Procedurálne programovanie

slido.com
1412341
PrPr – P11

Ján Zelenka

Ústav Informatiky

Slovenská akadémia vied

Oznamy

slido.com
1412341
PrPr – P11

Termín odovzdania druhého projektu (Spájaný zoznam štruktúr): odovzdanie v 11. týždni (3.12. do 23:59)

- neskoré odovzdanie 12. týždeň (10.12.do 23:59), penalizácia - uznáva sa 80% zo získaného počtu bodov
- za projekt musí získať študent **min. 6 bodov** (akceptovateľný),bez bodov za prezentáciu (2 bodov)

IDstudenta_Rok_projekt_2.c

Obsah prednášky

slido slido.com
1412341
PrPr – P11

1.Oddelený preklad

Spätná väzba: *https://forms.gle/6q5D2G6UwrtimXEx9*

Oddelený preklad

slido.com
1412341
PrPr – P11

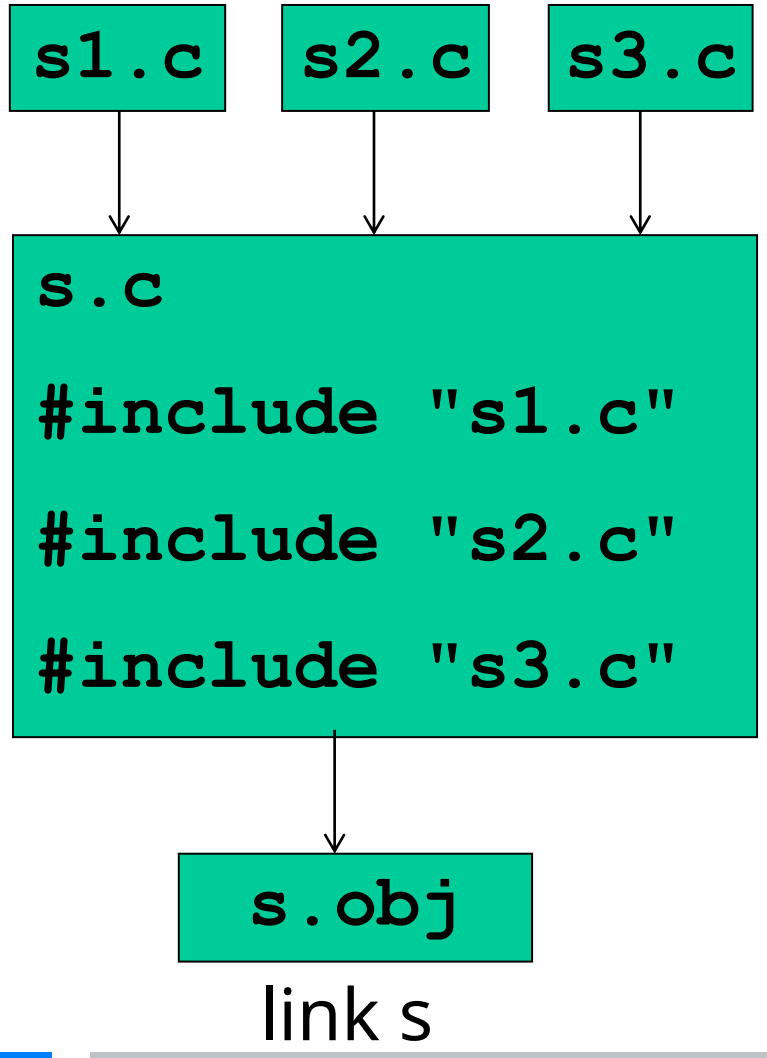
Oddelený preklad

slido slido.com
1412341
PrPr – P11

- vkladanie súborov
  - `#include` → vznikne jeden .OBJ súbor
    - Vkladá sa jeden, alebo viac súborov
      - Nevýhody:
        - pri zmene v ľubovoľnom súbore musíme zbytočne prekladať aj všetky ostatné
        - medzi jednotlivými súbormi nie je možné skryť ich globálne identifikátory
- oddelený preklad
  - každý súbor sa preloží zvlášť (vznikne viac .OBJ súborov) a potom sa spoja pomocou linkeru
  - možnosť rozdelenia problému na viacero menších nezávislých častí, na ktorých pracuje súčasne viacero programátorov

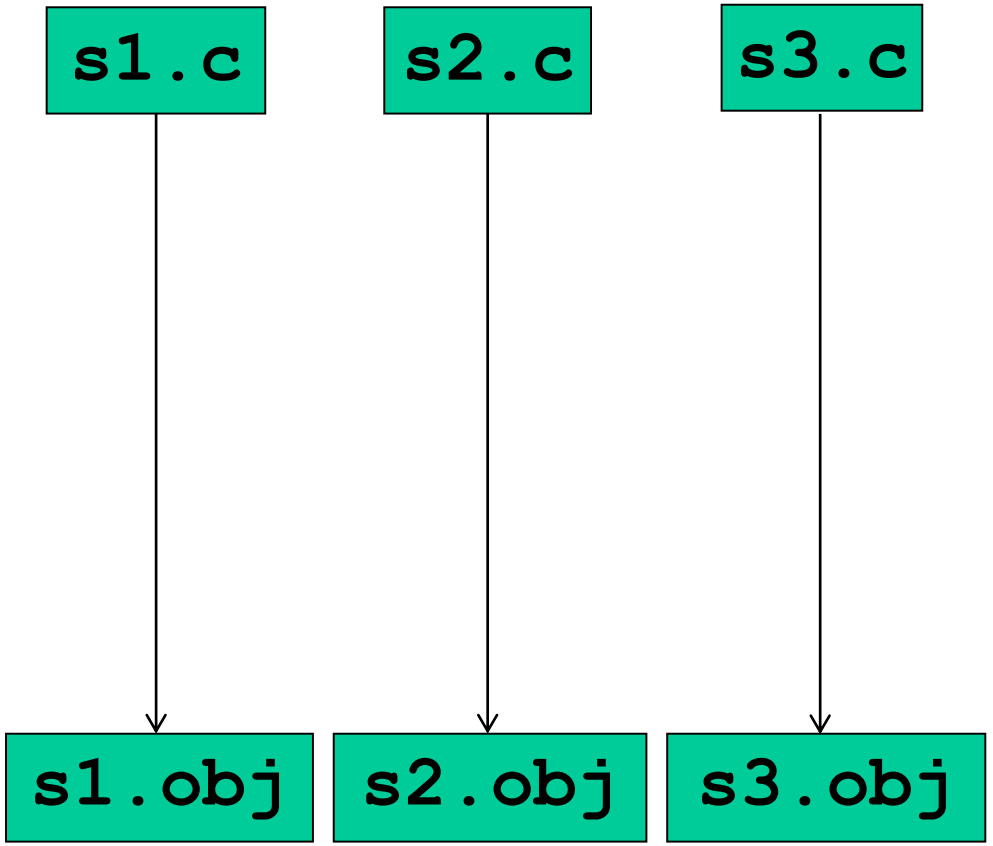
Príklad oddeleného prekladu

vkladanie súborov:



link s

oddelený preklad:



link s1, s2, s3

Oblasť platnosti identifikátorov

slido slido.com
1412341
PrPr – P11

Jazyk C umožňuje rôzne spôsoby ako stanoviť kde, kedy a aký identifikátor bude komu dostupný:

- globálne a lokálne premenné
- pamäťové triedy
- typové modifikátory

Globálne premenné

slido slido.com
1412341
PrPr – P11

organizácia v programe:

definície funkcií

globálne definície a deklarácie

globálne definície: definujú premenné, rozsah ich platnosti: od miesta definície po koniec súboru (nie programu - program sa môže skladať z viac súborov)

globálne deklarácie: deklarácie premenných, ktoré sú definované v inom súbore. často sú špecifikované slovom **`extern`** - externé deklarácie

Globálne definície

slido slido.com
1412341
PrPr – P11

```
int i;

void prva()
{...}

int j;

int druha()
{...}

char *tretia()
{...}
```

premenná `i` je platná pre všetky 3 funkcie

premenná `j` je platná len pre funkcie: `druha()` a `tretia()`

Lokálne premenné

- definované vo funkciách
- platnosť lokálnych premenných: od definície po koniec funkcie

```
int i1, i2;

void prva() {
int i1, j1;
...}

int j1, j2;

int druha(){
int i1, j1, k1;
...}
```

globálna premenná **i1** je prekrytá lokálnou premennou **i1** (používať sa môžu premenné: **i1**, **j1** (lokálne) a **i2** (globálna))

dve globálne premenné: **i2**, **j2** a tri lokálne premenné: **i1**, **j1**, **k1**.

Inicializácia lokálnych a globálnych premenných

slido slido.com
1412341
PrPr – P11

- lokálne premenné:
  - nie sú automaticky inicializované
- globálne premenné:
  - automaticky inicializované na 0 (0.0, \0)

    (lepšie - nespoliehať sa na to)

Pamäťové triedy

slido slido.com
1412341
PrPr – P11

Určujú v ktorej časti pamäte bude premenná umiestnená a kde bude viditeľná

- `auto`
- `extern`
- `static`
- `register`

Trieda auto

slido slido.com
1412341
PrPr – P11

Automatické premenné:

- implicitne pre lokálne premenné
- uložená v zásobníku (*stacku*)
- existuje od vstupu do funkcie do jej konca
- nie je automaticky inicializovaná (obsahuje náhodnú hodnotu)

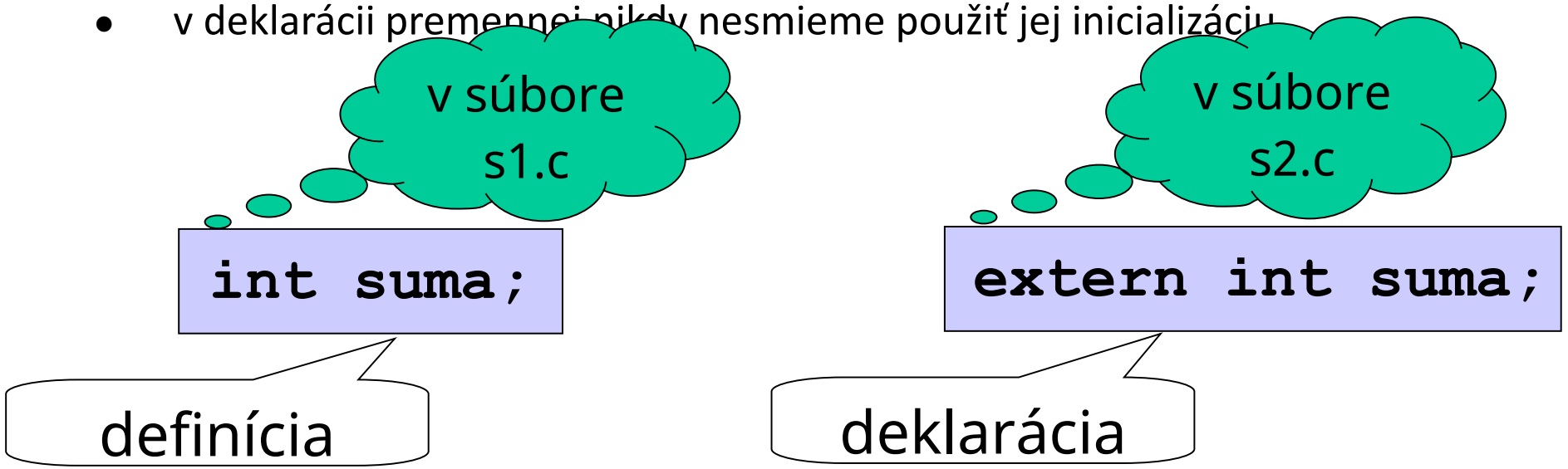
```
void func()
{
    auto int i;
    auto int j = 5;
}
```

je ekvivalentné

```
void func()
{
    int i;
    int j = 5;
}
```

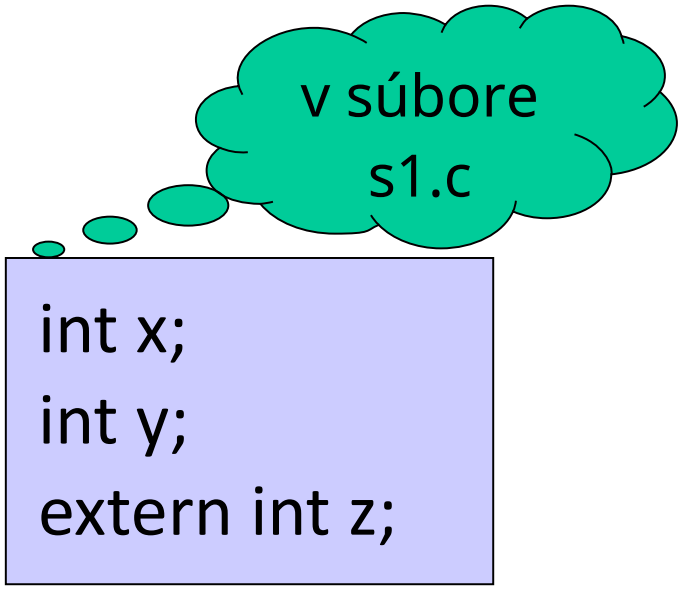
Trieda `extern`

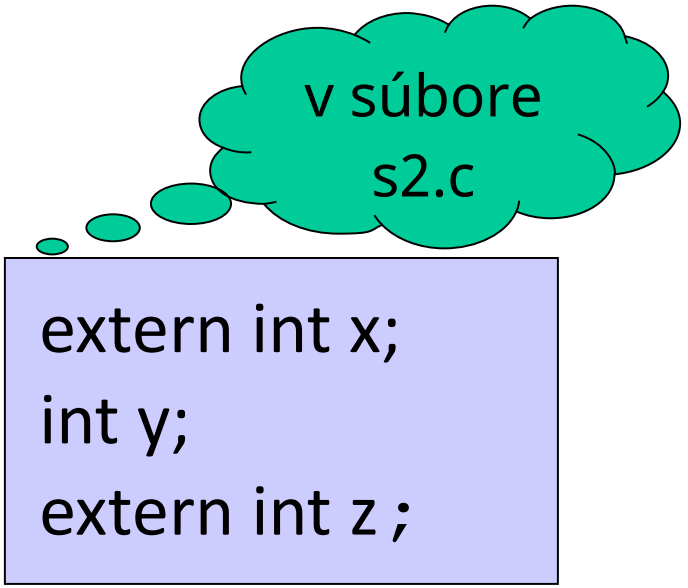
- implicitne pre globálne premenné
- uložená v dátovej oblasti
- používa sa pri oddelenom preklade súborov, kde viac súborov zdieľa jednu premennú
  - v jednom súbore je táto premenná definovaná bez kľúčového slovo **`extern`**, v ostatných súboroch musí byť použité **`extern`**
  - v deklarácii premennej nikdy nesmieme použiť jej inicializáciu



Trieda `extern`

slido slido.com
1412341
PrPr – P11





- iba premenná **x** je zdieľaná správne.
- premenná **y** je definovaná v oboch prekladaných súboroch a pri spojení oboch modulov linkerom by došlo ku kolízii.
- premenná **z** je v oboch súboroch iba deklarovaná, vo skutočnosti fyzicky neexistuje a nedá sa teda použiť.

Trieda `static`

1. spôsob použitia

slido slido.com
1412341
PrPr – P11

- neexistuje žiadna implicitná definícia
- premenné tejto triedy sú uložené v dátovej oblasti
- najčastejšie lokálne premenné (definované vo funkcii), ktoré si nechávajú svoju hodnotu medzi jednotlivými volaniami funkcie
- premenná existuje od prvého volania funkcie do konca programu

```
int f() {
    int x = 2;
    static int i = 0;
    printf("i: %d, x: %d \n", i, x);
    i++; x++;
}
```

```
for(j = 0; j < 3; j++) f();
```

```
i: 0, x: 2
i: 1, x: 2
i: 2, x: 2
```

Trieda `static`

2. spôsob použitia

slido slido.com
1412341
PrPr – P11

- globálne premenné: tiež **`static`** - viditeľné len v module, kde sú definované
  - takto označené premenné a funkcie sú pri oddelenom preklade dostupné iba vo vlastnom súbore
  - programátori sa nemusia obávať kolízie, ak použijú rovnaké názvy premenných, alebo funkcií v spoločnom programe
- ak viac **`static`** premenných, radšej každú na zvlášť riadku

```
static int i, j;
```



```
static int i;
static int j;
```

Trieda `static`

2. spôsob použitia

slido slido.com
1412341
PrPr – P11

v súbore s1.c

```
int x;
static int y;
extern int z;
```

v súbore s2.c

```
extern int x;
static int y;
extern int z ;
```

- iba premenná **x** je zdieľaná správne.
- premenná **y** je definovaná v oboch prekladaných súboroch a pri spojení oboch modulov linkerom si navzájom neprekážajú (zmena hodnoty y v jednom module neovplyvni hodnotu y v druhom module)
- premenná **z** je v oboch súboroch iba deklarovaná, vo skutočnosti fyzicky neexistuje a nedá sa teda použiť.

Trieda `register`

- jazyk C - nízkoúrovňový a preto umožňuje, aby premenná bola zapísaná v *registri*
  - ak sa dá
  - do ktorého registra - vyberá si prekladač
  - len lokálne premenné
  - tie, ku ktorým je vhodné pristupovať rýchlo
  - nemôžeme získať adresu registrovej premenej
  - globálna premenná typu register neexistuje!

```
register int i;
```

```
register int i;
register int j;
```

ak je viac premenných, označenie triedy **`register`** vyznačiť pre každú premennú zvlášť

Typové modifikátory

Ľubovoľná premenná nejakého *dátového typu* (`int i`) zaradená do nejakej *pamäťovej triedy* (napr. `static`) môže byť modifikovaná *typovým modifikátorom*:

- `const`
- `volatile`

napríklad:

```
static const unsigned int j = 5;
```

Modifikátor `const`

- po inicializácii už nemôže byť menená hodnota premennej
- podobne ako konštanta (**`#define`**), len má určený typ, čo konštanta nemá (dá sa využiť ku kontrolám prekladača (napr. typ skutočných parametrov funkcie))
- najčastejšie: pri definícii formálnych parametrov funkcie - parameter označený ako **`const`** je len vstupným parametrom (vo funkcii sa nemení)

napríklad:

```
int hladaj(const char *str, char co)
```

Modifikátor `const`

Časté chyby:

```
const float pi = 3.14159;
const int max = 100;

int pole[max];

pi = 3.14;
```

chyba: **`max`** nie je konštanta

chyba: **`pi`** sa už nedá meniť

Modifikátor `volatile`

slido slido.com
1412341
PrPr – P11

- zriedkavo použitý
- upozorňuje prekladač, že premenná môže byť modifikovaná nejakou bližšie nešpecifikovanou asynchrónnou udalosťou (napr. pomocou prerušenia)
  - napr. cyklus (neobsahujúci premennú **`pocet`**), ktorý je závislý na premennej **`pocet`** a tá je menená hardverovým prerušením
  - ak by nebola označená modifikátorom **`volatile`** - prekladač by to mohol optimalizovať tak, že by premennú **`pocet`** nahradil jej hodnotou, akú mala pred cyklom

Bloky

Blok programu - uzatvorený v `{ }` môže obsahovať definície premenných

- lokálne premenné (**`auto`**, alebo **`static`**)
- viditeľné sú len v bloku

```
if (i > 0) {
     int i;
     ...
}
else {
     double f;
     ...
}
```

Oblasť platnosti identifikátorov: zhrnutie

globálne a lokálne premenné

Pamäťové triedy:

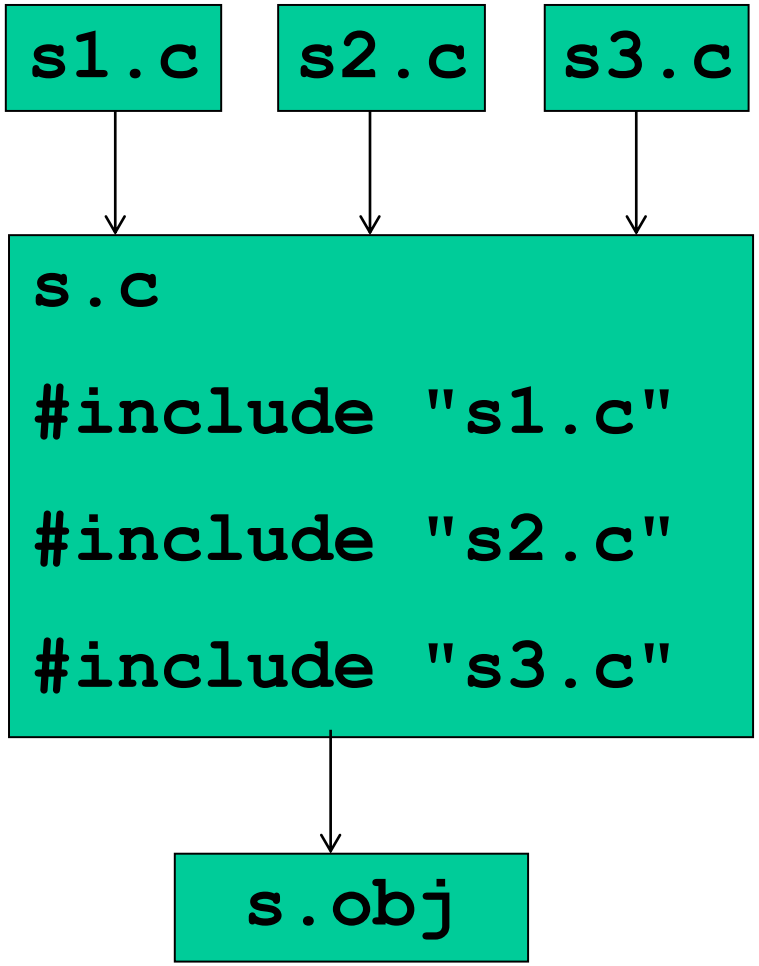
- **`auto`** (implicitne lokálne premenné, v stacku)
- **`extern`** (implicitne globálne premenné, v dátovej oblasti)
- **`static`** (lokálne premenné: zachovávajú hodnotu medzi volaniami funkie, globálne premenné: platnosť v rámci viacerých súborov)
- **`register`** (lokálne premenné, zapísané v registri)

Typové modifikátory:

- **`const`** (konštantné premenné, nemenia hodnotu)
- **`volatile`** (lokálne premenné, zmena hardverovým prerušením)

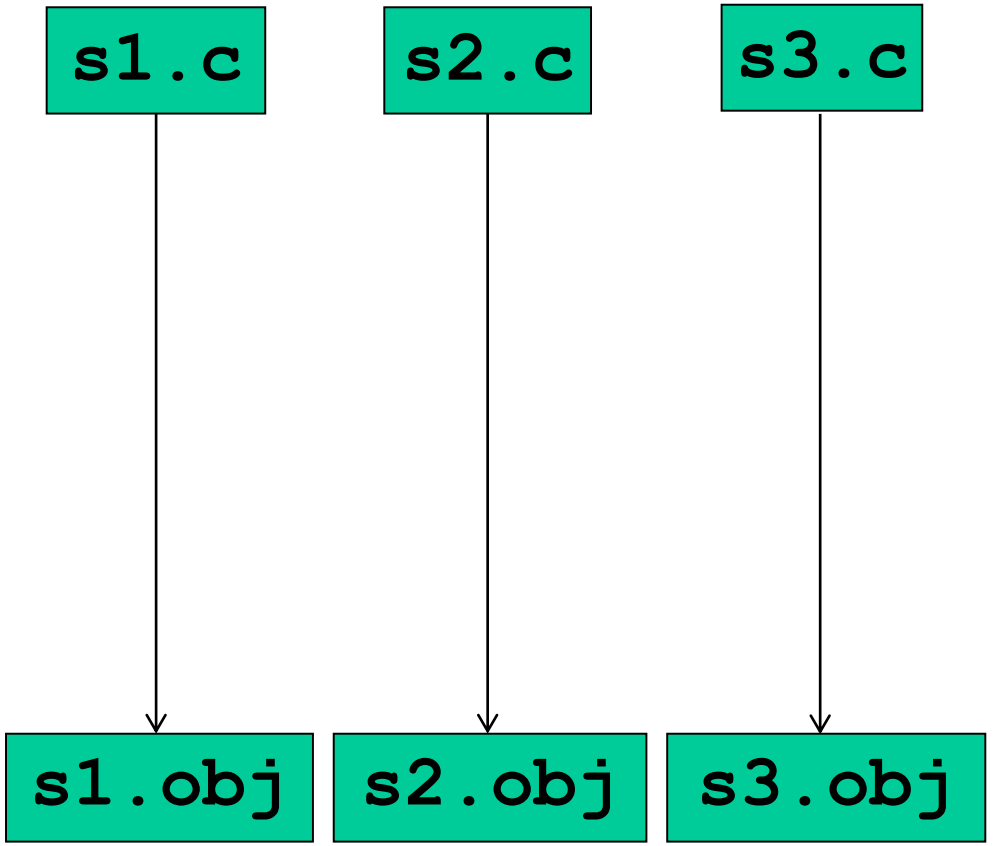
Oddelený preklad súborov

vkladanie súborov:



link s

oddelený preklad:



link s1, s2, s3

Oddelený preklad súborov

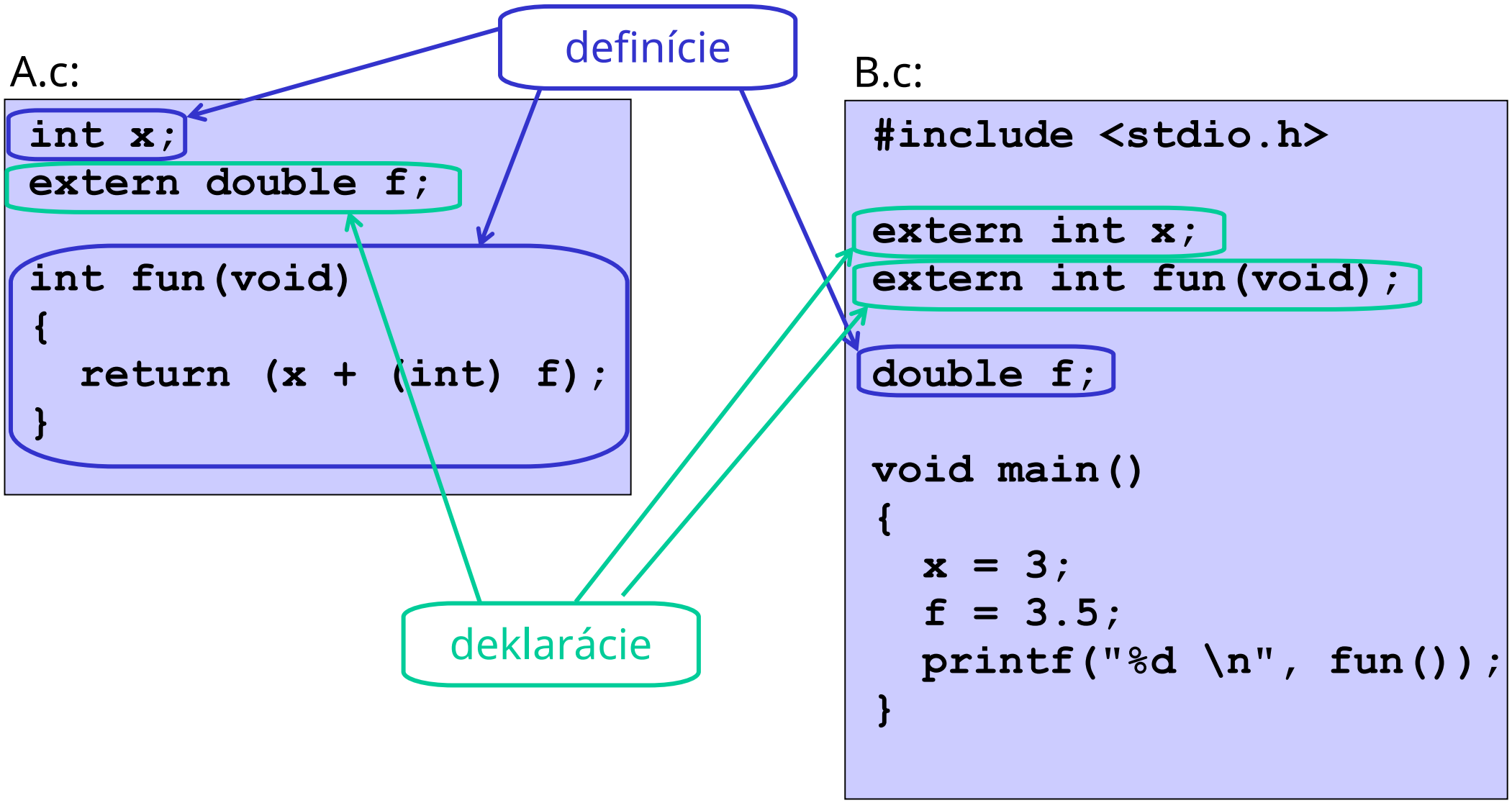
- rozdeliť program na viac čo najmenej závislých častí (**.c** súbory)
- definovať rozhranie - spôsob komunikácie medzi oddelenými časťami (**.h** súbory)
  - pomocou funkcií (lepšie ako pomocou globálnych premenných viditeľných vo všetkých súboroch)

Rozšírenie platnosti globálnej premennej

slido slido.com
1412341
PrPr – P11

- globálna premenná:
  - definovaná v jednom súbore
  - deklarovaná pomocou **`extern`** v ostatných súboroch

Príklad: rozšírenie platnosti globálnej premennej



Statické globálne premenné a funkcie

slido slido.com
1412341
PrPr – P11

Trieda `static` - viditeľné len v module, v ktorom boli definované

- pre globálne premenné
- pre funkcie

- Výhodné označenie, ak na programe pracuje viac programátorov – nemusia sa obávať, že zvolia rovnaké mená pre identifikátory

Príklad: statické globálne premenné a funkcie

slido slido.com
1412341
PrPr – P11

A.c:

```
static int x;
extern double f;

static int fun(void)
{
  return x;
}
static int funkcia(void)
{
  return (x + (int) f);
}
```

pri linkovaní **chyba**:
x: v A.c static
v B.c len deklarácia

B.c:

```
#include <stdio.h>

extern int x;
extern int funkcia(void);
double f;

static void fun(void) {
  printf("%d \n", funkcia());
}

void main() {
    x = 3;
    f = 3.5;
    printf("%d \n", fun());
}
```

Zdrojové a hlavičkové súbory

slido slido.com
1412341
PrPr – P11

.c súbory - definície globálnych premenných a funkcií:

- pre všetky moduly (žiadne označenie)
- len pre aktuálny modul (označenie **`static`**)

.h súbory - definujú rozhranie, t.j. definujú globálne premenné a funkcie, ktoré sa používajú v iných moduloch (označenie **`extern`**)

.c súbory include-ujú len **.h** súbory ktoré potrebujú (nikdy ne-include-ujú **.c** súbory)

Ako udržať poriadok vo veľkom programe

1. definície funkcií a premenných v súbore *program_1.c*
2. v *program_1.c* - striktne rozlíšené, čo sa bude používať v rámci súboru a čo mimo neho (čo najmenej)
3. všetky ostatné globálne premenné a funkcie označiť **`static`**
4. funkčné prototypy zdielaných funkcií a premenných - do *program_1.h*, označíme ich **`extern`**
5. ak poskytujeme aj symbolické konštanty - dáme ich len do *program_1.h*
6. *program_1.c* začína:

```
#include <stdio.h>
#include "program_1.h"
```

Ako udržať poriadok vo veľkom programe

7. súbor *program_2.c* obsahujúci ďalší modul programu a využíva premenné, funkcie alebo konštanty z modulu *program_1*, začína:

zdielané funkcie a premenné
deklarácia vlastného súboru

```
#include <stdio.h>
#include "program_1.h"
#include "program_2.h"
```

Odporučený obsah .c súborov

1. dokumentačná časť

   (meno súboru, verzie, stručný popis modulu, meno autora a dátum)

2. všetky potrebné **`#include`**

   len súbory **.h**, nie **.c**!

   najprv systémové **< >** a potom vlastné **" "**

3. deklarácie importovaných objektov

   len ak nie sú v **.h** súbore spolupracujúceho modulu (čomu dávame prednosť) - v iných moduloch sú tieto objetky bez špecifikovanej triedy

4. definície globálnych premenných

   premenné definované mimo funkcií zdielané inými modulmi (čo najmenej!)

Odporučený obsah .c súborov

slido slido.com
1412341
PrPr – P11

5. lokálne **`#define`**

   definícia konštánt a makier len pre aktuálny modul

   len ak veľmi rozsiahle - do zvlášť **.h**

6. definície lokálnych typov

   `typedef` len pre aktuálny modul

   len ak veľmi rozsiahle - do zvlášť **.h**

7. definície premenných len pre aktuálny modul

   globálne len pre aktuálny modul - **`static`**

8. úplné funkčné prototypy funkcií len pre aktuálny modul
9. funkcia **`main()`**
10. definície funkcií aj pre iné moduly
11. definície funkcií len pre aktuálny modul

Odporučený obsah .h súborov

1. dokumentačná časť
   (meno súboru, verzie, stručný popis modulu, meno autora a dátum)
2. definície symbolických konštánt ktoré sa budú používať aj v iných moduloch
3. definície makier s parametrami, ktoré sa budú používať aj v iných moduloch
4. definície typov, ktoré sa budú používať aj v iných moduloch
5. deklarácie premenných aktuálneho modulu, ktoré sa budú používať v iných moduloch (tu označené **`extern`**)
6. funkčné prototypy funkcií aktuálneho modulu, ktoré sa budú používať v iných moduloch (tu označené **`extern`**)

Príklad: oddelený preklad

- program na výpočet obsahu a obvodu kruhu:

  `kruh_main.c, kruh_main.h, kruh_io.c, kruh_io.h`

	globálne premenné a funkcie	lokálne premenné a funkcie

`kruh_main`	`double obvod`
	`double vyp_priemer(double polomer)`
		`double polomer, obsah`
		`void vyp_obsah(double polomer)`
		`doble vyp_obvod()`
		`void vypocet(void)`

`kruh_io`	`double vstup_dat()` `double vystup_dat(double obsah)`	`double polomer`

premenné a funkcie definované v jednom z modulov - môžu sa používať aj v rámci druhého modulu

```c
#include <stdio.h>
#include "kruh_main.h"
#include "kruh_io.h"

double obvod;
#define pi 3.14;

static double polomer;
static double obsah;

double vyp_priemer(double
                   polomer);
static void vyp_obsah
            (double polomer);
static doble vyp_obvod();
static void vypocet(void);

void main() {
  polomer = vstup_dat();
  if (polomer == CHYBA_DAT)
    printf("Chybny polomer.");
  else {
    vypocet();
    vystup_dat(obsah);
  }
}
```

```c
static void vyp_obsah(double
                      polomer)
{
  obsah = PI * polomer *polomer;
}

static doble vyp_obvod()
{
  return(PI*vyp_priemer(polomer));
}

static void vypocet(void)
{
  obvod = vyp_obvod();
  vyp_obsah(polomer);
}

double vyp_priemer(double polomer)
{
  return (2 * polomer);
}
```

glob. definície

kruh_main.c

Hlavičkový súbor modulu kruh_main

deklarácie premenných a funkcií aktuálneho modulu, ktoré sa budú používať v iných moduloch (tu označené **extern**)

```
extern double obvod;

extern double vyp_priemer(double polomer);
```

kruh_main.h

```c
#include <stdio.h>
#include "kruh_io.h"
#include "kruh_main.h"

#define kontrola(x) ((x >= 0.0) ? x : CHYBA_DAT)

static double polomer;

double vstup_dat(void) {
    printf("Zadaj polomer kruznice: ");
    scanf("%lf", &polomer);
    return kontrola(polomer);
}

void vystup_dat(double obsah) {
    double priemer;

    printf("Obsah kruhu s polomerom %6.2f je %.2f\n",
        polomer, obsah);
    priemer = vyp_priemer(polomer);
    printf("Obvod kruhu s polomerom %6.2f je %.2f\n",
        polomer, obvod);
}
```

glob. definície

kruh_io.c

Hlavičkový súbor modulu kruh_io

deklarácie premenných a funkcií aktuálneho modulu, ktoré sa budú používať v iných moduloch (tu označené **extern**)

```
#define CHYBA_DAT -1

extern double vstup_dat();

extern double vystup_dat(double obsah);
```

kruh_io.h

Oddelený preklad

slido slido.com
1412341
PrPr – P11

kruh_main.h

```
extern double obvod;
extern double vyp_priemer(double polomer);
```

kruh_io.h

```
#define CHYBA_DAT -1

extern double vstup_dat();
extern double vystup_dat(double obsah);
```

kruh_main.c

```
#include <stdio.h>
#include "kruh_main.h"
#include "kruh_io.h"

double obvod;
#define pi 3.14;

static double polomer;
static double obsah;

static void vyp_obsah
            (double polomer);
static doble vyp_obvod();
static void vypocet(void);

void main() {
  polomer = vstup_dat();
  if (polomer == CHYBA_DAT)
    printf("Chybny polomer.");
  else {
    vypocet();
    vystup_dat(obsah);
  }
}

static void vyp_obsah(double
                      polomer)
{
  obsah = PI * polomer *polomer;
}

static doble vyp_obvod()
{
  return(PI*vyp_priemer(polomer));
}

static void vypocet(void)
{
  obvod = vyp_obvod();
  vyp_obsah(polomer);
}

double vyp_priemer(double polomer)
{
  return (2 * polomer);
}
```

kruh_io.c

```
#include <stdio.h>
#include "kruh_main.h"
#include "kruh_io.h"

#define kontrola(x) ((x >= 0.0) ? x : CHYBA_DAT)

static double polomer;

double vstup_dat(void) {
    printf("Zadaj polomer kruznice: ");
    scanf("%lf", &polomer);
    return kontrola(polomer);
}


void vystup_dat(double obsah) {
    double priemer;

    printf("Obsah kruhu s polomerom %6.2f je %.2f\n",
        polomer, obsah);
    priemer = vyp_priemer(polomer);
    printf("Obvod kruhu s polomerom %6.2f je %.2f\n",
        polomer, obvod);
}
```

Ďakujem vám za pozornosť!

Spätná väzba: *https://forms.gle/6q5D2G6UwrtimXEx9*