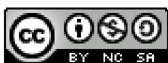
PROGRAMOZÁSI NYELVEK - CPP. ELŐADÁS JEGYZET

Szerkesztette: *Balogh Tamás*

2013. április 12.

Ha hibát találsz, kérlek jelezd a *info@baloghtamas.hu* e-mail címen!



Ez a Mű a Creative Commons Nevezd meg! - Ne add el! - Így add tovább! 3.0 Unported Licenc feltételeinek megfelelően szabadon felhasználható.

Előszó

A jegyzet nem teljes és nem hibátlan. Az esetlegesen előforduló hibákért, és kimaradt részekért felelőséget nem vállalok. A jegyzet megtanulása nem biztos, hogy elég a sikeres vizsgához.

Az előadás sorszáma a hetek sorszámával egyezik meg. A szünetek, ünnepnapok miatt elmaradt órákat is számítom. Így számomra könyebb összehangolni a többi előadással, gyakorlattal.

Előadó adatai:

Név: Pataki Norbert

E-mail: patakino@elte.hu

Követelmények: Vizsga

Konzultációk: PC3-as terem, hétfőn, csütörtökön 18:00-20:00

Vizsgák:

05. 21.
06. 04.
06. 11.
06. 18.
06. 25 UV
07. 02. UV

1. Előadás

Programozási paradigma:

paradigma: Milyen absztrakció mentén képezzük a részfeladatokat?

- Imperatív programozás
- Procedurális programozás (Fortran)
- Generikus programozás
- Objektum-orientált programozás
- Logikai programozás (Prolog)
- Funkcionális programozás
- Dekleratív programozás (SQL, HTML). Milyen eredményt szeretnénk elérni?

C++

A C++ multiparadigmás programnyelv, azaz több programozási paradigrmát is támogat: procedurális, objektum-orientált, generikus illetve **(?)**
Szabványai:

1998: C++ első szabványa
2003: Első szabvány módosítása
2011: Jelenlegi szabvány ("C++11")

2. Előadás

Fordítás:

1. Preprocesszálási fázis (előfordítás)
2. Nyeli fordítás
3. Öszeszerkesztés //Linker

Preprocesszálás

Preprocesszor: Szövegátalakító szoftver, search & replace, file-ok másolása, szövegrészek kivágása/bemutatása, direktívák feldolgozása (#)
Pl.:

```
#include <iostream> //Előre bekonfigurált include path-ban keresi a filet.
#include "complex.h"//Relatív az aktuális fileokhoz képest.
```

A következő két kódrészlet eredménye ugyanaz:

```
int v[20];
for(int i =0; i < 20; ++i)
{
  v[i] = i;
}
```

```
#define N 20

int v[N];
for(int i = 0; i < N; ++i)
{
  v[i] = i;
}
```

Ha módosítani akarunk a tömbünk méretén,az első esetben több helyen is át kell írni a kódot, viszont a második esetben elég csak a direktíva értéket megváltoztatni.

Makrók

A makrók preprocesszálási fázisban értékelődnek ki. Példák makrókra:

Abszolútérték

```
#define ABS(x) x<0?-1*x:x
  .
  .
  .
std::cout << ABS(-3); // -3<0? -1 * -3:-3 => 3

int b = -2;
std::cout << ABS(b-3); //b-3 < 0 ? => -1*b (:= c), c-3:b-3 => -1, 5
   helyett
```

A megoldás helytelensége a zárójelezés hiányából fakad. Javítsuk át a makrót:

```
#define ABS(x)((x)<0?-1*(x):(x))
```

Maximum érték két szám között

```
#define MAX(a,b) (a)<(b) ? (b) : (a)

int i = 6;
std::cout<<MAX(++i, 2); // i = - 8 lesz
```

Include-guard

complex.h

```
class Complex
{
double re, im;
//...
};
```

units.h

```
#include "complex.h"
.
.
.
.
```

main.cpp

```
#include "complex.h"
#include "units.h"
int main()
{complex x;
}
```

A probléma, hogy a `main.cpp` includeolja a *complex.h*-t és a *units.h*-t is, ami szintén includeolja a *complex.h*-t, így ützközés történik.
A megoldás, hogy úgynevezett *include-guard*-okat helyezünk el a forrásfájlainkban.

Include guard használata

```
#ifndef COMPLEX_H
#define COMPLEX_H
.
.
.
#endif
```

Linker

A linkelési fázisban a fordító összefésüli/összeszerkeszti a forrásfájlokat, és egy, futtatható állományt készít.
Létezik statikus, illetve dinamikus linkelés. Statikus linkelés során mindent feloldunk, míg dinamikus linkelés során (?)

3. előadás

Nyelvi fordítás

Tokenizálás

- Azonosítók: `std, cout`
- Kulcsszavak: `int, return`
- Konstans literálok: 0
- Szöveg konstans literálok: *"Hello world"*
- Operátorok: `<<, ::`
- Szeparátorok: `;, {, }`

Azonosítók

Betűvel kell kezdeni(vagy _ jellel), és betűvel, vagy számmal folytatni. Nem lehet kulcsszó
Pl.:`_auto, If, iF, labda47` helyes azonosítok, viszont `47labda, if` nem.
Egyes azonosítók fentartottak egy-egy szabványkönyvtárbeli azonosítónak. Mivel nem ismerjük az összes szabványkönyvtárbeli nevet, figyelni kell az azonosítóink megadásakor, mert könnyen névütközéshez vezethet, különösen ha `_név` formátumot adunk meg.

Kulcsszavak:

Pl: `if, return, double, class, auto, while`
Nem lehet használni őket azonosítóként, és kisbetűvel írjuk őket.

Konstans szövegiterál

"Hello" : `const char[6]`. 'H' 'e' 'l' 'l' 'o' '\0'
`const char * s = "Hell"; //Tömbök mindig konvertálódnak elsőelemre mutatópointerré.`

Konstansok:

Van egy típusa, és egy értéke.
4 (int, 4) $\Rightarrow$ absztrakció(0,1 sorozat). De hány elemű? Nem definiált, implementációfüggő. Lényeg, hogy optimális legyen.

16 bites rendszer: Legnagyobb beírható int: $2^{15} - 1 = 32767$
`int x;`
`20000 + x;` //túlcsordul-e?
20000L: *long* típus kikényszerítése.
sizeof(short) $\leq$ sizeof(int) $\leq$ sizeof(long) $\leq$ sizeof(float) $\leq$ sizeof(double) $\leq$ sizeof(longdouble).

4. Előadás

Lebegőpontos konstans

1.4 - double
1.39L - longd
1.1 - float
$43e - 1 == 4,3 == 43 * 10^{-1})$

Karakter konstansok

'a', 'T', '\u'
Előjel nem definiált(signed char / unsigned char), ezért
`sizeof(T)== sizeof(signed T)== sizeof(unsigned T)`
FOLYTKÖV!!!

Operátorok

`+, /, ==, =, &&, ||, ++, --, +=, -=`

Szokásos aritmetikai konvenciók

`a * b`, ahol `*` egy bináris operátor, az operandusok különböző típusúak:

1. Ha az egyik long típusú, akkor a másik is longgá konvertálódik.
2. Különben ha az egyik double, akkor a másik is double-lé konvertálódik.
3. Különben, ha az egyik float, akkor a másik is float-tá konvertálódik.
4. Különben ha az egyik unsigned int, akkor a másik is unsigned int-té konvertálódik.
5.

5. Előadás

Kiértékelési sorrend

szekvenciapontok: `; &&, ||`

```
int i = 1;
int v[3];
v[i] = i++;
```

`v[1] = 1` vagy `v[2] = 1`

Élettartam, láthatóség

Láthatóság

a.cpp

```
int i;
static int j; //hasonló a const fogalomhoz.
extern int k; //
namsespace      //névtelen névtér
{
   int y;
}

namespace X
{
  int i; // x::i;
  void f()
  {
    int i = k;
    if(i < j)   //itt a j még nincs elfedve
    {
      int i = j;
    }
  }
  void g();
}

void f()
{
  int j= i;     //itt az i a globális i, hisz nem vagyunk az X-ben
  j = x::i;     //itt a j már az f fv.-ben belül deklarált j
  j = k;          //itt is
}

void X::g()
{
  ++i;    //X::i, mivel X-ben vagyunk
  ::i = ::k;
}
```

A `::` operátorral lehet hivatkozni a globális `i`-re. Ha nem fedem el, akkor tudok hivatkozni a globális változóra a nevükkel, mint pl az f függvényben a `k`-val hivatkozok az `extern int k`-ra.

`static`:

`extern`:

`extern int k;` → Másik fordítási egységben kell definiálni.

Élettartam

Hol, mikor jön létre a változó, mikor szűnik meg?
Hol: memórián belül.
Memória:

- Stack
- Statikus tárterület
- Heap

Mikor: Melyik az az időszak a program futásában, amikor a változó konstruktora lefut?
Változók:

- Automatikus változók
- Globális / névtérbeli / osztály statikus tag
- Lokális statikusok
- Dinamikus változók
- Temporálisak
- Osztály adattag
- Tömbelem

Automatikus változók: runtime stacken jönnek létre. Stack frame(aktivációs rekord)

```
void f()
{
  int i = 3; // auto int i = 3;
}
```

Névtérbeli / globális / osztály statikus tag: Statikusok a statikus tárterületen jönnek létre. Létrejönnek a program indulásakor, és megszünnek a program futásának végén.

```
class directory
{
  static const char separator = ':';
  ...
};
```

Lokális statikusok: Létrejön ha a vezérlés eléri a def-t., és megszűnik a program futásának végén. A statikus tárterületen létre.

```
void f()
{
  static int i;
}
```

Dinamikus változók: Heap memóriában jönnek létre. Létrejön a **`new`** kiértékelésénél, és megszűnik a **`delete`** kiértékelésénél.

```
void f()
{
  int k;
  std::cin >> k;
  int *v = new int[k];
  ...
  delete []v;
}
```

A következőkre érdemes figyelni

```
int *p = new int[k];
delete []p;
p = new int(k);
delete p;
delete p; //futási idejű hiba(nem lehet kétszer felszabadítani).
p = 0;
delete p;   //skippeli a NULL pointer törlését.
```

6. Előadás

Élettartam

Osztályok nem statikus tagja: Létrejön, amikor a tartalmazó objektumot létrehozzuk, megszűnik, amikor a tartalmazó objektum megszűnik.

```
Class student
{
  double avg; //nem statikus tag
}
```

Tömbelem: Létrejön a tartalmazó tömb létrejöttekor, és megszűnik a tartalmazó tömb megszűnésekor.

Temporális objektumok: Létrejönnek a részkifejezés kiértékelésekor, megszűnnek kifejezés kiértékelésének a végén.

```
std::string s,t,u;

std::string conc = s + t + u; //s+t és s+t+u is temporális.
```

Kódrészlet 1.

```
char * answer (char * q)
{
  std::cout << q;
  char ans[80];
  std::cin >> arg;
  return ans;
}
```

???

Kódrészlet 2.

```
char ans[80]
char *answer(char* q)
{
  std::cout << q;
  std::cin.getline(ans, 80);
  return ans;
}
```

Kódrészlet 3.

```
char *answer(char* q)
{
  std::cout << q;
  static char ans[80];
  std::cin.getline(ans, 80);
  return ans;
}
```

Kódrészlet 4.

```
char *answer(char* q)
{
  std::cout << q;
  char* ans = new char[80];
```

```
  std::cin.getline(ans,80);
  return ans;
}

char* p = answer("Hogy vagy?");
std::cout << p;
delete []p;
```

Kódrészlet 5.

```
std::string answer(char* q)
{
  std::cout << q;
  std::string answ;
  std::getline(std::cin, ans);
  return ans;
}
```

Deklarációk, definíciók

```
int fac(int); //deklaráció
int x; //definíció
extern int x; //deklaráció
class student; //deklaráció

//A student osztály definíciója:
class student{
  double avg;
  ...
}
```

Pointerek definíciói

```
int* p, q; //p pointer, q sima int.
int *s[10]; // 10 elemű int*-okat tartalmazó tömb.
int (*e)[10]; //pointer 10 elemű int tömbre.
double *a(int) //deklaráció.
double (*b)(int) //függvénypointer definíciója(olyan függvény ahol int
    paraméter, és double-t ad vissza.
```

7. Előadás

Alprogramok(függvények, eljárások)

```
int max(int a, int b)
{
  return a < b ? b : a;
}
std::cout << max (8, max(2,4));
```

Rekurzív függvények

n faktoriálisát számláló rekurzív függvény

```
int fac(int n)
```

```
{
  if(0 == n)
    return 1;
  else
    return n * fac(n - 1);
}
```

Paraméter átadás

```
void f(int x) //formális paraméter
{

}

f(4); //aktuális paraméter
```

Paraméter átadás fajtái:

- Érték szerinti
- Cím szerinti
- Eredmény illetve érték/eredmény szerinti
- Név szerinti

Érték szerinti paraméter átadás

Függvényhíváskor új lokális változó jön létre, belemásolódik az aktuális paraméter értéke.
Függvény irányába közvetít információt.
Másolás: copy konstruktor.

Érték szerinti (tipikusan C)

```
void f(int x)
{
  ++x;  //nem változik meg az aktuális paraméter.
}

int y = 6;
f(y);
f(4);
f(y / 7);
```

Tömb típusú paraméter átadása

```
int v[] = {3, 8, 2};
fv(v) //Ami valójában: fv(&v[0]), azaz az első elemre mutató pointerré
   konvertálódik
```

Cím szerinti paraméter átadás

`y` ténylegesen ugyanaz, mint az `x` a függvényen belül.
Mindkét irányba közvetít információt.
Nincs másolás, így gyorsabb.

Cím szerinti paraméter átadás (fiktív nyelvű kódrészlet)

```
void f(int x)
{
```

```
  ++x;
}

int y = 2;
f(y); //y értéke 3.
```

Eredmény szerinti paraméterátadás

A függvényhívás végén az `x` lokális változó értéke bemásolódik az aktuális paraméterbe. Hívó irányába közvetít információt.

Eredmény szerinti paraméter átadás

```
void f(int x)
{
  x = 0;
  ++x;
}

int y;
f(y);
```

Érték-eredmény szerinti paraméterátadás

2 másolás is történiik: `x`-be bemásolódik `y` értéke a függvény hívásakor, és a végén visszamásolódik. Mindkét irányba közvetít információt.

Érték/eredmény szerinti paraméter átadás

```
void f(int x)
{
  x = 0;
  ++x;
}

int y = 4;
f(y);
```

Név szerinti paraméterátadás

Név szerinti paraméter átadás

```
void f(int x)
{
  x = 0;
  ++x;
}

f(3); //Mintha érték szerint
f(y); //Mintha cím szerint
f(y + 2); //Mindig kiértékelődnek(??)
t[y] + k);
```

C/C++

C-ben érték szerinti paraméterek vannak.
C++ érték szerinti és cím szerinti.
Cím szerinti paraméter átadást a & operátorral jelöljük.

```
void f(int &p) //referencia álnév.
{
  ++p;
}
```

Pointerek
Memóriacím
Megváltozhat, hogy hova mutat.
Nem kötelező inicializálni.
NULL pointer.

Referenciák
Álnév
Mindig ugyanannak az álneve.
Kötelező inicializálni.
Nincs null ref.

```
int v[] = {3,2,8,6};
int i = 1;
int &p = v[i]; //int &p egy álnév.
v++;
++p;
const int &cr = v[i];
++cr; /nem lehetséges.
```

Ha nem cím szerint adjuk át, vagyis nem írjuk ki a & operátort a paraméterek elé, akkor érték szerint adjuk át, ezáltlal csak egy másolatot adunk a függvénynek, így az eredeti változókat nem módosítjuk.

Két szám cseréje cím szerinti átadással

```
void swap(int &a, int &b)
{
  int temp = a;
  a = b;
  b = temp;
}

int main()
{
  int x = 4, y = 6;
  swap(x,y);
  std::cout << x << ", " << y; // eredmény: 6, 4
}
```

Két szám cseréje érték szerinti átadással

```
void swap(int a, int b)
{
  int temp = a;
  a = b;
  b = temp;
}

int main()
{
```

```
  int x = 4, y = 6;
  swap(x,y);
  std::cout << x << ", " << y; // eredmény: 4, 6
}
```

9. Előadás

Az előadáson következő linkeken elérhető forrás- és fejállományokat mutatta az előadó: date.cpp, date.h, datemain.cpp

Forrás

- ELTE IK programtervezői informatikus szak 2013 tavaszi féléves Programozási nyelvek - CPP előadás alapján írt órai jegyzetem.
- Bjarne Stroustrup: A C++ programozási nyelv, 2001