszta_lehet (titkos, titkos_meret);
}
int
main (void)
{
  char kulcs[KULCS_MERET];
  char titkos[MAX_TITKOS];
  char *p = titkos;
  int olvasott_bajtok;
  // titkos fajt berantasa
  while ((olvasott_bajtok =
    read (0, (void *) p,
    (p - titkos + OLVASAS_BUFFER <
     MAX_TITKOS) ? OLVASAS_BUFFER : titkos + MAX_TITKOS - p)))
    p += olvasott_bajtok;
```

```
    // maradek hely nullazasa a titkos bufferben
    for (int i = 0; i < MAX_TITKOS - (p - titkos); ++i)
      titkos[p - titkos + i] = '\0';
    // osszes kulcs eloallitasa
    for (int ii = '0'; ii <= '9'; ++ii)
      for (int ji = '0'; ji <= '9'; ++ji)
        for (int ki = '0'; ki <= '9'; ++ki)
    for (int li = '0'; li <= '9'; ++li)
      for (int mi = '0'; mi <= '9'; ++mi)
        for (int ni = '0'; ni <= '9'; ++ni)
          for (int oi = '0'; oi <= '9'; ++oi)
    for (int pi = '0'; pi <= '9'; ++pi)
      {
        kulcs[0] = ii;
        kulcs[1] = ji;
        kulcs[2] = ki;
        kulcs[3] = li;
        kulcs[4] = mi;
        kulcs[5] = ni;
        kulcs[6] = oi;
        kulcs[7] = pi;

        if (exor_tores (kulcs, KULCS_MERET, titkos, p - titkos))
          printf
      ("Kulcs: [%c%c%c%c%c%c%c%c]\nTiszta szoveg: [%s]\n",
       ii, ji, ki, li, mi, ni, oi, pi, titkos);

        // ujra EXOR-ozunk, igy nem kell egy masodik buffer
        exor (kulcs, KULCS_MERET, titkos, p - titkos);
      }

    return 0;
}
```

Ez az exor törő az előző feladattal működik, az ha a bemeneti szöveg magyar nyelvű és a titkosító kulcs 8 számjegyből áll. Nagyjából ez csak egy sima brute force algoritmus ami minden lehetséges kulccsal elvégzi a törést és ha az eredmény a feltételeknek (atlagos_szohossz és tiszta_lehet függvények) megfelel akkor azt adja vissza eredményül.

4.5. Neurális OR, AND és EXOR kapu

Megoldás videó: https://youtu.be/Koyw6IH5ScQ

Megoldás forrása: https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/NN_R/nn.r

Neurális hálónak nevezzük azt a párhuzamos működésre képes információfeldolgozó eszközt, amely nagyszámú, hasonló típusú elem összekapcsolt rendszeréből áll Ezenfelül egyik legfontosabb jellemzője az hogy rendelkezik tanulási algoritmussal és képes előhívni a megtanult információt.

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás forrása: https://github.com/nbatfai/nahshon/blob/master/ql.hpp#L64

Tanulságok, tapasztalatok, magyarázat...

5. fejezet

Helló, Conway!

5.1. Hangyaszimulációk

Írj Qt C++-ban egy hangyaszimulációs programot, a forrásaidról utólag reverse engineering jelleggel készíts UML osztálydiagramot is!

Megoldás videó: https://bhaxor.blog.hu/2018/10/10/myrmecologist

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

5.2. Java életjáték

Írd meg Java-ban a John Horton Conway-féle életjátékot, valósítsa meg a sikló-kilövőt!

Megoldás videó:

Megoldás forrása:

John Horton Conway angol matematikus nevéhez fűződik az életjáték nevű személytelen játék. A "játékosnak" kizárólag a kiindulási pozíciót, kezdőalakzatot kell meghatároznia, ezután csak figyelnie kell az események alakulását. Matematikai szempontból sejtautomatának nevezzük Conway játékát, melyben a négyzetrácsokat celláknak és a korongokat sejteknek nevezzük. Minden sejtet nyolc szomszédos cella vesz körül. Ha egy generációban egy sejtnek kettő vagy három élő szomszédja van, akkor a sejt élni fog a következő generációban is, minden más esetben a sejt kihal. Ha egy üres cellának pontosan három élő sejt van a szomszédjában, akkor ott új sejt születik. Ezek a szabályok a `időFejlődés()` függvényben kerülnek bevezetésre. Két rácsot használunk majd, az egyik a sejttér állapotát a t_n, a másik a t_n+1 időpillanatban jellemzi. A sejttérbe "élőlényeket" helyezünk, ez a siklóágyú. Adott irányban siklókat lő ki. Az `addKeyListener`, `addMouseListener`, `addMouseMotionListener` függvényekben meghatározzuk azokat a billentyűlenyomásokat és egéreseményeket, amelyekkel tudjuk a futó Életjáték-program eseményeinek alakulását befolyásolni.

```
public class Sejtautomata extends java.awt.Frame implements Runnable {
public static final boolean ÉLŐ = true;
public static final boolean HALOTT = false;
```

```
protected boolean [][][] rácsok = new boolean [2][][];
protected boolean [][] rács;
protected int rácsIndex = 0;
protected int cellaSzélesség = 20;
protected int cellaMagasság = 20;
protected int szélesség = 20;
protected int magasság = 10;
protected int várakozás = 1000;
private java.awt.Robot robot;
private boolean pillanatfelvétel = false;
private static int pillanatfelvételSzámláló = 0;

public Sejtautomata(int szélesség, int magasság) {
        this.szélesség = szélesség;
        this.magasság = magasság;
        rácsok[0] = new boolean[magasság][szélesség];
        rácsok[1] = new boolean[magasság][szélesség];
        rácsIndex = 0;
        rács = rácsok[rácsIndex];
        for(int i=0; i<rács.length; ++i)
            for(int j=0; j<rács[0].length; ++j)
                rács[i][j] = HALOTT;
        siklóKilövő(rács, 5, 60);
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent e) {
                setVisible(false);
                System.exit(0);
        }
});

addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyPressed(java.awt.event.KeyEvent e) {
                if(e.getKeyCode() == java.awt.event.KeyEvent.VK_K) {
                    cellaSzélesség /= 2;
                    cellaMagasság /= 2;
                    setSize(Sejtautomata.this.szélesség*cellaSzélesség,
                            Sejtautomata.this.magasság*cellaMagasság);
                    validate();
            } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_N) {
                    cellaSzélesség *= 2;
                    cellaMagasság *= 2;
                    setSize(Sejtautomata.this.szélesség*cellaSzélesség,
                            Sejtautomata.this.magasság*cellaMagasság);
                    validate();
            } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_S)
                    pillanatfelvétel = !pillanatfelvétel;
                else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_G)
                    várakozás /= 2;
                else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_L)
                    várakozás *= 2;
```

```
                repaint();
        }
});

addMouseListener(new java.awt.event.MouseAdapter() {
            public void mousePressed(java.awt.event.MouseEvent m) {
                int x = m.getX()/cellaSzélesség;
                int y = m.getY()/cellaMagasság;
                rácsok[rácsIndex][y][x] = !rácsok[rácsIndex][y][x];
                repaint();
    }
});

addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {
            public void mouseDragged(java.awt.event.MouseEvent m) {
                int x = m.getX()/cellaSzélesség;
                int y = m.getY()/cellaMagasság;
                rácsok[rácsIndex][y][x] = ÉLŐ;
                repaint();
    }
});

cellaSzélesség = 10;
cellaMagasság = 10;
try {
            robot = new java.awt.Robot(
                    java.awt.GraphicsEnvironment.
                    getLocalGraphicsEnvironment().
                    getDefaultScreenDevice());
    } catch(java.awt.AWTException e) {
            e.printStackTrace();
}

setTitle("Sejtautomata");
setResizable(false);
setSize(szélesség*cellaSzélesség,
                magasság*cellaMagasság);
setVisible(true);
new Thread(this).start();
}

public void paint(java.awt.Graphics g) {
        boolean [][] rács = rácsok[rácsIndex];
        for(int i=0; i<rács.length; ++i) {
            for(int j=0; j<rács[0].length; ++j) {
                if(rács[i][j] == ÉLŐ)
                    g.setColor(java.awt.Color.BLACK);
                else
                    g.setColor(java.awt.Color.WHITE);
                g.fillRect(j*cellaSzélesség, i*cellaMagasság,
```

```
                        cellaSzélesség, cellaMagasság);
                g.setColor(java.awt.Color.LIGHT_GRAY);
                g.drawRect(j*cellaSzélesség, i*cellaMagasság,
                        cellaSzélesség, cellaMagasság);
    }
}

        if(pillanatfelvétel) {
            pillanatfelvétel = false;
            pillanatfelvétel(robot.createScreenCapture
                    (new java.awt.Rectangle
                    (getLocation().x, getLocation().y,
                    szélesség*cellaSzélesség,
                    magasság*cellaMagasság)));
    }
}

public int szomszédokSzáma(boolean [][] rács,
            int sor, int oszlop, boolean állapot) {
        int állapotúSzomszéd = 0;
        for(int i=-1; i<2; ++i)
            for(int j=-1; j<2; ++j)
                if(!((i==0) && (j==0))) {
            int o = oszlop + j;
            if(o < 0)
                o = szélesség-1;
            else if(o >= szélesség)
                o = 0;

            int s = sor + i;
            if(s < 0)
                s = magasság-1;
            else if(s >= magasság)
                s = 0;

            if(rács[s][o] == állapot)
                ++állapotúSzomszéd;
                }

        return állapotúSzomszéd;
}

public void időFejlődés() {

        boolean [][] rácsElőtte = rácsok[rácsIndex];
        boolean [][] rácsUtána = rácsok[(rácsIndex+1)%2];

        for(int i=0; i<rácsElőtte.length; ++i) {
            for(int j=0; j<rácsElőtte[0].length; ++j) {
```

```
                int élők = szomszédokSzáma(rácsElőtte, i, j, ÉLŐ);

                if(rácsElőtte[i][j] == ÉLŐ) {

                    if(élők==2 || élők==3)
                        rácsUtána[i][j] = ÉLŐ;
                    else
                        rácsUtána[i][j] = HALOTT;
        }  else {

                    if(élők==3)
                        rácsUtána[i][j] = ÉLŐ;
                    else
                        rácsUtána[i][j] = HALOTT;
            }
        }
    }
        rácsIndex = (rácsIndex+1)%2;
}

public void run() {

        while(true) {
            try {
                Thread.sleep(várakozás);
            } catch (InterruptedException e) {}

            időFejlődés();
            repaint();
        }
}

public void sikló(boolean [][] rács, int x, int y) {

        rács[y+ 0][x+ 2] = ÉLŐ;
        rács[y+ 1][x+ 1] = ÉLŐ;
        rács[y+ 2][x+ 1] = ÉLŐ;
        rács[y+ 2][x+ 2] = ÉLŐ;
        rács[y+ 2][x+ 3] = ÉLŐ;

}

public void siklóKilövő(boolean [][] rács, int x, int y) {

        rács[y+ 6][x+ 0] = ÉLŐ;
        rács[y+ 6][x+ 1] = ÉLŐ;
        rács[y+ 7][x+ 0] = ÉLŐ;
        rács[y+ 7][x+ 1] = ÉLŐ;

        rács[y+ 3][x+ 13] = ÉLŐ;
```

```
        rács[y+ 4][x+ 12] = ÉLŐ;
        rács[y+ 4][x+ 14] = ÉLŐ;

        rács[y+ 5][x+ 11] = ÉLŐ;
        rács[y+ 5][x+ 15] = ÉLŐ;
        rács[y+ 5][x+ 16] = ÉLŐ;
        rács[y+ 5][x+ 25] = ÉLŐ;

        rács[y+ 6][x+ 11] = ÉLŐ;
        rács[y+ 6][x+ 15] = ÉLŐ;
        rács[y+ 6][x+ 16] = ÉLŐ;
        rács[y+ 6][x+ 22] = ÉLŐ;
        rács[y+ 6][x+ 23] = ÉLŐ;
        rács[y+ 6][x+ 24] = ÉLŐ;
        rács[y+ 6][x+ 25] = ÉLŐ;

        rács[y+ 7][x+ 11] = ÉLŐ;
        rács[y+ 7][x+ 15] = ÉLŐ;
        rács[y+ 7][x+ 16] = ÉLŐ;
        rács[y+ 7][x+ 21] = ÉLŐ;
        rács[y+ 7][x+ 22] = ÉLŐ;
        rács[y+ 7][x+ 23] = ÉLŐ;
        rács[y+ 7][x+ 24] = ÉLŐ;

        rács[y+ 8][x+ 12] = ÉLŐ;
        rács[y+ 8][x+ 14] = ÉLŐ;
        rács[y+ 8][x+ 21] = ÉLŐ;
        rács[y+ 8][x+ 24] = ÉLŐ;
        rács[y+ 8][x+ 34] = ÉLŐ;
        rács[y+ 8][x+ 35] = ÉLŐ;

        rács[y+ 9][x+ 13] = ÉLŐ;
        rács[y+ 9][x+ 21] = ÉLŐ;
        rács[y+ 9][x+ 22] = ÉLŐ;
        rács[y+ 9][x+ 23] = ÉLŐ;
        rács[y+ 9][x+ 24] = ÉLŐ;
        rács[y+ 9][x+ 34] = ÉLŐ;
        rács[y+ 9][x+ 35] = ÉLŐ;

        rács[y+ 10][x+ 22] = ÉLŐ;
        rács[y+ 10][x+ 23] = ÉLŐ;
        rács[y+ 10][x+ 24] = ÉLŐ;
        rács[y+ 10][x+ 25] = ÉLŐ;

        rács[y+ 11][x+ 25] = ÉLŐ;

}

public void pillanatfelvétel(java.awt.image.BufferedImage felvetel) {
```

```
        StringBuffer sb = new StringBuffer();
        sb = sb.delete(0, sb.length());
        sb.append("sejtautomata");
        sb.append(++pillanatfelvételSzámláló);
        sb.append(".png");
        try {
            javax.imageio.ImageIO.write(felvetel, "png",
                    new java.io.File(sb.toString()));
        } catch(java.io.IOException e) {
            e.printStackTrace();
        }
}

public void update(java.awt.Graphics g) {
        paint(g);
}

public static void main(String[] args) {
        new Sejtautomata(100, 75);
    }
}
```

5.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

5.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása:

A BrainB Benchmark szoftver a jövő kiemelkedő e-sportolóinak korai felismerésében, felkutatásában hivatott segíteni. A mérőprogram arra a játékélményre, jelenségre épül, mely során a játékos átmenetileg elveszíti a karakterét az intenzív, komplex pillanatokban. A vizuális effektek kitakarják egymást, így idővel az egyszerű szoftverekben is nehézkessé válhat a karakterünk irányítása. Ebben a programban a 'Samu Entropy' karaktert (dobozt) kell figyelnünk: akkor eredményes a mérés, ha sikerül a doboz közepén lévő kék pöttyön tartanunk a kurzort. Az értékek növekedésével egyre több doboz (vizuális elem) jelenik meg a képernyőn, a dobozok pedig egyre gyorsabban kezdenek el mozogni. Ha a játékos a mérőprogram használata során egy adott pillanattól kezdve 1200 ms-on (több mint 1 másodpercen) keresztül a karakteren tudja tartani az egérmutatót, akkor a játékos a benchmark szerint megtalálta (FOUND) a karaktert. Ha több, mint 1 másodpercig nincs kapcsolata akkor a játékos elvesztette (LOST) a karaktert.

6. fejezet

Helló, Schwarzenegger!

6.1. Szoftmax Py MNIST

Python

Megoldás videó: https://youtu.be/j7f9SkJR3oc

Megoldás forrása: https://github.com/tensorflow/tensorflow/releases/tag/v0.9.0 (/tensorflow-0.9.0/tensorflow/exa
https://progpater.blog.hu/2016/11/13/hello_samu_a_tensorflow-bol

A TensorFlow egy szoftverkönyvtár, gépi tanulási algoritmusok leírására és végrehajtására. Roppant flexibilis, nagyon széles körű algoritmusok megvalósítására alkalmas, például a beszédfelismerésben, a robotikában, az információ kinyerésben, a számítógépek elleni támadások felderítésében, és az agykutatásban.

A TensorFlow számítást egy irányított gráf írja le. Az adatáramlás a gráf élei mentén történik. A gráfban mindegyik csúcs egy műveletet reprezentálhat és mindegyik csúcsnak lehet nulla vagy több inputja, ugyanígy nulla vagy több outputja. A gráf normál élei mentén áramló értékek tenzorok, tetszőleges dimenziójú vektorok.

A feladatban a TensorFlow segítségével készítünk fel egy modellt kézírású számjegyek felismerésére. Ehhez az MNIST adatállományt is felhasználjuk, ami kézírással írt számjegyek képeit tartalmazza.

6.2. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

6.3. Minecraft-MALMÖ

Megoldás videó: https://youtu.be/bAPSu3Rndi8

Megoldás forrása:

A Project Malmo egy olyan platform, amely segítségével kutatásokat végezhetünk a mesterséges intelligencia világában, méghozzá a népszerű játék, a Minecraft használatával. Olyan Python ágenst készítünk, amely egy alapértelmezett biomban gazdag világban az akadályokat legyőzve, jobbra-balra tartva felfedezi a világot.

Először meg kell látogatnunk a hivatalos GitHub repót, ahol láthatjuk, hogy a telepítés elvégezhető akár a Python Pip Package segítségével (amit ugyebár az első feladatban használtunk), de dolgozhatunk pre-built verzióval is, oprendszertől függetlenül. Én ezúttal Windows oprendszeren dolgoztam, és a pre-built fájlokat töltöttem le, melyeket a forráskönyvtárban mellékeltem is.

7. fejezet

Helló, Chaitin!

7.1. Iteratív és rekurzív faktoriális Lisp-ben

Megoldás videó:

Megoldás forrása:

7.2. Weizenbaum Eliza programja

Éleszd fel Weizenbaum Eliza programját!

Megoldás videó:

Megoldás forrása:

7.3. Gimp Scheme Script-fu: króm effekt

Írj olyan script-fu kiterjesztést a GIMP programhoz, amely megvalósítja a króm effektet egy bemenő szövegre!

Megoldás videó: https://youtu.be/OKdAkI_c7Sc

Megoldás forrása: https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/GIMP_Lisp/Chrome

Tanulságok, tapasztalatok, magyarázat...

7.4. Gimp Scheme Script-fu: név mandala

Írj olyan script-fu kiterjesztést a GIMP programhoz, amely név-mandalát készít a bemenő szövegből!

Megoldás videó: https://bhaxor.blog.hu/2019/01/10/a_gimp_lisp_hackelese_a_scheme_programozasi_nyelv

Megoldás forrása: https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/GIMP_Lisp/Mandala

Tanulságok, tapasztalatok, magyarázat...

7.5. Lambda

Hasonlítsd össze a következő programokat!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.6. Omega

Megoldás videó:

Megoldás forrása:

8. fejezet

Helló, Gutenberg!

8.1. Juhász István: Magas szintű programozási nyelvek 1

Alapfogalmak: Tutor: Ország Levente Máté A magas szintű nyelv a számítógépek programozására kialakult nyelvek egyik szintje. Forrásszövegnek nevezzük a magas szintű programozási nyelven írt programot. Kétféle technikával is készíthetünk gépi nyelvű programot a forrásprogramból: fordítóprogramos és interpreteres technikával. Én személy szerint eddig csak fordítóprogrammal alakítottam ki a programjaim végleges formáját (GNU Compiler), szóval az interpreteres technikáról még csak nem is hallottam korábban.

A fordítóprogram a következő lépéseket hajtja végre: lexikális elemzés (darabolás egységekre), szintaktikai elemzés (teljesülnek-e a nyelv szabályai), szemantikai elemzés és kódgenerálás. Míg a fordítóprogramos technikánál készül tárgyprogram, addig az interpreteres technikánál nem, ez a fő eltérés. A programozási nyelvek saját szabványait hivatkozási nyelvnek nevezzük (szintaktikai + szemantikai szabályok).

Következik ezután a programnyelvek osztályozása: vannak imperatív, deklaratív és egyéb nyelvek. Imperatív nyelvek esetén a program utasítások sorozata és a legfőbb programozói eszköz a változó. Az algoritmus működteti a processzort. Én elsősorban imperatív nyelveken szoktam programozni (C - eljárásorientált, C++ - objektumorientált). A deklaratív nyelvek nem algoritmikus nyelvek, a programozónak nincs lehetősége memóriaműveletekre (vagy csak korlátozott módon).

Adattípusok: Az adattípus konkrét programozási eszközök komponenseként jelenik meg. Egyik meghatározó tényezője a tartomány, amely olyan elemeket tartalmaz, amiket felvehet értékként egy programozási eszköz. Fontosak a műveletek is, amelyeket az elemeken tudunk végrehajtani. A harmadik meghatározó dolog az adattípusok világában a reprezentáció, azaz egyfajta belső ábrázolási mód. Saját reprezentációt megadni azonban csak nagyon kevés programozási nyelvben lehet (pl. Ada).

Az adattípusok két nagy csoportja az egyszerű és az összetett adattípusok. Mindkét csoportra hoz példát a könyv, ezek közül egy jópárat már ismertem én is. Az egyszerű típusok közül az egész, valós, karakteres és a logikai típus már a programozás-tanulás korai fázisában előjön. Ugyanez elmondható a tömbről is, mely ugyan az összetett típusokhoz tartozik, de egy- és kétdimenziós változatait gyakran használjuk, szinte a kezdetektől fogva.

A mutató típus értéke egy tárbeli cím, illetve elérhetünk vele egy megcímzett területen elhelyezkedő értéket is.

A nevesített konstans: Olyan programozási eszköz, amelynek három komponense van: név, típus és érték.

C-ben van beépített nevesített konstans, de FORTRAN-ban pl. nem volt. A legegyszerűbb a #define név literál makró használata. Ekkor az előfordító a forrásprogramban a név minden előfordulását helyettesíti a literállal.

Változó: Az imperatív programozási nyelvek fő eszköze, négy komponense van: név, attribútumok, cím, érték. A változó mindig a nevével jelenik meg, a másik három komponenst a névhez rendeljük hozzá.

Az attribútumok a változó futás közbeni viselkedését határozzák meg, a cím pedig a tárnak azt a részét, ahol a változó értéke elhelyezkedik. Ha két különböző névvel rendelkező változónak a futási idő egy adott pillanatában azonos a címkomponense és az értékkomponense is, akkor beszélhetünk többszörös tárhivatkozásról. Az eljárásorientált nyelvek leggyakoribb utasítása az értékadó utasítás pl. C: változónév = kifejezés;

Kifejezések: Szintaktikai eszközök, két komponensük az érték és a típus. Összetevői az operandusok, az operátorok és a kerek zárójelek. Létezik egyoperandusú (unáris), kétoperandusú (bináris) és háromoperandusú (ternáris) operátor is.

A kifejezések három alakja a prefix (pl. * 3 5 - operátor, operandusok), az infix (pl. 3 * 5 - operandus, operátor, operandus) és a postfix (pl. 3 5 * - operandusok, operátor). A kifejezés kiértékelésének nevezzük azt a folyamatot, amikor a kifejezés értéke és típusa meghatározódik.

A kiértékelés szempontjából a logikai operátorokkal rendelkező kifejezések speciálisak, ugyanis előfordulhat, hogy nem végezzük el az összes műveletet. Pl. ha ÉS műveletnél az első operandus hamis, akkor mindenképp hamis lesz a végeredmény. Két programozási eszköz típusa azonos deklaráció-, név- és struktúra egyenértékűség esetén.

A C egy kifejezésorientált nyelv. A mutató típus tartományának elemeivel összeadás és kivonás végezhető. A tömb típusú eszköz neve mutató típusú.

```
//Operátorok:
() - függvényoperátor;
[] - tömboperátor;
. - minősítő operátor;
-> - mutatóval történő minősítés operátora;
* - indirekciós operátor;
! - egyoperandusú operátor;
~ - egyes komplemens operátora;
* - szorzás operátora;
/ - osztás operátora;
% - maradékképzés operátora;
+ - összeadás operátora;
- - kivonás operátora;
<< >> - léptető operátorok;
< > <= >= = != - hasonlító operátorok;
& ^ | - nem rövidzár logikai operátorok;
&& || - rövidzár logikai operátorok;
? : - háromoperandusú operátor.
```

Utasítások: Olyan egységek, amelyekkel megadjuk az algoritmusok lépéseit és amely segítségével a fordítóprogram tárgyprogramot generálhat. Két nagy csoportjuk a deklarációs és a végrehajtható utasítások.

A deklarációs utasítások mögött nem áll tárgykód. A tárgykód a végrehajtható utasításokból generálódik, fordítóprogram segítségével. Az értékadó utasítás feladata beállítani vagy módosítani a változók értékkomponensét. Az üres utasítások hatására a processzor egy üres gépi utasítást hajt végre. Az ugró utasítás

(GOTO címke) a program egy adott pontjáról egy adott címkével ellátott végrehajtható utasításra adja át a vezérlést.

A kétirányú elágaztató utasítás (IF feltétel THEN tevékenység [ELSE tevékenység]) arra szolgál, hogy a program egy adott pontján két tevékenység közül válasszunk, illetve egy adott tevékenységet végrehajtsunk vagy sem. ELSE-ág hiánya esetén rövid kétirányú elágaztató utasításról beszélünk.

A többirányú elágaztató utasítás arra szolgál, hogy a program egy adott pontján egymást kölcsönösen kizáró tevékenységek közül egyet végrehajtsunk.

```
C-ben:
SWITCH (kifejezés) {
CASE egész_konstans_kifejezés: [tevékenység]
[CASE egész_konstans_kifejezés: [tevékenység]]...
[DEFAULT: tevékenység]
};
```

A ciklusszervező utasítások lehetővé teszik, hogy a program egy adott pontján egy bizonyos tevékenységet akárhányszor megismételjünk. Felépítése: fej, mag, vég.

A feltételes ciklusnál az ismétlődést az igaz vagy hamis érték határozza meg. A feltétel a fejben vagy a végben szerepel, a mag az ismételendő végrehajtható utasításokat tartalmazza. Az előírt lépésszámú ciklusnál a ciklusparaméterek a fejben vannak. A változó által felvett értékekre fut le a ciklusmag. A ciklusváltozó a tartománynak vagy minden elemét fölveheti, vagy csak a tartományban szabályosan elhelyezkedő bizonyos értékeket.

A felsorolásos ciklusnak van ciklusváltozója, és minden felvett érték mellett lefut a mag. A végtelen ciklusnál sem a fejben, sem a végben nincs információ az ismétlődésre vonatkozóan. Az összetett ciklus pedig az előző ciklusfajták kombinációja.

```
C-ben:
//Kezdőfeltételes ciklus:
WHILE(feltétel) végrehajtható_utasítás
//Végfeltételes ciklus:
DO végrehajtható_utasítás WHILE(feltétel);
//FOR-ciklus:
FOR([kifejezés1]; [kifejezés2]; [kifejezés3]) végrehajtható_utasítás
//Vezérlő utasítások:
CONTINUE;
BREAK;
RETURN[kifejezés];
```

A programok szerkezete: Az eljárásorientált programnyelvekben a program szövege programegységekre tagolható. A program vagy fizikailag önálló részekből áll (külön-külön fordíthatóak), vagy egyetlen egységként kell lefordítani, ez nyelvfüggő. A kettő kombinációja is előfordulhat.

Az eljárásorientált nyelvek programegységei: alprogram, blokk, csomag, taszk. Az alprogram egy bemeneti adatcsoportot képez le egy kimeneti adatcsoportra úgy, hogy egy specifikáció megadja az adatok leírását. Az alprogram akkor alkalmazható, ha a program különböző pontjain ugyanaz a programrész megismétlődik. Felépítése: fej vagy specifikáció, törzs vagy implementáció, vég.

Az alprogram komponensei: név, formális paraméterlista, törzs, környezet. A név a fejben szereplő azonosító. A formális paraméterlistában azonosítók szerepelnek. A törzsben deklarációs és végrehajtható utasítások szerepelnek. A környezet alatt a globális változók együttesét értjük.

Az alprogramok két fajtája az eljárás és a függvény. Az eljárás valamilyen tevékenységet hajt végre, a függvény egyetlen értéket határoz meg. Hívási láncról akkor beszélünk, ha egy programegység meghív egy másik programegységet, az egy programegységet, és így tovább.

Amikor egy aktív alprogramot hívunk meg, akkor rekurzióról beszélünk. A rekurzió lehet közvetlen (önmagát hívja meg) vagy közvetett (korábban szereplő alprogramot hívunk meg).

A blokk olyan programegység, amely csak másik programegység belsejében helyezkedhet el. Van kezdete, törzse, vége. Bárhol elhelyezhető, ahol végrehajtható utasítás állhat. Csak eljárásorientált nyelveknek egy része ismeri.

Paraméterek: Akkor beszélünk paraméterkiértékelésről, amikor egy alprogram hívásánál egymáshoz rendelődnek a formális- és aktuális paraméterek, és meghatározódnak azok az információk, amelyek a paraméterátadásnál a kommunikációt szolgálják. Mindig a formális paraméterlista az elsődleges, egy darab van belőle. Aktuális paraméterlista viszont annyi lehet, ahányszor meghívjuk az alprogramot.

Az, hogy melyik formális paraméterhez melyik aktuális paraméter fog hozzárendelődni, függ a kötéstől. Sorrendi kötés esetén a formális paraméterekhez a felsorolás sorrendjében rendelődnek hozzá az aktuális paraméterek. Név szerinti kötés esetén az aktuális paraméterlistában határozhatjuk meg az egymáshoz rendelést úgy, hogy megadjuk a formális paraméter nevét és mellette valamilyen szintaktikával az aktuális paramétert. Ha a formális paraméterek száma fix, akkor a paraméterkiértékelés kétféle módon mehet végbe. Vagy meg kell egyeznie az aktuális paraméterek számának a formális paraméterek számával, vagy kevesebb lehet, mint a formális paraméterek száma.

A programozási nyelvek egy része a típusegyenértékűséget vallja, ekkor az aktuális paraméter típusának azonosnak kell lennie a formális paraméter típusával. A programozási nyelvek másik része azt mondja, hogy az aktuális paraméter típusának konvertálhatónak kell lennie a formális paraméter típusára.

A paraméterátadás az alprogramok és más programegységek közötti kommunikáció egy formája. Mindig van egy hívó (tetszőleges programegység) és egy hívott (alprogram). A nyelvek a következő paraméterátadási módokat ismerik: érték szerinti, cím szerinti, eredmény szerinti, érték-eredmény szerinti, név szerinti, szöveg szerinti.

Az érték szerinti paraméterátadás esetén a formális paramétereknek van címkomponensük a hívott alprogram területén. A cím szerinti paraméterátadásnál a formális paramétereknek nincs címkomponensük a hívott alprogram területén. Az eredmény szerinti paraméterátadásnál a formális paramétereknek van címkomponense a hívott alprogram területén, az aktuális paraméternek pedig lennie kell címkomponensének.

Az érték-eredmény szerinti paraméterátadásnál a formális paraméternek van címkomponense a hívott területén és az aktuális paraméternek rendelkeznie kell érték- és címkomponenssel. A név szerinti paraméterátadásnál az aktuális paraméter egy, az adott szövegkörnyezetben értelmezhető tetszőleges szimbólumsorozat lehet. A szöveg szerinti paraméterátadás a név szerintinek egy változata, annyiban különbözik tőle, hogy a hívás után az alprogram elkezd működni, az aktuális paraméter értelmező szövegkörnyezetének rögzítése és a formális paraméter felülírása csak akkor következik be, amikor a formális paraméter neve először fordul elő az alprogram szövegében a végrehajtás folyamán.

Az alprogramok formális paramétereit három csoportba sorolhatjuk: input (pl. érték szerinti), output (pl. eredmény szerinti), input-output (pl. érték-eredmény szerinti).

Blokk és hatáskör: A blokk programegység, amely csak másik programegység belsejében helyezkedhet el, külső szinten nem állhat. Van kezdete, törzse és vége. A kezdetet és a véget egy-egy speciális karaktersorozat vagy alapszó jelzi. A törzsben lehetnek deklarációs és végrehajtható utasítások. A blokknak nincs paramétere és bárhol elhelyezhető, ahol végrehajtható utasítás állhat.

Egy név hatásköre alatt értjük a program szövegének azon részét, ahol az adott név ugyanazt a programozási eszközt hivatkozza, tehát jelentése, felhasználási módja, jellemzői azonosak. Azt a tevékenységet, amikor egy név hatáskörét megállapítjuk, hatáskörkezelésnek hívjuk.

A statikus hatáskörkezelés fordítási időben történik, a fordítóprogram végzi. Azt a nevet, amely egy adott programegységben nem lokális név, de onnan látható, globális névnek hívjuk. A dinamikus hatáskörkezelés futási idejű tevékenység, a futtató rendszer végzi.

Az eljárásorientált nyelvek a statikus hatáskörkezelést valósítják meg. A C nyelv a függvényt és a blokkot ismeri.

```
//A blokk alakja:
{
  deklarációk
  végrehajtható_utasítások
}
```

Absztrakt adattípus: Olyan adattípus, amely megvalósítja a bezárást vagy információrejtést. Ezen adattípusnál nem ismerjük a reprezentációt és a műveletek implementációját.

Az ilyen típusú programozási eszközök értékeihez csak szabályozott módon, a műveleteinek specifikációi által meghatározott interfészen keresztül férhetünk hozzá. Az elmúlt évtizedekben az ADT a programnyelvek egyik legfontosabb fogalmává vált és alapvetően befolyásolta a nyelvek fejlődését.

Generikus programozás: Az újrafelhasználhatóság és a procedurális absztrakció eszköze. Bármely programozási nyelvbe beépíthető ilyen eszközrendszer.

Lényege, hogy egy paraméterezhető forrásszöveg-mintát adunk meg. A mintaszöveget a fordító kezeli, előállítható belőle egy konkrét szöveg, ami aztán lefordítható. A generikus formális paramétereinek száma mindig fix.

Input/output: Az I/O platform-, operációs rendszer-, implementációfüggő. Az az eszközrendszer a programnyelvekben, amely a perifériákkal történő kommunikációért felelős, amely az operatív tárból oda küld adatokat, vagy onnan vár adatokat.

Középpontjában az állomány áll. A logikai állomány egy olyan programozási eszköz, amelynek neve van, és amelynél pl. a rekordfelépítés, rekordformátum, elérés, szerkezet stb. attribútumként jelenik meg. A fizikai állomány pedig a szokásos operációs rendszer szintű, konkrét, a perifériákon megjelenő, az adatokat tartalmazó állomány. Egy állomány funkció szerint lehet input, output vagy input-output.

Kétféle adatátviteli mód létezik: a folyamatos és a bináris/rekord módú. Formátumos módú adatátvitelnél minden egyes egyedi adathoz a formátumok segítségével explicit módon meg kell adni a kezelendő karakterek darabszámát és a típust. Szerkesztett módú adatátvitelnél minden egyes egyedi adathoz meg kell adni egy maszkot, amely szerkesztő és átviendő karakterekből áll. Listázott módú adatátvitelnél a folytonos karaktersorozatban vannak a tördelést végző speciális karakterek, amelyek az egyedi adatokat elhatárolják egymástól, a típusra nézve pedig nincs explicit módon megadott információ.

Ha egy programban állományokkal akarunk dolgozni, akkor a következőket kell végrehajtanunk: deklaráció, összerendelés, állomány megnyitása, feldolgozás, lezárás. C-ben az I/O eszközrendszer nem része a nyelvnek, standard könyvtári függvények állnak rendelkezésre. Létezik a bináris és a folyamatos módú átvitel, utóbbinál egy formátumos és egy szerkesztett átvitel keverékeként.

8.2. Kernigan-Ritchie: A C programozási nyelv

Vezérlési szerkezetek: Egy nyelv vezérlésátadó utasításai az egyes műveletek végrehajtási sorrendjét határozzák meg. Egy olyan kifejezés, mint x = 0, i++ vagy printf(...) utasítássá válik, ha egy pontosvesszőt írunk utána. A C nyelvben ugyanis a pontosvessző az utasításlezáró jel (terminátor).

A {} kapcsos zárójelekkel deklarációk és utasítások csoportját tudjuk összefogni egyetlen összetett utasításba vagy blokkba. Pl. if, else, while, for utasítások utáni utasításokat összefogó zárójelpár. Az if-else utasítást döntés kifejezésére használjuk.

Az utasítás először kiértékeli a kifejezést, és ha ennek értéke igaz, akkor az 1. utasítást hajtja végre. Ha hamis és van else rész, akkor a 2. utasítás hajtódik végre.

```
if (kifejezés)
  1. utasítás
else
  2. utasítás
//az else rész opcionális
```

Az else-if utasítás adja a többszörös döntések programozásának egyik legáltalánosabb lehetőségét. A gép sorra kiértékeli a kifejezéseket és ha bármelyik ezek közül igaz, akkor végrehajtja a megfelelő utasítást, majd befejezi a láncot.

```
if (kifejezés)
  utasítás
else if (kifejezés)
  utasítás
.
.
.
else
  utasítás
```

A switch utasítás úgy működik, hogy összehasonlítja egy kifejezés értékét több egész értékű állandó kifejezés értékével, és az ennek megfelelő utasítást hajtja végre. A case ágakban egy egész állandó vagy állandó egész értékű kifejezés található, és ha ennek értéke megegyezik a switch utáni kifejezés értékével, akkor végrehajtódnak az utasítások. A default ág opcionális, akkor hajtódik végre, ha egyetlen case ághoz tartozó feltétel sem teljesült.

```
switch (kifejezés) {
  case állandó kifejezés: utasítások
  case állandó kifejezés: utasítások
  .
  .
  .
  default: utasítások
}
```

A while ciklus esetén a program először kiértékeli a kifejezést. A ciklus addig folytatódik, amíg a kifejezés nullává (false) nem válik.

```
while (kifejezés)
  utasítás
```

A for utasítás mindhárom komponense kifejezés. Egyszerű inicializálás és újrainicializálás esetén előnyösebb a while ciklusnál, mivel a ciklust vezérlő utasítások jól látható formában helyezkednek el.

```
for (1. kifejezés, 2. kifejezés, 3. kifejezés)
  utasítás
```

A harmadik ciklusszervező utasítás a do-while. A gép először végrehajtja az utasítást és csak utána értékeli a kifejezést. Addig tart a ciklus, amíg a kifejezés értéke hamis nem lesz.

```
do
  utasítás
while (kifejezés);
```

A break utasítás lehetővé teszi a for, while vagy do utasításokkal szervezett ciklusok idő előtti elhagyását, valamint a switch utasításból való kilépést. A ciklusmagban található continue utasítás hatására azonnal megkezdődik a következő iterációs lépés.

A goto utasítással megadott címkékre ugorhatunk. Használatának egyik legelterjedtebb esete, amikor több szinten egymásba ágyazott szerkezet belsejében kívánjuk abbahagyni a feldolgozást és egyszerre több, egymásba ágyazott ciklusból szeretnénk kilépni.

Utasítások: Végrehajtásuk a hatásukban nyilvánul meg és nem rendelkeznek értékkel.

```
utasítás:
  címkézett_utasítás
  kifejezésutasítás
  összetett_utasítás
  kiválasztó_utasítás
  iterációs_utasítás
  vezérlésátadó_utasítás
```

Az utasításokhoz előtagként megadott címke tartozhat.

```
címkézett_utasítás
  azonosító: utasítás
  case állandó_kifejezés: utasítás
  default: utasítás
```

A címkék önmagukban nem módosítják az utasítások végrehajtásának sorrendjét. Az utasítások többsége kifejezésutasítás. Funkcióját tekintve a legtöbb kifejezésutasítás értékadás vagy függvényhívás.

```
kifejezésutasítás:
  kifejezés
```

A C nyelvben lehetőség van összetett utasításokra is, melyek több utasítás egyetlen utasításként kezelését teszi lehetővé.

```
összetett_utasítás:
  {deklarációs_lista utasítás_lista}
```

```
deklarációs_lista:
  deklaráció
  deklarációs_lista deklaráció
utasítás_lista:
  utasítás
  utasítás_lista utasítás
```

A kiválasztó utasítások minden esetben a lehetséges végrehajtási sorrendek egyikét választják ki.

```
kiválasztó_utasítás:
  if (kifejezés) utasítás
  if (kifejezés) utasítás else utasítás
  switch (kifejezés) utasítás
```

Az iterációs utasítások egy ciklust határoznak meg.

```
iterációs_utasítás:
  while (kifejezés) utasítás
  do utasítás while (kifejezés):
  for (kifejezés; kifejezés; kifejezés) utasítás
```

A vezérlésátadó utasítások a vezérlés feltétel nélküli átadására alkalmasak.

```
vezérlésátadó_utasítás:
  goto azonosító;
  continue;
  break;
  return kifejezés;
```

8.3. Benedek-Levendovszky: Szoftverfejlesztés C++ nyelven

A C és a C++ nyelv: A C++ nyelvben az üres paraméterlista egy void paraméter megadásával ekvivalens: a függvénynek nincs paramétere.

```
void f(void)
{
  // Az f függvény törzse...
}
```

A szabványos C++ nyelvben a main függvénynek két formája létezik:

```
int main()
{
  ...
}
```

és

```
int main(int argc, char* argv[])
{
  ...
}
//argc: parancssor-argumentumok száma
//argv: parancssor-argumentumok
```

A C++ nyelvben került bevezetésre a bool típus, ami logikai igaz/hamis értéket tud reprezentálni. Pl. bool success = false. A bool típus olvashatóbb kódot eredményez, lehetőség van a függvénynevek és operátorok bool és int típusokra vonatkozó túlterhelésére.

A C++ nyelvben a wchar_t beépített típus lett, így használatához a típusdefinícióra nincs szükség.

```
wchar_t c = L's';
wchar_t* text = L"sss";
```

A függvényeket a nevük és az argumentumlistájuk együttesen azonosítja, tehát a C++ nyelvben van lehetőség azonos nevű függvények létrehozására. A függvények argumentumainak alapértelmezett értékét is meg tudjuk már adni. Amennyiben a függvény hívásakor nem adunk meg értéket, a függvény az adott argumentum alapértelmezett értékével kerül meghívásra.

A C++ bevezette a referenciatípust, ami megszünteti a pointerek szerepét a cím szerinti paraméterátadásban.

```
#include <stdio.h>
void f (int& i)
{
  i = i + 2;
}
int main(void)
{
  int a = 0;
  f(a);
  printf("%d\n", a);
}
//Ebben a C++ programban látható, hogy nem kell a változó címét képezni és ↩
   az i szimbólumot ugyanolyan szintaxissal használhatjuk, mint egy int ↩
   típusú változót.
//A program kimenete 2 és egy soremelés lesz.
//Egy adott típusú referenciát a referencia neve elé írt & jellel ↩
   deklarálunk.
//Az & jel egyargumentumú operátor is egyben, amely a mögötte álló változó ↩
   címét adja vissza.
```

Operátorok és túlterhelésük: A C nyelvben az operátorok az argumentumaikon végeznek műveletet, az adott művelet eredményét a visszatérési értékük feldolgozásával használhatjuk. Az adott operátor mellékhatásának nevezzük az operátorok értékének megváltoztatását.

A c++ kifejezés esetén a c változó a ++ operátor argumentuma, az operátor visszatérési értéke a c változó eredeti értéke, mellékhatásként az operátor kiértékelése után a c változó értéke eggyel több lesz. A C++ nyelv a C-hez képest bevezet néhány új operátort (pl. hatókör-operátor ::, pointer-tag operátor .* és ->*).

A C nyelvben csak érték szerinti paraméterátadás van, nem tudjuk megváltoztatni egy változó értékét, csak ha a változóra mutató pointert adunk át. A C++-ban azonban a referencia szerinti paraméterátadás "mellékhatásra" is képes.

```
//A prefix ++ operátort kétféleképpen is meghívhatjuk:
++i; //hagyományos, operátor-írásmód
operator++ (i); //C++: a függvényszintaxis is megengedett
//A C++-ban az operátor kulcsszó, ezzel adjuk meg, hogy egy speciális  ↩
   függvényről van szó:
c = a + b; //hagyományos, operátor-írásmód
c = operator + (a, b); //az = operátor-, a + függvényszintaxissal
operator = (c, operator + (a, b)); //függvényszintaxis
```

C++ sablonok: Olyan osztálysablonok és függvénysablonok, melyek esetében az adott osztály, illetve függvény definiálásakor bizonyos elemeket nem adunk meg, hanem paraméterként kezelünk. Ezen paraméterek megadása explicit vagy implicit módon az adott osztálysablon, illetve függvénysablon felhasználásakor történik.

A C++ sablonok a generikus típusok C++ nyelvbeli megfelelői. Jellemző alkalmazási területük olyan tárolóosztályok (pl. dinamikusan nyújtózkodó tömb, láncolt lista stb.) létrehozása, amelyek tetszőleges típusú elem tárolására használhatók fel.

```
//A függvénysablonok legegyszerűbb felhasználási módja az implicit  ↩
   példányosítás:
int n = max(3, 5);
double d = max(2.3, 4.2);
//A fordító a paraméterek típusából kikövetkezteti, milyen típust kell  ↩
   behelyettesíteni a sablonparaméterek helyére, vagyis a paramétereket nem ↩
    adtuk meg explicit módon.
```

Implicit példányosítás esetén egy adott sablonparaméter csak egy típust jelölhet.

```
//Sablonparaméter nemcsak típus lehet, hanem típusos konstans is:
#include <iostream>
using namespace std;

template <int N>
int Square()
{
  return N*N;
}
int main()
{
  const int x = 10;
  cout << "Square of " << X << " is " << Square<X>() << endl;
}
//A Square függvénysablon sablonparaméterként egy int konstanst vár,  ↩
   amelynek négyzetével tér vissza.
```

A sablonparaméter megadása során a class helyett a typename kulcsszó is használható. Amennyiben egy adott függvénynek több implementációja létezik, felmerül a kérdés, hogy a függvényhívás során melyik függvény fog meghívódni.

Ha létezik olyan közönséges függvény, melynek paraméterei típus szerint pontosan megegyeznek, akkor az adott függvény hívódik meg. Ha létezik olyan függvénysablon, melynek paraméterei típus szerint pontosan megegyeznek, akkor az adott függvény hívódik meg. Ha létezik közönséges függvény vagy függvénysablon, mely esetében típuskonverzióval megegyeznek a paraméterek, akkor az adott függvény hívódik meg.

```
//Osztálydefinícióból sablondefiníciót a template kulcsszó használatával ↩
   készíthetünk:
template <class ItemType> class Fifo
{
...
};
```

A szabványos adatfolyamok: A C++ adatfolyamokban (stream) gondolkodik, amelyek bájtok sorozatát jelentik. Az adatfolyam típusától függően az istream típusú objektumok csak olvasható, bemeneti adatfolyamokat (input stream) takarnak, míg az ostream osztály példányai csak írható, kimeneti adatfolyamonként (output stream) használhatók.

A rendszerhívások költsége igen nagy, az adatfolyamokat egy bufferrel látják el, amelyeket az adatfolyambuffer osztályok példányai valósítanak meg. Az adatfolyambufferek összegyűjtik a karaktersorozatokat, és több cout kiírást egy rendszerhívással írnak ki a képernyőre. Az endl, illetve egy következő cin beolvasás automatikusan kiírja a couthoz tartozó buffereket. Ha szeretnénk kiüríteni a buffert, akkor a cout << flush; vagy cout.flush(); megoldásokat alkalmazhatjuk.

Az adatfolyam állapotát egy iostate típusú tagváltozó jelzi, amelynek állapotát az alábbi konstansokkal lehet beállítani: eofbit, failbit, badbit, goodbit. Ha a failbit be van állítva, az hibát jelent. A badbit komoly, fatális hibát jelent. Ha az eofbit be van állítva, akkor az adatfolyam elérte az állomány végét. A goodbit konstans azt jelzi, hogy a fenti hibák közül egy sem lépett fel, vagyis ez nem tekinthető önálló jelzőbitnek.

A C++ szabványos könyvtára tartalmaz egy string osztályt, amely szükség szerint változtatja a méretét: ha karaktereket fűzünk hozzá, automatikusan elvégzi a szükséges helyfoglalást. Ha egy ilyen sztringet szeretnénk beolvasni, természetesen használhatjuk a >> operátort. Ha olyan stringet szeretnénk beolvasni, amely szóközt is tartalmaz, akkor az std::getline függvényt használhatjuk.

Manipulátorok és formázás: Az adatfolyam-objektumoknak vannak tagfüggvényeik, amelyekkel beállíthatjuk az állapotát, adatokat olvashatunk és írhatunk, valamint egyéb műveleteket végezhetünk az adatfolyamon. Ugyanakkor az adatfolyamok manipulálásának nem ez az egyedüli módja: használhatunk úgynevezett manipulátorokat.

Az I/O manipulátor egy olyan adatfolyam-módosító speciális objektum, amelyet a szokásos kiviteli (<<) és bemeneti (>>) operátorok argumentumaként alkalmazunk az adatfolyamokra. Az előre definiált manipulátorok az #include <iomanip> állományban és az std névtérben találhatók.

Egy másik gyakran használt manipulátor pl. az endl, amely elhelyez egy sor vége karaktert az adatfolyamban, majd kiüríti a buffert. Az ends egy sztring vége karaktert helyez el az adatfolyamban. Vannak olyan manipulátorok, melyeknek van paramétere (pl. setprecision) és vannak, melyeknek nincs (pl. endl).

A jelzőbitek olyan bitek, amelyeket beállíthatunk vagy törölhetünk (pl. ios::fixed). Előfordulhat, hogy egy tulajdonságot több biten tárolunk, ilyenkor egy olyan bináris számot adunk meg, amit maszknak nevezünk. A C++ I/O-hoz kapcsolódó jelzőbitek az ios nevű osztályban vannak definiálva.

A normál alakos kiírást az ios::scientific jelzőbittel állíthatjuk be. Ha két tulajdonságot egyszerre szeretnénk állítani, az ios::floatfield maszkot használhatjuk.

Állománykezelés: A C++ az állománykezeléshez is adatfolyamokat használ, amelyeket ezúttal az ifstream (bemeneti állomány-adatfolyam), illetve az ofstream (kimeneti állomány-adatfolyam) osztályok reprezentálnak. A kétirányú adatfolyamot az fstream osztály valósítja meg.

Mivel az objektumorientált konstrukciók lehetővé teszik a konstruktorok és destruktorok használatát, az állományok megnyitását a konstruktorok végzik, lezárását pedig a destruktorok. Az állomány-adatfolyamosztályok definíciói az fstream fejlécfájlban találhatók az std névtérben. A megnyitandó állomány nevét a konstruktorban adhatjuk meg, a hibakezelés a már megszokott módon, az adatfolyam-objektum vizsgálatával végezhető.

Ha a konstruktor vagy a destruktor nem felel meg nekünk, az adatfolyam-objektumoknak létezik open, illetve close függvényük. Az istream és ostream adatfolyamok mindegyike megnyitható írásra és olvasásra. Az értelmezett műveletek azonban az adatfolyamok mögött lévő buffer típusától függenek.

Típuskonverziók: A C nyelvben az enum és az int típus között oda-vissza létezik implicit konverzió. Ezzel szemben C++-ban, ha enum típusra konvertálunk, ki kell írnunk a típuskonverziót.

```
enum days {Mon, Tue, Wed, Thu, Fri};
int main(void)
{
  enum days day = Mon;
  int d;
  //C: OK, C++: OK
  d = Mon;
  //C: OK, C++: hiba
  day = d;
  //C: OK, C++: OK
  day = (enum days)d;
}
```

Ha egy másik típusról szeretnénk konvertálni a mi osztályunk típusára, a konverziós konstruktor jelent megoldást. Ha az osztályunkról szeretnénk egy másik típusra konvertálni, akkor a konverziós operátor a megfelelő eszköz.

A C++ "szeletelés kapás" (slicing-on-the-fly) jelensége során az eredeti, konvertált objektum megmarad, mindössze a konverzió során nem másolódik át egy rész, bár az új objektum kétségkívül úgy néz ki, mintha a réginek levágták volna az alját. Mivel másolókonstruktor mindig van, ez a típusú konverzió szükség esetén automatikusan is végbemegy.

A konstans típuskonverzió képes egyedül konstans típust nem konstanssá tenni, illetve volatile típust nem azzá. Egyéb konverziókra nem alkalmazható. A dinamikus típuskonverzió az öröklési hierarchián lefelé történő konverziókhoz szükséges. Az újraértelmező típuskonverzió az implementációfüggő konverziók esetén használható.

III. rész

Második felvonás

Bátf41 Haxor Stream
A feladatokkal kapcsolatos élő adásokat sugároz a https://www.twitch.tv/nbatfai csatorna, melynek permanens archívuma a https://www.youtube.com/c/nbatfai csatornán található.

9. fejezet

Helló, Arroway!

9.1. A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

9.2. Java osztályok a Pi-ben

Az előző feladat kódját fejleszd tovább: vizsgáld, hogy Vannak-e Java osztályok a Pi hexadecimális kifejtésében!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

IV. rész

Irodalomjegyzék

9.3. Általános

[MARX] Marx, György, *Gyorsuló idő*, Typotex , 2005.

9.4. C

[KERNIGHANRITCHIE] Kernighan, Brian W. Ã©s Ritchie, Dennis M., *A C programozási nyelv*, Bp., Műszaki, 1993.

9.5. C++

[BMECPP] Benedek, Zoltán Ã©s Levendovszky, Tihamér, *Szoftverfejlesztés C++ nyelven*, Bp., Szak Kiadó, 2013.

9.6. Lisp

[METAMATH] Chaitin, Gregory, *META MATH! The Quest for Omega*, http://arxiv.org/PS_cache/math/-pdf/0404/0404335v7.pdf , 2004.

Köszönet illeti a NEMESPOR, https://groups.google.com/forum/#!forum/nemespor, az UDPROG tanulószoba, https://www.facebook.com/groups/udprog, a DEAC-Hackers előszoba, https://www.facebook.com/-groups/DEACHackers (illetve egyéb alkalmi szerveződésű szakmai csoportok) tagjait inspiráló érdeklődésükért és hasznos észrevételeikért.

Ezen túl kiemelt köszönet illeti az említett UDPROG közösséget, mely a Debreceni Egyetem reguláris programozás oktatása tartalmi szervezését támogatja. Sok példa eleve ebben a közösségben született, vagy itt került említésre és adott esetekben szerepet kapott, mint oktatási példa.