Opakovanie

- (primitívne) Dátové typy:
  `int, double, char,`
- `boolean, String (v Jave)`
- Deklarácie premenných
- Priradenie
- Polymorfizmus

II. Základné elementy

5. Špeciality

- Skratky
- Operácie v postrannými efektami
- Typová konverzia (type casting)
- Skrátené vyhodnotenie logických výrazov

Skratky

- Pre premennú `i` typu int znamená
  - `i++;` a `++i;` skrátený zápis pre
    `i = i + 1;`
  - `i--;` a `--i;` skrátený zápis pre
    `i = i - 1;`

Skratky

- Pre aritmetický výraz `e` a premennú `x` (`int` / `double`) znamená
  - `x *= e;` skratku pre `x = x * e;`
  - `x /= e;` skratku pre `x = x / e;`
  - `x %= e;` skratku pre `x = x % e;`
  - `x += e;` skratku pre `x = x + e;`
  - `x -= e;` skratku pre `x = x - e;`

Postranné efekty

- Konštrukcie `i++`, `++i`, `i--` und `--i` sú výrazy typu `int`!

- `x = i++;` a `x = i--;`

  Premennej `x` je priradená hodnota `i` **pred** zvýšením, resp. znížením `i`.

- `x = ++i;` a `x = --i;`

  Premennej `x` je priradená hodnota `i` **po** zvýšení, resp. znížení `i`.

Príklady

```
r = n++ * n-- / 2;
```

má rovnaký efekt ako

```
r = n * (n + 1) / 2; // v Jave
r = n * n / 2;       // v C
```

```
r = n++ * n / 2;
```

má rovnaký efekt ako

```
r = n * (n + 1) / 2; // v Jave
r = n * n / 2;       // v C
n = n + 1; // aj v Jave aj v C
```

`r = n++ * --i / n;`

Vyhodnotenie v Jave cez elementárne operácie:

```
int h1 = n;
n = n + 1;
i = i - 1;
int h2 = i;
int h3 = h1 * h2;
int h4 = n;
int r = h3 / h4;
```

`n++`

`--i`

Pozor! Poradie vyhodnocovania (priradenie temporálnym premenným h1 a h2) je dôležité!

Ďalšie postranné efekty

- Je povolené (**ale nebudeme to (často:-) používať**) použiť priradenie priamo vo výraze!
- Príklad:

Toto **nie je** porovnanie!

```
int r2;
int r1 = ( r2 = n * (n + 1) ) / 2;
```

- `=` má najnižšiu prioritu z operátorov (okrem zátvorky) a vyhodnocovanie prebieha zľava doprava, preto `r2 = n * (n + 1) ;`

```
r1 = r2 / 2;
```

Skrátené vyhodnotenie `&&`

- Keď je prvý operand logického súčinu (`&&`) vyhodnotený ako `false`, vyhodnotenie druhého operandu sa nevykoná a výsledok je `false`.
- Príklad (s `int n = 0`):

```
boolean b = (n != 0) && (5 / n >= 1)
```

Skrátené vyhodnotenie `||`

- Keď je prvý operand logického súčtu (`||`) **`true`**, vyhodnotenie druhého operandu sa nevykoná a výsledok je **`true`**.
- Príklad (s **`int n = 0`**):

```
boolean b = (n = = 0) || (5 / n >= 1)
```

Explicitná typová konverzia

- Unárne operátory (`int`),(`float`) a (`double`) zmenia typ výrazu (type casting).
- Pozor:
  - Možná strata presnosti
  - Pri prekročení rozsahu daného typu (pri pretečení) žiadne chybové hlásenie v Jave a v C; čo môže viesť k zlému výsledku!

6. Príkazy

- Elementárne príkazy
- Postupnosti príkazov a bloky
- Podmienené príkazy
- Iteračné príkazy - cykly

6.1 Elementárne príkazy

- Deklarácie
- Priradenia

EBNF: Elementárne príkazy

```
<Príkaz> =          <Deklarácia>      |
                    <Priradenie>      |
                    ...               // (neskôr viac)

<Deklarácia> =      ...               // (pozri stranu 16, Fólie3)

<Priradenie> =      <Identifikátor> "=„ <Výraz> ";" |
                    ...               // (neskôr viac)
```

6.2 Bloky príkazov

- Programy sú v podstate postupnosti príkazov:

```
<PostupnosťPríkazov> =
    <Príkaz>  |
    <PostupnosťPríkazov> <Príkaz>
```

- Príkazy je možné spájať do blokov

Príklad: Blok

```
...
int r;
{  int h1 = n;
   n = n + 1;
   i = i - 1;
   int h2 = i;
   int h3 = h1 * h2;
   int h4 = n;
   r = h3 / h4;
}
System.out.println("r = " + r);
```

Blok je v Jave aj v C označený zloženými zátvorkami;

Premenná, ktorá je deklarovaná v bloku, môže byť použitá iba v danom bloku, je platná iba v rámci bloku. Pred a za blokom je nedeklarovaná a teda neznáma.

Príklad: Blok

```
int i = 1;
{ int i = 0;
   ...
}
```

Premenná môže byť v bloku deklarovaná iba jedenkrát v bloku. V jazyku C môže byť deklarová v bloku aj v podbloku, pričom premenná v podbloku prekrýva premennú v nadbloku. V Jave nie je povolené deklarovať premennú v nadbloku aj podbloku.

EBNF: Blok

```
<Príkaz> =...   |          // (ako doteraz)
                <Blok>     |
                ...              // (neskôr viac)
```

```
<Blok> = "{" [ <PostupnosťPríkazov> ] "}"
```

6.3 Riadiace štruktúry

6.3.1 Podmienené príkazy

Príklad 1:

Náš program **`Suma`** počíta pre prirodzené čísla zmysluplný výsledok!
Ak je zadané záporné číslo, mohol by program užívateľa informovať, že počíta sumu iba pre nezáporné čísla.

Príklad v Jave

```
public class Suma2
{
  public static void main (String[] args)
  {
    int n = Integer.parseInt(args[0]);
     if (n >= 0)
     {
         int x = n * ( n + 1) / 2;
         System.out.println(„Suma: " + x);
     }
     else
     System.out.println(“Prosim ziadne zaporne cisla!“);
  }
}
```

Príklad v C

```
#include <stdio.h>
  int main()
  {
    int n;
    printf("Zadajte cele cislo: ");
    scanf("%d", &n);
   if (n >= 0)
   {
    int x = n * ( n + 1) / 2;
     printf("Suma: %d", x);
   }
   else
   printf("Prosim ziadne zaporne cisla! \n");

   return 0;
}
```

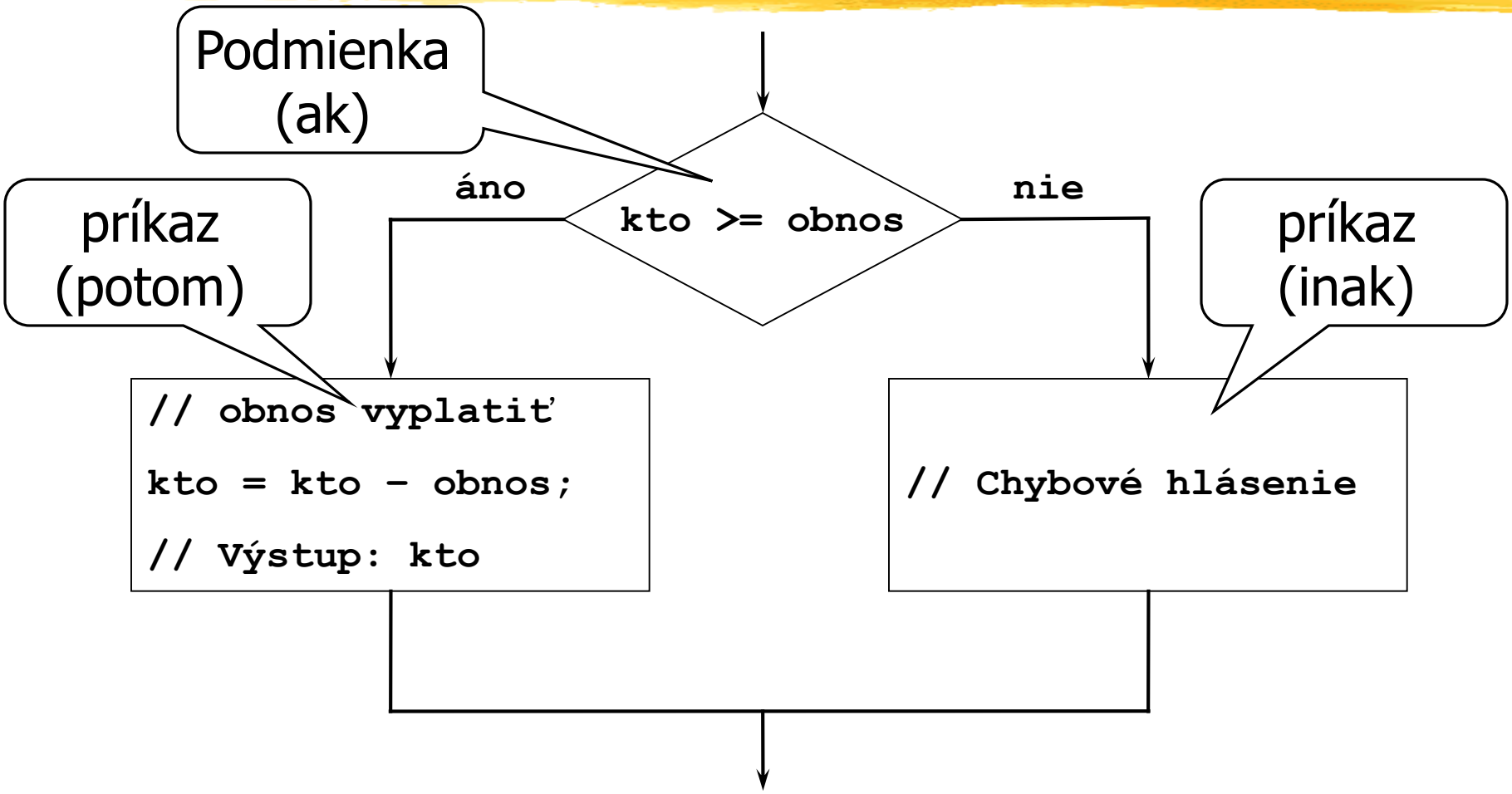
Príklad 2

Príklad 2:

Zákazník chce vybrať zo svojho konta so stavom konta uloženom v premennej `kto` hotovosť `obnos`.

Banka nepovoľuje možnosť ísť do mínusu!

Príklad ako vývojový diagram



Príklad v Jave

```
...

if ( kto >= obnos )
{ // vyplatit(obnos); (Vyplatenie nie je naprogr.)
  kto = kto - obnos;
  System.out.println("Vas stav konta je: " + kto);
}
else
{ System.out.println("Na konte nie je dostatok
  penazi.");
}

...
```

Príklad 2 (Pokračovanie)

Príklad 2 (pokračovanie):

Zákazník chce aby mu bol vyplatený aspoň zvyšok, ktorý sa na konte nachádza.

Príklad v Jave

```
if ( kto >= obnos )
{ // vyplatit(obnos); (Vyplatenie nie je naprogr.)
  kto = kto - obnos;
  System.out.println("Vas stav konta je: " + kto);
}
else if (kto > 0)
{ // vyplatit(kto);
  kto = 0;
  System.out.println("Vas stav konta je: " + kto);
}
else
{ System.out.println("Na konte nie je dostatok
  penazi."); }
```

EBNF: Podmienený príkaz

```
<Príkaz> =      ...                          |
                <PodmienenyPrikaz>           |
                ...
```

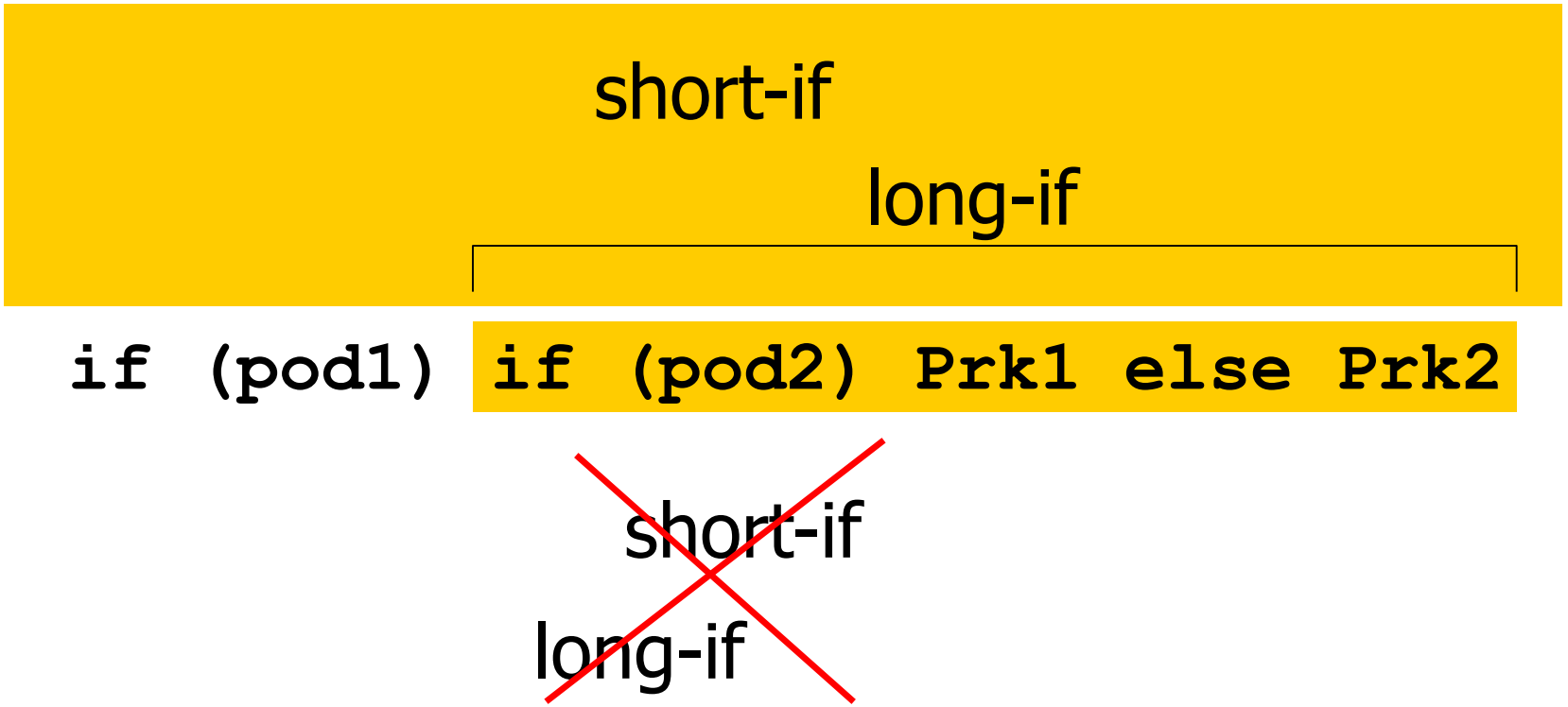
Short-If: V inak-prípade je Príkaz ignorovaný!

```
<PodmienenyPrikaz> =
   "if" "(" <Vyraz> ")" <Prikaz>                              |
   "if" "(" <Vyraz> ")" <Prikaz> "else" <Prikaz>              |
```

Vyraz musí byť typu **boolean**!

Long-If: V inak-prípade bude vykonaný druhý príkaz!

Problém: Jednoznačnosť



Problém: Jednoznačnosť

- „Naša“ EBNF pre podmienku neposkytuje vždy jednoznačné riešenie!
- Skutočná EBNF (pozri napr. dokumentáciu Javy), poskytuje vždy jednoznačné riešenie; táto EBNF je ale pomerne komplikovaná a ťažko čitateľná!
- Namiesto toho:

Pravidlo:

V dlhom If-príkaze (long-if) nikdy nie je ako prvý príkaz vnorený krátky If-príkaz (short-if)!

Lepšie: Zamedziť nedorozumeniu používaním zátvoriek (blokov):

```
if (pod1) { if (pod2) Pr1 else Pr2 }
if (pod1) { if (pod2) Pr1 } else Pr2
```

6.3 Riadiace štruktúry (pokr.)

6.3.2 Iteračné príkazy (Cykly)

Príklad 1:

Chceme násobiť postupne všetky čísla od 1 po `n`, t.j. vypočítať faktoriál čísla `n`:

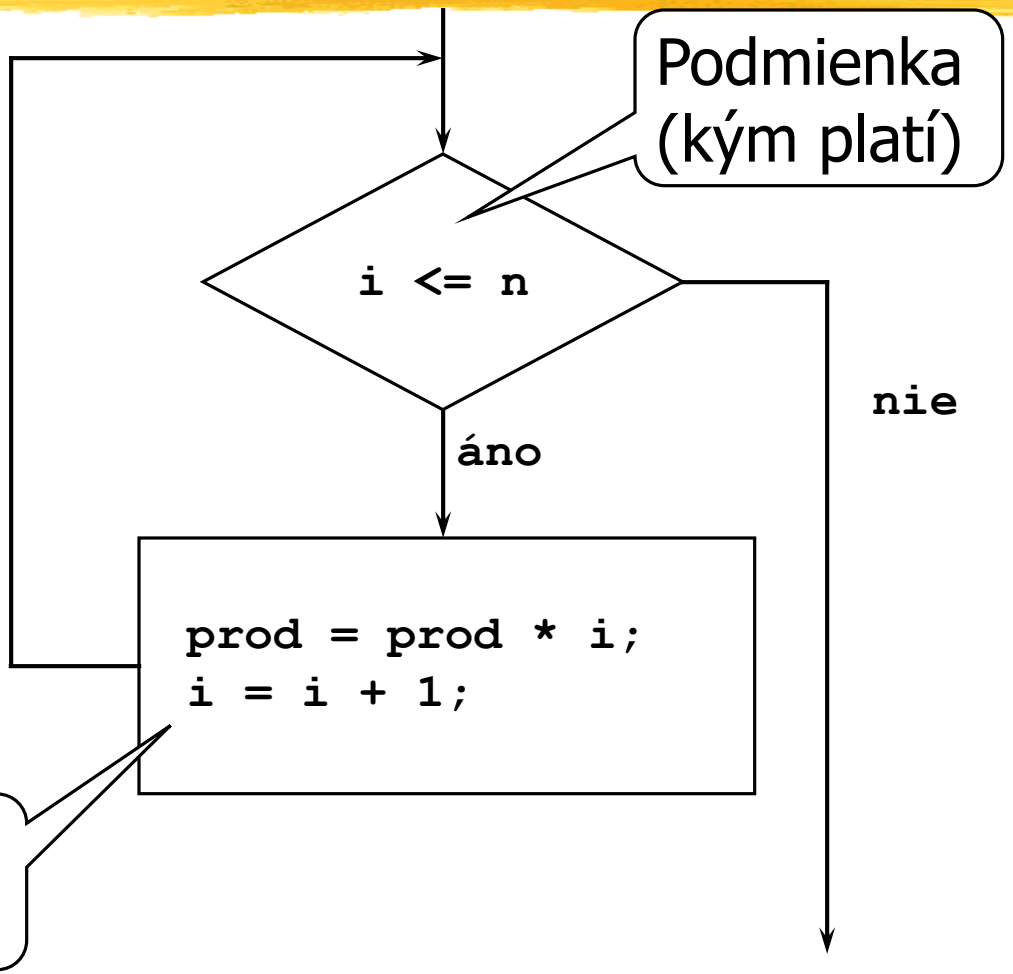
`n! = 1 * 2 * 3 * ... * n =` $\prod_{i=1}^{n} i$

Príklad

```
...

 int prod = 1;
  int i = 1;
  while (i <= n)
  { prod = prod * i;
    i = i + 1;
  }



...
```

Príklad

```
...

 int prod = 1;
  int i = 1;
 while (i <= n)
 { prod = prod * i;
   i = i + 1;
 }

...
```



Príklad v Jave

```
public class Faktorial
{
  public static void main (String[] args)
  {
     int n = Integer.parseInt(args[0]);
     int prod = 1;
     int i = 1;
     while (i <= n)
     { prod = prod * i;
          i = i + 1;
     }
   System.out.println("Faktorial cisla "+n+" je "+prod);
   }
}
```

Príklad v C

```
#include <stdio.h>
  int main()
   { int n;
   printf("Zadaj n \n");
   scanf("%d", &n)
   int prod = 1;
   int i = 1;
  while (i <= n)
  { prod = prod * i;
    i = i + 1;
  }
  printf("Faktorial cisla %d je %d", n, prod);
}
```

EBNF: Iteračné príkazy

```
<Príkaz> =      ...                          |
                <IteračnýPríkaz>             |
                ...

<IteračnýPríkaz> =
  "while" "(" <Výraz> ")" <Príkaz>
```

Výraz musí byť typu **boolean**!

Problém: Ukončenie cyklu Terminácia

Kto garantuje, že podmienka iteračného cyklu sa stane niekedy nepravdivá (false) a tým sa ukončí cyklus?

Programátor, t.j. Vy!!!

Problém: Ukončenie cyklu Terminácia

Sú formy iteračných príkazov – cyklov, ktoré vždy terminujú!

(viac na ďalších fóliách, predtým ešte jeden príklad)

Príklad: Výpočet odmocniny

Výpočet odmocniny $\sqrt{x}$ pomocou delenia intervalu!

Idea:

- Zvolíme Intervall [d, h], v ktorom sa $\sqrt{x}$ určite nachádza.
- Rozdelíme interval na dva intervaly [d,s] und [s,h];
- Zvolíme interval, v ktorom sa $\sqrt{x}$určite nachádza a opakujeme procedúru, kým nedosiahneme požadovanú presnosť, t.j. kým interval nie je dostatočne malý.

Príklad: Výpočet odmocniny

```
// Vstup:  double x;  x >= 0
double d = 0;          // Spodná hranica počiatočného intervalu
double h = x + 1;      // Horná hranica počiatočného intervalu
double eps = 1.0e-10; // želaná relatívna presnosť
double s = (d + h) / 2;

while ( (h - d) / s > eps)
{if (s * s >= x)
    h = s;
  else
    d = s;
  s = (d + h) / 2;
}
// Výstup: s
```

Výpočet odmocniny v C

```
#include <stdio.h>
int main () {
   double x;
   printf("Zadajte x \n");
   scanf("%lf", &x);
   double d = 0;
   double h = x + 1;
   double s = (d + h) / 2;
   double eps = 1.0e-10;

while ((h-d) / s > eps)
{  if (s * s >= x)
         h = s;
   else
         d = s;
   s = (d + h) / 2;
}
printf("Odmocnina %lf je %lf", x, s);
return 0;
}
```