Procedurálne programovanie

slido.com
2265888
PrPr – P7

Ján Zelenka

Ústav Informatiky

Slovenská akadémia vied

Obsah prednášky

slido slido.com
2265888
PrPr – P7

1. **Reťazce - Opakovanie**
2. **Konštantné premenné**
3. **Parametre funkcie `main()`**

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

Reťazce

slido.com
2265888
PrPr – P7

Reťazce

- znaková premenná uchováva kód znaku
  - celé číslo
- reťazce sú jednorozmerné **polia typu char**
  - z celkovej pamäte je aktívna len časť od začiatku poľa po znak '\0' ukončovací znak
  - ak nie je reťazec ukončený znakom '\0', považuje sa za reťazec celá nasledujúca oblasť v pamäti až do najbližšieho znaku '\0'
  - dĺžka reťazca je pozícia znaku '\0'
    - nemusíme si ju pamätať v premennej
    - skrátenie pomocou posunu null

Reťazce

reťazec s najviac **5 znakmi**:

- staticky:

```
char s[6] = "ahoj";
```

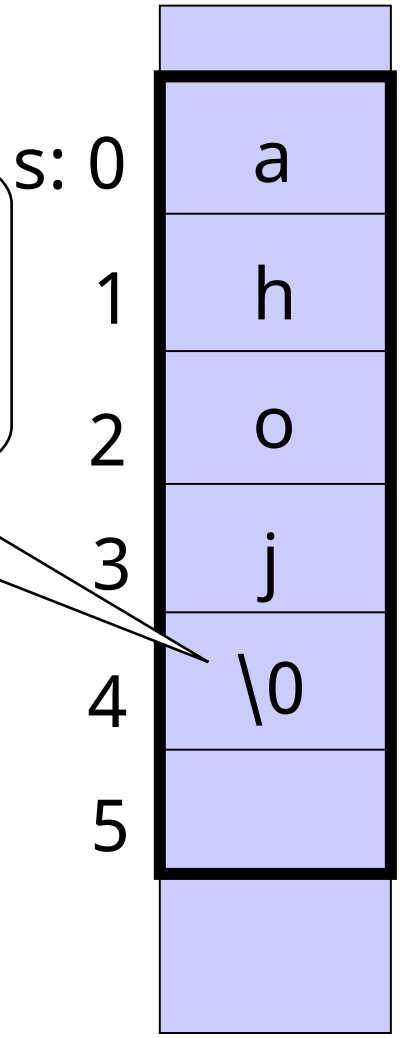
```
char s[] = "abrakadabra";
```

- "dynamicky":

```
char *s;
s = (char *) malloc(6);
```

znak null alebo 0
(ASCII kód 0)
nepliesť si s cifrou 0
(jej kód je 48)

inicializuje sa miesto
práve pre daný text



Reťazce

```
char s[] =
{'A','h','o','j'};
```

neukončený '\0'

```
char s[] =
{'A','h','o','j',0};
```

ukončený '\0'

```
char s[6] = "ahoj";
```

pole inicializované na "ahoj"

```
char *s = "ahoj";
```

s nie je dynamický reťazec, ale ukazovateľ na typ **char** a je inicializovaný adresou reťazcovej konštanty (ktorá má obsah ahoj)

Reťazce

```
char *retazec = "Jan";
retazec = "Karol";
retazec[3] = 's';
```

nie je možné meniť reťazcovú konštantu

```
char s[10] = "Hallo";
s = "ahoj";
s[3] = 'r';
```

nie je možné priradiť statickému reťazcu konštantu

Reťazce

- 'z' vs "retazcova_konstanta"
- statická časť programu
- Inicializácia reťazca (== pole znakov ukončené '\0'):
- ako pole:
- pomocou konštanty:

```
char s[] = {'A','h','o','j','\0'};
```

```
char s[] = "Ahoj";
```

Reťazce

- Pracujeme s ním ako s jednorozmerným poľom
  - operátor []

```
char s[10];

for (i = 0; i < 10-1; i++)
    s[i] = '*';
s[10-1] = '\0';
```

dôležité: ukončiť reťazec!

Reťazce

- Môžeme využiť ukazateľovú aritmetiku

```
char s[] = "Ahoj svet";

*(s + 4) = 0; //s[4]

char* s2 = s + 5;
```

s je adresa na prvý znak reťazca

zmena konca reťazca

ukazovateľ na časť iného reťazca

- Reťazce nemenia svoju veľkosť automaticky
  - automatické zväčšovanie
  - zápis za koniec poľa
- operátor + sčíta ukazovatele na reťazce nie je to zreťazenia (Java, C++)

Reťazce

- dynamicky vytvorený reťazec sa nedá inicializovať
- (inicializácia sa vykonáva pri preklade, kedy ešte pole nie je vytvorené)

```
char *s;
s = (char *) malloc(10);
s = "ahoj";

char c;
int i=0;
while ((c=getchar()) != '\n' && i < 9)
   s[i++] = c;
s[i] = '\0';
```

alokovanie 10 znakov do **s**

načítanie aj pomocou **scanf**

Reťazce

```
char s[10];
...
scanf("%s", s);
```

sem nepartrí **&**, pretože **s** je adresa

- **scanf()** vynecháva biele znaky a číta po prvý biely znak
  - Výhoda nemusíme ošetrovať prázdne znaky na začiatku riadku
  - Nevýhoda načítavanie sa ukončí pri prvom prázdnom znaku
- ak je na vstupe **" ahoj Eva!"**, **scanf()** prečíta iba **"ahoj"** a zvyšok zostáva v bufferi klávesnice

Formátovaný vstup a výstup z a do reťazca

- použitie výhod formátovaného vstupu a výstupu, ale nevypísať nič na obrazovku ani nenačítavať

```
int sprintf(char *s, char *format, ...);
```

pracuje ako **fprintf**, ale zapisuje do reťazca **s**

```
int sscanf(char *s, char *format, ...);
```

pracuje ako **fscanf**, ale číta z reťazca **s**

Riadkovo orientovaný vstup a výstup z terminálu

- okrem `scanf()`, `printf()` aj:

```
char *gets(char *s);
```

číta celý riadok do **`s`**: na koniec nezapíše **`\n`**, ale **`\0`**, vracia ukazovateľ na **`s`**, ak je riadok prázdny dáva do **`s \0`** a vráti **`NULL`**

```
int puts(char *s);
```

vypíše reťazec a odriadkuje (**`\n`**), vráti nezáporné číslo ak sa podarilo vypísať, inak **`EOF`**

Riadkovo orientovaný vstup a výstup z terminálu

- okrem **`fscanf()`**, **`fprintf()`** aj:

```
char *fgets(char *s, int max, FILE *fr);
```

číta riadok zo súboru do konca riadku ale maximálne **`max`** znakov, načítané zapíše do **`s`** (aj s **`\n`**), vracia ukazovateľ na **`s`**, ak je koniec súboru tak **`NULL`**

```
int fputs(char *s, FILE *fw);
```

do súboru **`fw`** vypíše reťazec **`s`**, neodriadkuje ani neukončuje pomocou **`\0`**, vráti nezáporné číslo ak sa podarilo vypísať, inak **`EOF`**

Reťazce

Definícia typu pre reťazce

```
typedef char *STRING;
```

Treba rozlišovať:

- nulový ukazovateľ **`NULL`** a nulový reťazec `'\0'`:
  - nulový ukazovateľ neukazuje na žiadne miesto v pamäti,
  - nulový reťazec má v 0-tom prvku znak `'\0'`
- `"x"` a `'x'`:
  - `"x"` je reťazec s jedným znakom ukončený `'\0'` (2 Byty)
  - `'x'` je jeden znak (1 Byte)

Štandardné funkcie pre prácu s reťazcami

slido slido.com
2265888
PrPr – P7

- Reťazec sa nedá kopírovať priradením
- Reťazce sa nedajú porovnávať (==, !=, >, < atď.)

Funkcionality si treba naprogramovať, alebo použiť funkcie z knižnice

- nie sú súčasťou samotného jazyka C

- Využitie knižnice **string.h**
  - dokumentácia http://www.cplusplus.com/reference/cstring/
  - prehľad: https://en.cppreference.com/w/c/string/byte
- Základné pravidlá
  - reťazec je ukončený znakom '\0'
  - pri modifikácii reťazca má výstupný reťazec dostatočnú veľkosť

Štandardné funkcie pre prácu s reťazcami

```
int strlen(char *s);
```

vracia dĺžku reťazca (bez `\0`)

```
char *strcpy(char *kam, char *co);
```

kopírovanie reťazca **`co`** do **`kam`**, vracia ukazovateľ na **`kam`**
(reťazec **`kam`** musí byť dosť dlhý)

Štandardné funkcie pre prácu s reťazcami

```
char *strcat(char *kam, char *co);
```

pripojí reťazec **co** ku **kam**, vracia ukazovateľ na **kam**
(reťazec **kam** musí byť dosť dlhý
-> strlen(kam) + strlen(co) + 1)

```
int strcmp(char *s1, char *s2);
```

vracia **0**, ak sú reťazce rovnaké, záporné číslo, ak **s1** je skôr (abecedne), inak kladné číslo (porovnáva sa na základe kódov znakov -> 'Z' je skôr ako 'a'

Štandardné funkcie pre prácu s reťazcami

```
char *strchr(char *s, char c);
```

nájdenie znaku **c** v reťazci **s**, vracia prvý výskyt znaku, ak sa v **s** nenachádza, vráti **NULL**

```
char *strstr(char *s1, char *s2);
```

vracia ukazovateľ na prvý výskyt reťazca **s2** v reťazci **s1**, v prípade neúspechu **NULL**

Práca s časťou reťazca

- podobne ako uvedené funkcie,
- v názve je **`n`** (zo slova *number*), napr. **`strncpy()`**:

```
char *strncpy(char *s1, char *s2, int max);
```

kopírovanie najviac max znakov z reťazca **`s2`** do **`s1`**, vracia ukazovateľ na **`s1`**

Práca s reťazcom naopak

- podobne ako uvedené funkcie,
- v názve je r (zo slova *reverse*), napr. **`strrchr()`**:

```
char *strrchr(char *s, char c);
```

nájdenie znaku **`c`** v reťazci **`s`**, vracia posledný výskyt znaku, ak sa v **`s`** nenachádza, vráti NULL

Prevody reťazcov na čísla

- konvertovanie reťazca číslic na číslo (funkcie definované v **`stdlib.h`**)

```
int atoi(char *s);
```

prekonvertuje reťazec znakov na **`int`**

```
long atol(char *s);
```

prekonvertuje reťazec znakov na **`long`**

```
float atof(char *s);
```

prekonvertuje reťazec znakov na **`float`**

Pri vstupe a výstupe nie je konverzia potrebná (**`scanf()`** a **`printf()`**)

Kontrola znakov v reťazci

Knižnica **type.h** – http://www.cplusplus.com/reference/cctype/

- **isalnum(znak)** – je znak písmeno alebo číslica?
- **isdigit(znak)** – je znak desiatková číslica?
- **isxdigit(znak)** – je znak hexadecimálna číslica?
- **isalpha(znak)** – je znak písmeno (veľké/malé)?
- **islower(znak)** – je znak malé písmeno?
- **isupper(znak)** – je znak veľké písmeno?
- **ispunct(znak)** – je znak špeciálny (vypísateľný)?
- **isprint(znak)** – dá sa znak vypísať (medzera, písmeno...)?
- **isgraph(znak)** – má znak grafickú podobu (písmeno, číslica…)?
- **isspace(znak)** – je znak biely (nový riadok, medzera..)?
- **iscntrl(znak)** – je znak riadiaci?

Prevod znakov:

- **tolower(znak)** – veľké písmeno prevedie na malé písmeno
- **toupper(znak)** – malé písmeno prevedie na veľké písmeno

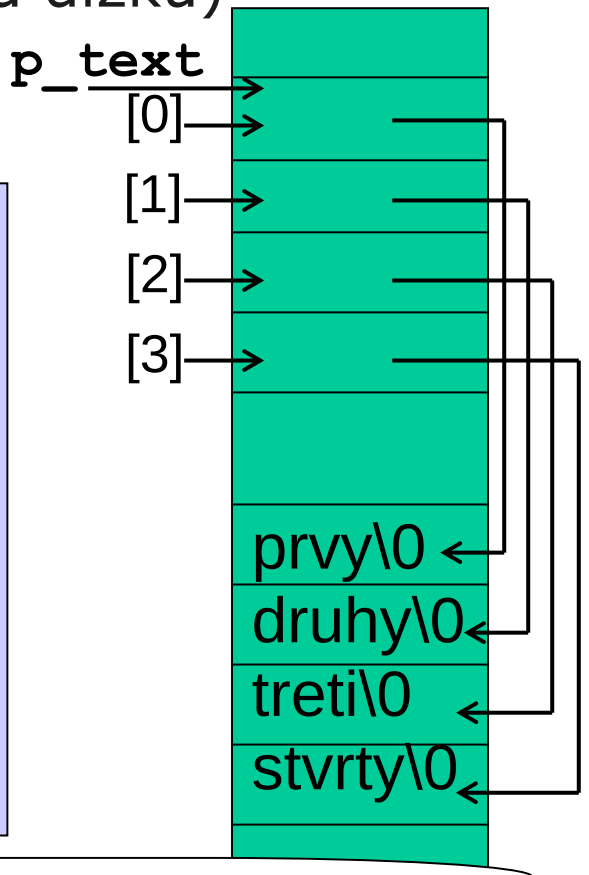
Pole reťazcov

- pole ukazovateľov na reťazce
- zvyčajne zubaté (reťazce/riadky majú rôznu dĺžku)

iba reťazec **p_text[2]** je alokovaný dynamicky, ostatné sú statické

```
char *p_text[4];

p_text[0] = "prvy";
p_text[1] = "druhy";
p_text[2] = (char *) malloc(6);
strcpy(p_text[2], "treti");
p_text[3] = "stvrty";
strcpy(p_text[3], "Stvrty");
```



chyba **p_text[3]** zápis do pamäti s konštantným reťazcom

Pole reťazcov

```
char *p_text[4], c, *p;
...
c = p_text[0][0];

p = &p_text[0][0];
while (*p != '\0')
    putchar(*p++);

printf("%s \n", p_text[1]);

puts(p_text[2]);
```

prístup k jednotlivým prvkom reťazca

vypísanie reťazca po znakoch

vypísanie reťazca pomocou **`printf()`**

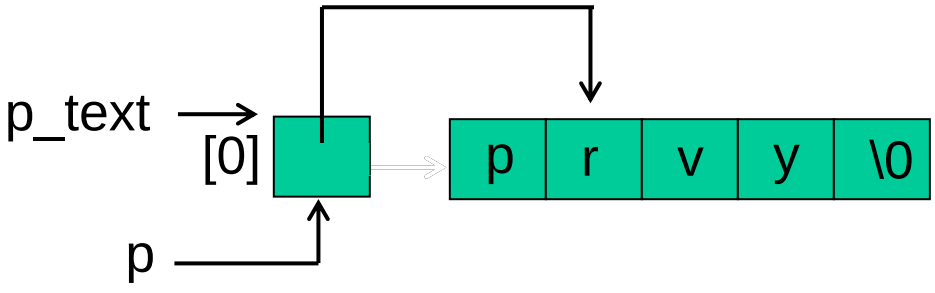
vypísanie reťazca pomocou **`puts()`**

Pole reťazcov

```
char *p_text[4], **p;
...
p = p_text;
puts(++*p);
```

- **p** ukazuja na **p_text**,
- ***p** ukazuje na **p_text[0]**
- príkaz **++*p** zväčší hodnotu na tej adrese o 1, teda zväčší **p_text[0]**

- vypíše sa "**rvy**" pretože ***p** ukazuje na nultý prvok poľa **p_text**
- **p_text[0]** potom ukazuje na "**rvy**" a táto zmena je trvalá



Pole reťazcov

slido slido.com
2265888
PrPr – P7

```
char *p_text[4], **p;
...
p = p_text;
puts(*++p);
```

vypíše sa **"druhy"** pretože sme najprv zvýšili **p** o 1 (posunuli sme ho na druhý riadok a potom vypísali reťazec, kam ukazuje **p**)

```
char *p_text[4], **p;
...
p = p_text;
for (i = 0; i < 4; i++)
    puts(*p++);
```

++ má väčšiu prioritu ako ***,** riadok sa najprv vypíše a ukazovateľ **p** sa posunie na druhý riadok.

Časté chyby pri práci s reťazcami

slido slido.com
2265888
PrPr – P7

- nedostatočne veľký cieľový reťazec (napr. **`strcpy()`**)
- nekorektné ukončenie reťazca
  - funkcie z knižnice string sa správajú nekorektne
  - napr. zapisujeme mimo rozsah poľa
  - vzniká napr. pri prepísaní, nevložením... znaku `'\0'`
- dĺžka/veľkosť reťazca sa udáva bez znaku `'\0'` (**`strlen()`**)
- operátor `==` neporovnáva reťazce (**`strcmp()`**)
- operátor `+` nezreťazuje reťazce

Konštantné premenné

slido slido.com
2265888
PrPr – P7

Pozor, **konštantné výrazy** (napr. definícia statického poľa) sa definujú **pomocou direktívy predprocesora**
(ukážeme si na nasledujúcej prednáške)

Kľúčové slovo const

- označuje nemennú (*read-only*) **premennú**
  - napr. matematické konštanty, stavy programu...
- prechádzame nechceným implementačným chybám
  - jej hodnota sa nemení
  - argument funkcie, ktorý sa nesmie zmeniť

```
void procedura(const float f){
    const int a;
    cont int i = 10;
}
```

chyba, neinicializovaná konštanta

Kľúčové slovo const

slido slido.com
2265888
PrPr – P7

- používajte ho čo najčastejšie
  - zlepšuje typovú kontrolu
  - nemeníte konštantné premenné
  - lepšia optimalizácia prekladačom
  - zlepšuje prácu ďalším programátorom s vaším kódom
- premenná s **const** je lokálna v danom súbore

Reťazcové konštanty

slido slido.com
2265888
PrPr – P7

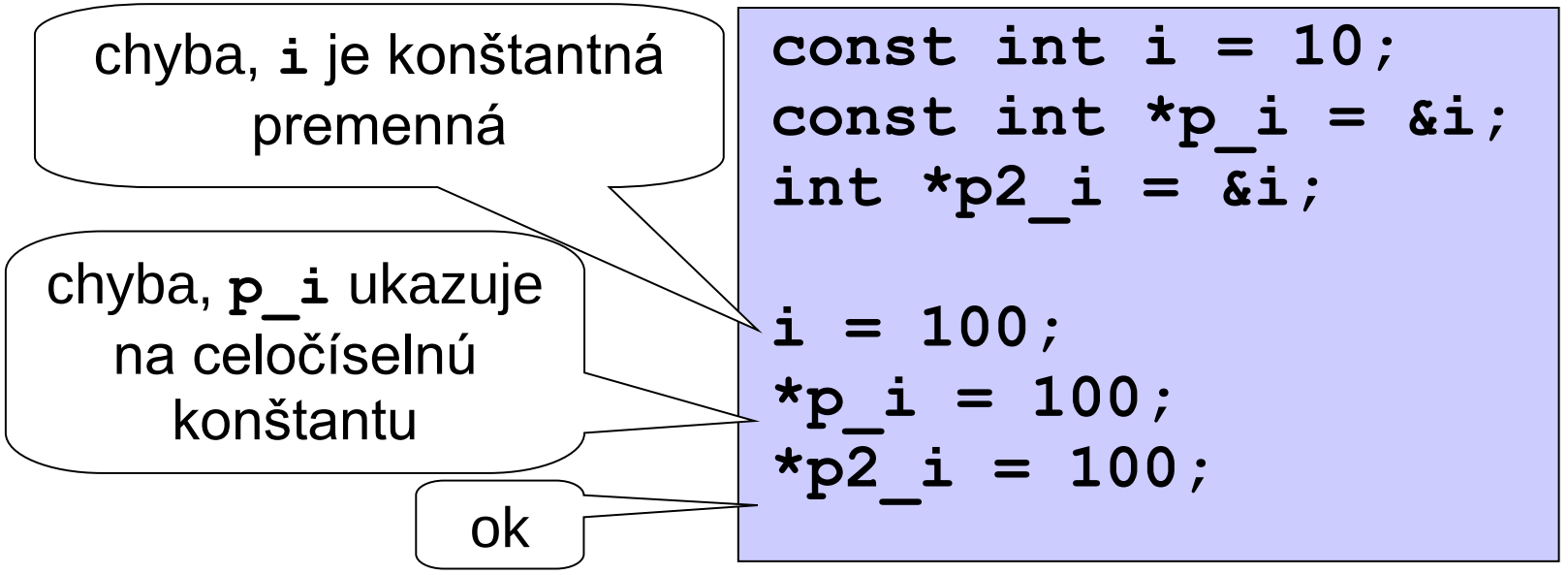
"Ahoj"

- uložený v statickej časti
- je typu **`char*`**
- Vhodnejšie je s ním pracovať ako **`const char*`**

```
const char *p_text[4];
```

Kľúčové slovo const a ukazovateľ

- konštantná je iba hodnota premennej
  - platí pre ukazovateľ a jeho dereferencovanie
- Nie je konštantné miesto, kam ukazovateľ ukazuje
  - const je plytké
  - konštantnú premennú vieme modifikovať cez nekonštantný ukazovateľ



Kľúčové slovo const a ukazovateľ

```
int i = 10;
```

```
const int i_2 = 10;
```

```
const int *p_2 = &i;
```

```
int* const p_3 = &i;
```

```
const int* const p_4 = &i;
```

`i_2` je konštantná premenná

`p_2` ukazuje na celočíselnú konštantu

`p_3` je konštantná premenná

`p_4` je konštantný ukazovateľ na celočíselnú konštantu

```
i = 100;
```

```
i_2 = 100;
```
chyba

```
*p_2 = 100;
```
chyba

```
*p_2 = &i;
```

```
*p_3 = 100;
```

```
*p_3 = &i;
```
chyba

```
*p_4 = 100;
```
chyba

```
*p_4 = &i;
```
chyba

Ukazovateľ na konštantu argumentom funkcie

slido slido.com
2265888
PrPr – P7

```
void procedura(const int *pole){
    pole[0] = 10;
    int *pom_pole = pole;
    pom_pole[0] = 10;
}
```

chyba, pole je "read-only"

warning: initialization discards '**const**' qualifier from pointer target type

ok

Parametre funkcie `main()`

slido.com
2265888
PrPr – P7

Ako prijíma program vstupné dáta

slido slido.com
2265888
PrPr – P7

- štandardný vstup (napr. klávesnica)
- súbor (napr. disk)
- zdieľaná pamäť
- príkazový riadok (argumenty pri spustení programu)
- správy (message queue)
- sieťová komunikácia (sokety)
- prerušovanie, semafory

Funkcia `main()`

slido slido.com
2265888
PrPr – P7

```
int main()
```

- návratová hodnota: vracia správu operačnému systému
- argumenty:

počet parametrov programu

parametre programu predané pri spustení

```
int main(int argc, char *argv[])
```

**argv==*argv[]

premenné prostredia

```
int main(int argc, char **argv,char **envp)
```

- ak je **`argc`** > 0, prvý argument je meno, alebo cesta k spustenému programu
- jednotlivé premenné prostredia sa dajú vypýtať pomocou funkcie **`getenv()`** z knižnice **`stdlib.h`**
  - premenné prostredia nemeňte

Parametre funkcie `main()`

```
int main(int argc, char *argv[])
```

program nazveme napr. **`test`**,
volanie: **`test parameter1 parameter2`**

→ **`argc: 3`**
`argv[0]: test`
`argv[1]: parameter1`
`argv[2]: parameter2`

pozn.: názov je v **`argv[0]`**

medzera je oddeľovač parametrov (parameter s medzerou musí byť v úvodzovkách

volanie: **`test "ahoj nazdar" cau`** → **`argc`**: 3

Parametre funkcie `main()`

ak je argument **"-h"**, program vypíše **"help"**, inak **"program"**
nakoniec program vypíše premenné prostredia

```
#include <stdio.h>

int main(int argc, char *argv[], char *envp[]) {
  if(argc == 2 && !strcmp(argv[1], "-h"))
    printf("help\n");
  else
    printf("program\n");

  printf("\nPremenne prostredia:\n");
  for(int i=0; envp[i] != NULL; i++)
    printf("%s\n", envp[i]);

  return 0;}
```

nie je známy počet premenných prostredia.
Štandard garantuje, že posledný je **NULL**

Ďakujem vám za pozornosť!

slido.com
2265888
PrPr – P7

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