Programátorské minimum

Daniel Hládek

Table of Contents

Úvod

Jazyk C pre jednoducho uvažujúcich.

Začínajúcich programátorov často odradí veľké množstvo pojmov ktoré je nutné sa naučiť a technologických problémov, ktoré je potrebné prekonať. Naučiť sa programovať znamená prekonanie týchto počiatočných prekážok. Cieľom tejto príručky je priblížiť jazyk C ľuďom bez predchádzajúcej skúsenosti s programovaním a uľahčiť prekonanie počiatočných prekážok.

V sérii niekoľkých tutoriálov Vás príručka naučí základné programátorské postupy. Naučiť sa programovať sa nedá inak ako vyskúšaním "na vlastnej koži". Predpokladáme, že máte počítač s nainštalovaným prekladačom, textovým editorom a viete spustiť príkazový riadok.

Pre ešte väčšie zjednodušenie je možné použiť C Online IDE (Codechef alebo IDEOne) kde nie je potrebné pracovať s klasickým prekladačom. Programujete vo webovom prehliadači, vaše výsledky sa preložia na vzdialenom servri a zobrazia. Nevýhodou využívania webového prostredia je jeho prílišná jednoduchosť a obmedzenosť.

Naučíte sa za 4 hodiny

Programátorské minimum:

1. Napísať a preložiť triviálny program.
2. Vypísať správu.
3. Vytvoriť a inicializovať premennú.
4. Načítať číslo zo štandardného vstupu.
5. Navrhnúť funkciu s jedným parametrom, ktorá vracia jednu hodnotu.
6. Zavolať vlastnú funkciu s jedným parametrom a uložiť návratovú hodnotu.
7. Vypísať správu s parametrom.

Príklad

```c
#include <stdio.h>

float mocnina(float);

int main() {
    printf("Mocninová kalkulačka\n");
    float vysledok = 0;
    printf("Výsledok je zatiaľ %f\n",vysledok);
    char vstup[10];
    printf("Zadaj hodnotu na max. 10 miest:");
    fgets(vstup,10,stdin);
    printf("Zadali ste %s\n",vstup);
    float parameter = 0;
    sscanf(vstup,"%f",&parameter);
    printf("Hodnota parametra je %f\n",parameter);
    vysledok = mocnina(parameter);
    printf("Výsledok je %f\n",vysledok);
    return 0;
}

float mocnina(float arg){
    float parameter = arg * arg;
    return parameter;
}
```

Ahoj Svet

Cieľom tohto tutoriálu je naučiť vás vytvoriť triviálny program v jazyku C.

Naučíte sa

- Zdrojový kód je zápis algoritmu v programovacom jazyku.
- Programovací jazyk je súbor pravidiel pre zápis algoritmov.
- Spustiteľný kód získame spracovaním (prekladom) zdrojového kódu
- Spustiteľný kód je plnohodnotný program

Vytvorenie zdrojového textu

V textovom editore si otvorme súbor, ktorý môžeme nazvať `hello.c` a do neho napíšeme:

```
#include <stdio.h>

int main(){
    printf("Ahoj svet\n");
    return 0;
}
```

Textový súbor uložíme a prvý krôčik máme za sebou. Práve sme vytvorili svoj prvý program.

Preklad

Náš program, je síce správny, ale počítač bude mať ťažkosti s jeho vykonaním. Zdrojový text v jazyku C nie je priamo vykonateľný. Našou ďalšou úlohou bude preklad zdrojového textu do tvaru, ktorý sa dá spustiť. Požitie kompilátora na zdrojový text, ktorý sme vytvorili je jednoduché:

```
gcc hello.c -o hello
```

Ak sme pri prepise neurobili žiadnu chybu, tak by preklad mal prebehnúť bez problémov a vznikne spustiteľný súbor `hello`.

Chybové hlásenia

Ak sme predsa len nejakú chybu urobili, prekladač napíše chybovú hlášku a preklad neprebehne. Pozorne si hlásenie prečítajte a skúste opraviť chybu. Chybové hlásenie (skoro) vždy obsahuje číslo riadku a znak, kde sa chyba nachádza a podľa toho sa ku chybe vieme vrátiť. Ak je chybových hlásení viac, ignorujeme všetky okrem prvého. Opravíme prvú chybu, preložíme program a sledujeme, či oprava pomohla. Ak nie, postup opakujeme dovtedy, pokiaľ nie je preklad úspešný.

Spustenie programu

Ak preklad prebehol správne, môžeme vyskúšať výsledok. Najprv overíme situáciu, či sa tam spustiteľný súbor naozaj nachádza:

```
ls
hello.c hello
```

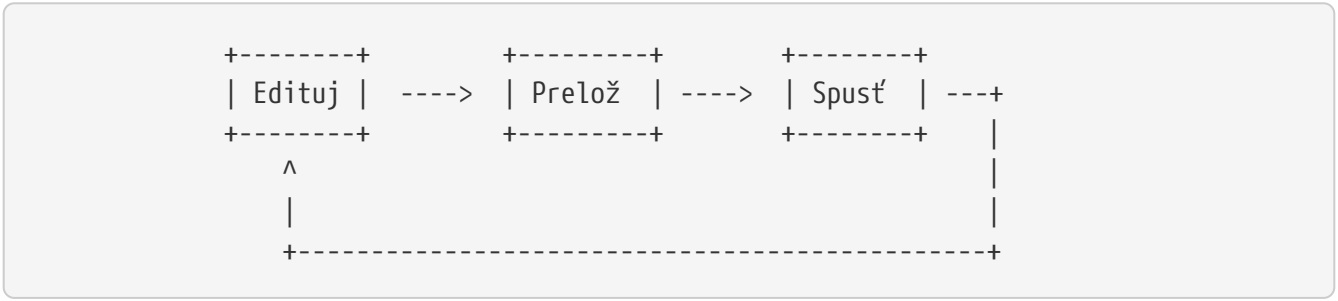
Príkaz `ls` by mal zobraziť zdrojový súbor aj spustiteľný súbor. Spustiteľný súbor môžeme spustiť:

```
./hello
Ahoj svet
```

Vývojový cyklus

Tento proces budeme opakovať stále dookola. V prvom kroku sme vytvorili alebo upravili existujúce zdrojové kódy. V druhom kroku sme sa pokúsili ich preložiť. Ak sa vyskytla chyba, tak sme sa museli vrátiť, znova upraviť zdrojové kódy a znova program preložiť. Posledným krokom vývojového cyklu je spustenie programu. Ak sme s výsledkom spokojní, nemusíme pokračovať a výsledný program môžeme odovzdať do používania. Ak nie, musíme začať znova.

Postup prác pri programovaní



Úprava programu

Ak aj spustenie prebehlo úspešne, môžem Vám gratulovať - práve ste sa stali (začínajúcim) programátorom. Náš prvý program je hotový.

Základy sú síce vybudované, ale výsledná stavba je malá. Zatiaľ vieme slušne pozdraviť, ale to je všetko. Skúsime využiť vedomosti ktoré máme aby sme výsledok trochu vylepšili. Okrem sveta skúsme pozdraviť aj niekoho iného.

Nájdime riadok v programe, ktorý pravdepodobne spôsobuje výpis správy na obrazovku a pozrime sa na neho bližšie:

```
①
printf("Ahoj svet\n")②
  ;③
```

Okrem správy "Ahoj svet" si všimneme niekoľko vecí, ktoré nám veľa povedia o jazyku C.

① Okrem správy, ktorá sa má vypísať tam vidíme printf a úvodzovky. Takýto zápis so zátvorkami nazývame **volanie funkcie**, Časť pred zátvorkou je **názov funkcie** a časť v zátvorke sú **argumenty funkcie** .

② Správa `"Ahoj svet"` je ohraničená úvodzovkami a zakončená znakom `'\n'`. Znaky ohraničené úvodzovkami nazývame **reťazec**. `'\n'` znamená nový riadok (line feed).

③ Riadok je zakončený bodkočiarkou. Takýto riadok nazývame **príkaz**.

Program v jazyku C sa skladá z definícií funkcií a ich volaní. Každú funkciu v jazyku C musíme pred volaním definovať, lebo jazyk C ako taký nepozná žiadne funkcie, iba spôsob ako ich definovať a volať.

V prípade funkcie `printf` je situácia jednoduchá, lebo funkciu už definoval niekto za nás. Nám už iba stačí "naučiť" kompilátor, čo znamená `printf` tým, že oznámime, kde definíciu funkcie nájde.

Na overenie týchto tvrdení použijeme tzv. *kreatívnu chirurgiu* a mierne zmrzačíme náš program. Upravíme niektorý riadok programu a zopakujeme proces prekladu. S vedeckým odstupom sledujeme čo sa stane:

```c
#include <stdio.h> ③

int main(){
    printf("Ahoj svet\n"); ① ②
    return 0;
}
```

① Zmeňte `"svet\n"` na `"TUKE"` Zistili sme, že `"\n"` je zápis pre koniec riadku.

② Zmeňte `printf` na `print` Zistili sme, že takú funkciu systém nepozná. Opravte chybu a pokračujeme.

③ Vymažte riadok s `#include`. Zistili sme, že napriek očakávaniam kompilátor hlási chybu na riadku s `printf`.

Zoznam funkcií, ktoré môžeme pri práci použiť sa nazýva "Štandardná knižnica jazyka C".

Úlohy na precvičenie

Modifikujte program tak, aby do prvého riadka vypísal Vaše meno a do druhého riadka Vašu obľúbenú farbu napr.:

```
Moje meno je Chuck Norris
   a moja obľúbená farba je krvavo čierna.
```

Pozrite si dokumentáciu štandardnej knižnice a skúste využiť ľubovoľnú ďalšiu funkciu, napr. `sleep()`. Nezabudnite, že pred použitím musíte funkciu "naučiť" pomocou `#include`.

Premeň ma! Premenné a dátové typy

V tomto bloku sa naučíme využívať pamäť počítača.

Naučíte sa

- Je možné uložiť hodnotu a tú neskôr použiť
- Každá hodnota má dátový typ.
- Dátové typy je možné meniť, má to ale vedľajšie efekty.
- Výpis premennej v rôznom formáte
- Rovnaký typ je možné vypisovať rôznymi spôsobmi — aj číslo s desatinnou čiarkou vieme zapisovať rôznym spôsobom
- Čítať dokumentáciu funkcie printf

Celočíselná premenná

V prvom kroku upravíme náš jednoduchý program:

```
#include <stdio.h>

int main(){
    int pocitadlo  = 7;
    printf("Ahoj svet\n",pocitadlo);
    return 0;
}
```

Pridali sme riadok, pomocou ktorého sme vytvorili pamäťové miesto s názvom `pocitadlo`. Číslo v pamäťovom mieste vieme podľa ľubovôle meniť, preto ho nazývame **premenná**.

Pozrime sa bližšie na príkaz pomocou ktorého vytvárame premennú:

```
int pocitadlo  = 7;
```

Prvá vec, ktorú si všimneme je kľúčové slovíčko `int`. Týmto slovíčkom označujeme **dátový typ** s ktorým pracujeme. Dátových typov je niekoľko, v tomto prípade pracujeme s celými číslami (integer). Celočíselný dátový typ nám umožňuje pracovať s celými číslami (ale nie s číslami s desatinnou čiarkou).

Druhým slovíčkom za dátovým typom je **názov premennej**. Názov premennej funguje podobne ako menovka na poštovej schránke alebo názov políčka vo formulári. Pomocou názvu premennej vykonávame všetky operácie.

Tretím slovíčkom na riadku je symbol `=`. V klasickom matematickom zápise to znamená znamienko rovnosti. V zápise jazyka C je ale význam úplne iný a znamená operáciu **priradenia**. V tomto prípade sa do premennej s názvom `pocítadlo` priradí (zapíše) hodnota 7.

Výpis premennej

To, že si vieme zapamätať nejaké číslo je síce skvelé, ale samo o sebe zbytočné. Aby boli premenné naozaj užitočné, potrebujeme s premennou vykonávať nejaké operácie. Pozrime sa bližšie na riadok s príkazom na výpis na obrazovku a modifikujeme ho tak, aby vypisoval aj hodnotu premennej:

```
printf("Počítadlo má hodnotu %d\n", pocitadlo);
```

V riadku s príkazom na výpis správy sme za čiarku pridali ďalší argument s menom premennej ktorú máme zahrnúť do výpisu. Znaky medzi úvodzovkami obsahujú správu ktorá sa má vypísať a znaky, ktoré zastupujú hodnotu premennej na výpis. Zástupné znaky vyjadrujú akým spôsobom sa má premenná vypísať, v tomto prípade `%d` znamená výpis vo forme celého čísla.

Úloha na vyriešenie

Preštudujte si možné formátovacie reťazce pre funkciu `printf` a vypíšte hodnotu premennej na minimálne zadaný počet miest, chýbajúce znaky doplňte nulami. Manuálovú stránku funkcie `printf` si zobrazíte príkazom `man printf` (v príkazovom riadku) alebo vyhľadáte na internete.

Inicializácia premennej

Skúste upraviť program nasledovným spôsobom (vynecháme operáciu priradenia):

Program so zle inicializovanou premennou

```
#include <stdio.h>

int main(){
    int pocitadlo;
    printf("Ahoj %d\n",pocitadlo);
    return 0;
}
```

Zistili sme, že na prvý pohľad síce program pracuje normálne, ale nie je jasné aká hodnota bude v premennej `pocítadlo`.

Zabudnutá inicializácia spôsobuje nepredvídateľné chovanie programu.

Keďže sme žiadnu hodnotu do premennej nepriradili a v ďalšom riadku túto neznámu hodnotu využívame, program sa správa nepredvídateľne. Ak do premennej žiadnu hodnotu nepriradíme tak jej hodnota závisí od toho, akú pamäť nám operačný systém pridelil. V premennej môže byť hodnota nula, ale aj akákoľvek iná hodnota ktorú nevieme predvídať. Na skutočnosť, že pracujeme s nedefinovanou hodnotou nás ale program nevie upozorniť a vyzerá to tak, že je všetko v poriadku. Je zodpovednosťou programátora aby sa vyhol takýmto situáciám a využíval postupy a nástroje ktoré tomu zabránia.

```c
#include <stdio.h>

int main(){
    int pocitadlo = 0; ①
    printf("Ahoj %d\n",pocitadlo);
    return 0;
}
```

① Táto premenná je inicializované správne a vieme že vždy bude nulová.

Dátový typ

Dátový typ definuje možnosti a obmedzenia premennej s ktorou pracujeme. Premenná funguje podobne ako obrazovka na kalkulačke. Obrazovka na kalkulačke má určité obmedzenia ktoré sa podobajú obmedzeniam dátového typu - na kalkulačke môžeme pracovať s číslami, ale nie s písmenami. Číslo na kalkulačke môžeme prečítať alebo ho zmeniť. Podľa typu kalkulačky s ním môžeme vykonávať sadu operácií - spočítavanie alebo násobenie.

Do teraz napísaný program nám umožňuje prácu iba s celočíselnými hodnotami. Do celočíselnej premennej nie je možné uložiť hodnotu s desatinnou čiarkou. Pokiaľ sa o to pokúsime, hodnota za desatinnou čiarkou sa zanedbá a dôjde k zaokrúhleniu smerom nadol. Na túto skutočnosť nás kompilátor nemusí upozorniť.

Modifikujme náš program:

Dochádza k automatickému zaokrúhľovaniu

```c
#include <stdio.h>

int main(){
    int pocitadlo = 1.1;
    printf("Ahoj %d\n",pocitadlo);
    return 0;
}
```

Aká hodnota sa vypíše?

Zmeňme formát výpisu premennej na číslo s desatinnou čiarkou:

Dochádza k nesprávnemu formátu výpisu

```c
#include <stdio.h>

int main(){
    float pocitadlo = 1.1;
    printf("Ahoj %d\n",pocitadlo);
    return 0;
}
```

Pozitívna zmena nenastala, lebo stále platia obmedzenia pre celočíselný dátový typ. Na to aby sme správne vedeli pracovať s číslom s desatinnu čiarkou, musíme použiť vhodný formátovací znak:

Správny typ premennej aj správny formát výpisu premennej.

```
#include <stdio.h>

int main(){
    float pocitadlo = 1.1;
    printf("Ahoj %f\n",pocitadlo); ①
    return 0;
}
```

① Všimnite si, že formátovacia značka `%d` (pre celé čísla) sa zmenila na `%f` (pre čísla s desatinnou čiarkou).

Dátový typ premennej a formát výpisu na obrazovku sú dve rozdielne veci. Aj keby sme zmenili formátovaciu značku na `%d`, obsah premennej `pocitadlo` to neovplyvní.

Operácie s dátovými typmi

Celočíselný dátový typ (`int`) je vhodný najmä ako počítadlo alebo na uchovávanie indexov. Nie je veľmi vhodný na matematické operácie ako napr. násobenie alebo delenie. Výsledkom delenia dvoch celých čísel je reálne číslo, a túto skutočnosť musíme brať do úvahy. Môžeme si to vyskúšať na jednoduchom matematickom programe, ktorý vykonáva operáciu delenia (`/`):

```
#include <stdio.h>

int main(){
    int vysledok = 5 / 2;
    printf("Ahoj %f\n",vysledok);
    return 0;
}
```

Keďže premenná `vysledok` má celočíselný dátový typ `int`, je jasné, že výsledok nebude správny. Navyše, nepomôže ani intuitívna oprava:

```
    float vysledok = 5 / 2;
    printf("Ahoj %f\n",vysledok);
```

Dôvodom je to, že podľa štandardu je výsledkom delenia dvoch celých čísel opäť celé číslo, takže automaticky dochádza k zaokrúhľovaniu. Tento proces, keď prekladač zmení dátový typ bez nášho dovolenia sa nazýva **implicitná dátová konverzia**. Do premennej, ktorá je vhodná na uloženia čísla s desatinnou čiarkou (typ `float`) sa uloží už zaokrúhlená hodnota. Prekladač jazyka C musíme presvedčiť, že delíme čísla s desatinnou čiarkou:

```
float vysledok = 5.0 / 2.0;
printf("Ahoj %f\n",vysledok);
```

Teraz to už funguje správne.

Typová konverzia

Matematické operácie vieme vykonávať aj s obsahom premennej, ktorý sme si zapamätali v predošlých krokoch. Stačí nahradiť číselné hodnoty v predošlom príklade názvami premenných, ktoré sme si inicializovali pred tým.

```
float podiel = delenec / delitel;
printf("Ahoj %f\n",podiel);
```

Samozrejme, musíme definovať premenné “deliteľ” a “delenec”, t.j. priradiť im vhodný dátový typ a počiatočnú hodnotu. Problém s dátovými typmi platí aj keď pracujeme s hodnotami, ktoré sú uložené v premenných. Navyše, prvý pohľad na riadok, kde priradíme hodnotu do premennej “podiel”, nám nič nepovie o tom, či operácia prebehne správne. Aby sme si boli istí, že delenie prebehne správne, musíme splniť tieto predpoklady:

- obidva operandy musia mať typ s desatinnou čiarkou
- deliteľ nesmie byť nula.

Keby sme chceli správne deliť dve celé čísla, nemusíme dostať správny výsledok:

```
int delenec = 5;
int delitel = 2;
float podiel = delenec / delitel;
printf("Ahoj %f\n",podiel);
```

Úloha na precvičenie

Čo sa stane ak bude deliteľ nulový? Čo sa stane, ak budeme ďalej narábať s výsledkom delenia nulou?

Vyhodnotenie operácií

Aby sme si boli istý, že delenie dvoch premenných bolo správne, musíme vykonať tzv. **explicitnú typovú konverziu**:

```
int delenec = 5;
int delitel = 2;
float podiel = (float)delenec / (float)delitel;
printf("Ahoj %f\n",podiel);
```

Typová konverzia je ďalšia operácia s premennými, ktorú pozná jazyk C. Do zátvorky uvedieme typ do ktorého chceme meniť. Prekladač vytvorí dočasné pamäťové miesto so zadaným dátovým typom do ktorého uloží hodnotu za zátvorkou. Toto dátové miesto s novým typom sa potom použije pri ďalších operáciách vo výraze.

Operácia priradenia (`=`) je vyhodnocovaná v určitom poradí. Najprv sa vyhodnotí výraz (*expression*) na pravej strane a výsledok sa uloží do premennej, ktorej názov sa nachádza na ľavej strane (*lvalue*). Iné pravidlá platia pre vyhodnocovanie operácie delenia. Najprv sa vyhodnotí výraz na ľavej strane, potom na pravej strane.

Kopírovanie hodnoty

Operácia priradenia je ekvivalentná operácii kopírovaniu hodnoty. Hodnotu z jednej premennej môžeme nakopírovať do druhej premennej a tam s ňou pracovať bez toho, aby to ovplyvnilo pôvodnú premennú. Môžeme napísať:

```
int povodna = 5;
int nova = povodna;
printf("Ahoj %d\n",nova);
```

Zistili sme, že hodnota premennej `povodna` sa nakopírovala do premennej `nova`. Že ide naozaj o operáciu kopírovania si môžeme overiť tým, že zmeníme hodnotu pôvodnej premennej a sledujeme, či to ovplyvní hodnotu novej premennej.

```
int povodna = 5;
int nova = povodna;
printf("Nova hodnota je  %d\n",nova);
printf("Povodna hodnota je  %d\n",povodna);
povodna = 4;
printf("Po priredeni je povodna hodnota  %d\n",povodna);
printf("a nova hodnota stale je  %d\n",nova);
```

Funguj - Návrh a používanie vlastnej funkcie

Naučíme sa vytvárať a využívať vlastné funkcie.

Naučíte sa:

- volanie funkcie bez parametrov
- parametre funkcie
- volanie funkcie odovzdávanie parametrov kópiou
- návratová hodnota funkcie a jej využitie

Definícia funkcie

S vytváraním a využívaním funkcií v jazyku C sme sa už stretli, možno aj bez toho aby ste to vedeli. V našom hračkovom programe sme si už definovali vlastnú funkciu, ktorej sme dali meno `main`.

```
#include <stdio.h>

int main(){ ①
    printf("Ahoj svet\n");
    return 0;
}
```

① Definícia funkcie `main`.

Definícia funkcie sa skladá z nasledovných častí:

- návratový typ (v tomto prípade `int`)
- názov funkcie (v tomto prípade `main`)
- argumenty funkcie (uvedené v okrúhlych zátvorkách, v tomto prípade žiadne)
- telo funkcie (časť v zložených zátvorkách)

Ak určíme nejaký návratový typ, tak sľubujeme prekladaču, že pomocou príkazu `return` vrátime hodnotu daného typu. Príkaz `return` spôsobí okamžité prerušenie vykonávania funkcie a funkcia vráti danú hodnotu, ktorú je možné ďalej spracovávať. Posledný príkaz

```
    return 0;
```

ukončí vykonávanie hlavnej funkcie a operačnému systému odovzdá hodnotu 0. Ak ako návratový typ určíme `void`, neočakávame žiadnu návratovú hodnotu a príkaz `return` nemusíme uviesť. Tieto znalosti môžeme využiť a definovať si vlastnú funkciu pre výpis správy.

Definícia vlastnej funkcie `pozdrav`

```
#include <stdio.h>


void pozdrav(){ ①
    printf("Dobry den\n");
}

int main(){ ②
    printf("Ahoj svet\n");
    return 0;
}
```

① Definícia funkcie `pozdrav`.

② Definícia funkcie `main`.

Čo sa stane? Preklad programu prebehne úspešne, to znamená, že program je syntakticky správny. Napriek očakávaniu ale zdvorilý pozdrav nebude vypísaný.

Úlohy na precvičenie

Ćo sa stane, keď vynecháme príkaz `return` z hlavnej funkcie? Čo sa stane, keď hlavnej funkcii určíme návratový typ `void`?

Volanie funkcie

Definícia funkcie je určitým spôsobom ekvivalentná k definícii slova v slovníku. Ak poznáme slovo, to ešte neznamená, že sme ho použili. Ak chceme definovanú funkciu aj použiť, musíme ju **zavolať**. Volanie funkcie bez parametrov vykonáme zapísaním mena funkcie a prázdnych okrúhlych zátvoriek. Každý príkaz musíme ukončiť bodkočiarkou.

```
#include <stdio.h>

void pozdrav(){
    printf("Dobry den\n");
}

int main(){
    pozdrav(); ①
    printf("Ahoj svet\n");
    return 0;
}
```

① Volanie vlastnej funkcie `pozdrav`

Zistili sme, že volanie vlastnej funkcie sa neodlišuje od volania funkcie zo štandardnej knižnice. Náš jednoduchý program neobsahuje príkaz na vykonanie funkcie `main`. Podľa dohody sa o niečo také

stará operačný systém pri spustení programu. Operačný systém pri spustení programu vyhľadá definíciu funkcie `main` a tej odovzdá riadenie. Až potom sa vykonávajú funkcie, ktoré určil programátor.

Návratová hodnota a argumenty funkcie

```
void pozdrav(){
    printf("Dobry den\n");
}
```

Takto definovaná funkcia nemá definovanú návratový hodnotu. Výpis správy, ktorý funkcia vykoná je možné brať ako **vedľajší efekt**, ktorý nemá priamy súvis so vstupom funkcie.

Funkciu si môžeme predstaviť ako kuchynského robota, ktorý vie spracovať určitý druh potravín. Vstupné argumenty sú suroviny, ktoré vkladáme do kuchynského robota. Návratový typ je výsledok spracovania vstupných surovín. Napríklad možným vstupom do mlynčeka na mäso je kus mäsa a výstupom je mleté mäso. Automatický mlynček na mäso bude mať problémy s niektorými surovinami. Napríklad orechy alebo hrozno na vstupe môžu spôsobiť zničenie robota alebo výstup nebude mať očakávané vlastnosti.

Funkcia s vedľajším efektom

```
                +---------+
vstup ---> | funkcia | ---> výstup
                +----+----+
                     |
                     v
               vedľajší efekt
```

Funkcia, ktorú sme definovali vyššie nemá definované vstupné argumenty ani výstupný typ. Funkciu si môžeme definovať tak, že výstupná hodnota závisí od vstupných argumentov. Vstupné argumenty sú definície premenných do ktorých sa priradí hodnota počas volania funkcie. Vstupné premenné uvedieme počas definície do okrúhlych zátvoriek a oddelíme ich čiarkou.

```
int spocitaj(int a,int b){
    return a + b;
}
```

Týmto kusom kódu sme definovali funkciu, ktorá vezme dve čísla, spočíta ich a vráti výsledok ako návratovú hodnotu. Takúto funkciu vieme použiť na ľubovoľné premenné alebo hodnoty, stačí ich zadať ako argumenty funkcie:

```c
#include <stdio.h>

int spocitaj(int a,int b){
    return a + b;
}

int main(){
    spocitaj(2,3);
    return 0;
}
```

Ak si vyskúšame takýto program, zistíme, že nie je veľmi užitočný. Napriek tomu, že vykonáme operáciu spočítania, výsledok sa nedostaví. Je to z toho dôvodu, že sme na to nedali príkaz. Program poslušne vykoná to, o čo sme ho požiadali, ale nič viac.

Na zobrazenie hodnoty premennej je potrebné použiť funkciu `printf()`. Ako jej druhý argument uvedieme názov premennej, ktorej obsah sa má vypísať. Dávajme ale pozor, aby formátovacia značka v prvom argumente sedela s typom premennej:

```c
#include <stdio.h>

int spocitaj(int a,int b){
    return a + b;
}

int main(){
    int vysledok = spocitaj(2,3);
    printf("Vysledok spocitania 2 + 3 je %d\n",vysledok);
    return 0;
}
```

Tento program je oveľa užitočnejší. Najprv si definujeme premennú `vysledok` a do nej si uložíme návratovú hodnotu funkcie `spocitaj`. Funkcia `printf` spôsobí výpis správy na obrazovku ako vedľajší efekt vykonania. Návratová hodnota funkcie `printf` nás nezaujíma, preto ju ignorujeme. Návratová hodnota celého programu bude *0*, čo je signál operačnému systému, že je všetko v poriadku.

Keďže už vieme definovať vlastné funkcie s argumentami aj pracovať s návratovými hodnotami, môžeme si definovať funkciu, ktorá spočíva ľubovoľné dve čísla a výsledok vypíše na obrazovku. Využijeme kód, ktorý už ovládame:

```c
#include <stdio.h>

int spocitaj(int a,int b){ ①
    return a + b;
}

void vypis_sucet(int a,int b){ ②
    int vysledok = spocitaj(a,b);
    printf("Vysledok spocitania %d + %d je %d\n",a,b,vysledok);
}

int main(){ ③
    vypis_sucet(4,5);
    return 0;
}
```

① Argumenty funkcie sú dve celé čísla. Návratové hodnota funkcie `spocitaj` je celé číslo, ktoré je súčtom argumentov. Funkcia nemá vedľajšie efekty.

② Argumenty funkcie `vypis_sucet` sú dve celé čísla. Funkcia má prázdny návratový typ (nemusí mať `return`). Jej vedľajším efektom je výpis na obrazovku.

③ Funkcia `main` nemá argumenty a volá ju operačný systém pri spustení programu. Má celočíselný návratový typ, ktorého hodnotu prevezme operačný systém.

Úloha na precvičenie

1. Aký bude výsledok volania funkcie `vypis_sucet(4.5,4.5)` ?
2. Aká bude návratová hodnota a čo sa vypíše na obrazovku?

Hoď to do stroja - Jednoduchý Vstup z klávesnice a podmienky

Naučíte sa

- dátový typ reťazec
- parametre funkcie printf a sscanf
- Výpis čísla vlastne znamená jeho premenu na reťazec.
- Používať funkciu fgets a sscanf.
- čítať dokumentáciu funkcie fgets a sscanf.

V predošlom tutoráli sme sa naučili vytvoriť funkciu na spočítanie dvoch čísel a na výpis výsledku. V tomto bloku sa naučíme, ako vytvoriť interaktívny program, ktorý bude schopný požiadať používateľa o vstup, načítať zadanú hodnotu do premennej a vypísať výsledok. Naučíme sa aj bojovať s nezodpovednými používateľmi, ktorí testujú našu pozornosť a trpezlivosť a zadávajú niečo iné ako očakávame. Výsledkom bude kalkulačka, vhodná aj pre malé deti.

Načítanie z klávesnice

Doteraz vytvorený program už je použiteľný ako jednoduchá kalkulačka, ale iba pre nás programátorov. Tento postup naozaj nie je vhodný pre každého. Bolo by fajn, keby s našou kalkulačkou vedela pracovať aj moja babka.

Do teraz sme počiatočné hodnoty premenných určovali priamo v programe. Na to aby sme zmenili čísla na spočítanie, musíme zmeniť náš program na správnom mieste a zopakovať proces prekladu a spustenia. Slovo premenná je od slova *meniť* a preto skúsime zmeniť náš program tak, aby sa premenná vedela meniť počas behu. Požiadame používateľa o vstup krátkou správou, v ktorej mu vysvetlíme, čo od neho chceme:

Žiadosť o vstup

```c
#include <stdio.h>

int main(){
    printf("Súčtová kalkulačka\n");
    printf("Prosím zadajte prvý argument\n");
    return 0;
}
```

Vstup od používateľa sa v jazyku C dá riešiť rôznymi spôsobmi. Najjednoduchším (a zároveň najhorším [1: vysvetlíme neskôr]) spôsobom je využitie funkcie `scanf`. Zhodou okolností (ale nie náhodou) je použitie funkcie `scanf` veľmi podobné použitiu funkcie `printf`. Ako prvý argument šablónu reťazca ktorý očakávame a druhý argument je **adresa premennej**, kde sa má uložiť výsledok. Adresu premennej získame pomocou operátora `&`.

Vstup celočíselnej premennej

```
int vstup = 0;
scanf("%d",&vstup);①
```

① Druhý argument funkcie `scanf` očakáva adresu, preto musíme použiť operátor `&` na získanie adresy premennej

Takto si definujeme celočíselnú premennú `vstup`, do ktorej si uložíme počiatočnú hodnotu nula. V druhom príkaze voláme funkciu `scanf`, ktorá spôsobí to, že program čaká na celočíselný vstup od používateľa z klávesnice a ak je to možné, tak výsledok uloží na zadanú adresu premennej. Celý program s kalkulačkou môže vyzerať takto:

```
#include <stdio.h>

int spocitaj(int a,int b){
    return a + b;
}

void vypis_sucet(int a,int b){
    int vysledok = spocitaj(a,b);
    printf("Vysledok spocitania %d + %d je %d\n",a,b,vysledok);
}

int main(){
    printf("Súčtová kalkulačka\n");
    printf("Prosím zadajte prvý argument\n");
    int a = 0;
    scanf("%d",&a);
    printf("Prosím zadajte druhý argument\n");
    int b = 0;
    scanf("%d",&b);
    vypis_sucet(a,b);
    return 0;
}
```

Úloha na precvičenie

Preložte program, spustite ho a dajte vyskúšať Vašej babke alebo sedemročnému dieťaťu. Za akých podmienok program funguje správne a kedy nastáva zlyhanie?

Ošetrenie vstupu

Po preskúšaní programu sme zistili, že program síce funguje správne, ale iba v prípadoch keď zadaný vstup vyhovuje formátovacej značke `%d`, teda celé číslo. Ak používateľ zadá niečo iné, výsledky nie sú podľa očakávania. Jedným z možných riešení je používateľa na túto skutočnosť dopredu upozorniť:

```
printf("Súčtová kalkulačka celých číslek\n");
printf("Prosím zadajte prvý argument (ak nezadáte celé číslo, tak sa kalkulačka pokazí
)\n");
int a;
scanf("%d",&a);
```

Kalkulačka by sa mala správať tak, ako to používateľ očakáva, inak vznikne nespokojnosť na strane spotrebiteľa, môjho šéfa a v konečnom dôsledku aj moja nespokojnosť. Ani by som nebol veľmi šťastný, keby som si takúto kalkulačku kúpil. Pripomína mi to prístup niektorých úradov k oprave ciest. Namiesto vyasfaltovania dier pribudne značka s chrbátom dvojhrbej ťavy alebo s obrázkom uja s lopatou.

Ťava podľa http://ascii.co.uk/art/camel

```
               ,,_
               (=-'
          /\/\  ))
        ~/    \/ |
        | )___(  |
        |/     \||
  ejm98 |'      |'
```

Skúsme urobiť kalkulačku odolnejšiu voči neočakávanému vstupu a overiť, či načítanie prebehlo úspešne. Aby sme to vedeli urobiť, musíme si bližšie prečítať dokumentáciu funkcie `scanf`. Zistíme, že návratová hodnota funkcie `scanf` nie je až taká nezaujímavá ako v prípade funkcie `printf`.

Technickú dokumentáciu funkcie `scanf`. zobrazíme príkazom `man scanf` alebo na internete.

Funkcia `scanf` vráti počet úspešne načítaných hodnôt, čo je v našom prípade 1. Túto skutočnosť môžeme využiť na to aby sme našu kalkulačku urobili odolnejšou voči neočakávanému vstupu. Ak používateľ nezadal správny reťazec, môžeme ho na túto skutočnosť upozorniť. Zapamätáme si návratovú hodnotu a môžeme ju vypísať:

```
int a = 0;
int pocet_hodnot = scanf("%d",&a);
printf("Pocet uspesne nacitanych hodnot je %d\n",pocet_hodnot);
printf("Nacitana hodnota je %d\n",a);
```

Vieme síce zistiť, či nastal nesprávny vstup, ale nevieme s tým nič robiť. Aby sme vedeli ošetriť nesprávny vstup od používateľa, musíme napísať kód, ktorý sa vykoná iba v prípade, že používateľ zadal nesprávny vstup. Na to použijeme podmienku **if**.

Úloha na precvičenie

Čo sa stane ak namiesto celého čísla zadám reťazec alebo číslo s desatinnou čiarkou?

Podmienka if

Jazyk C nám umožňuje napísať taký kód, ktorý sa spustí iba v prípade, že je splnená určitá podmienka. Používame na to kľúčové slovíčko **if** za ktorým do okrúhlych zátvoriek napíšeme podmienku. Nasledujúci príkaz alebo blok príkazov sa vykoná iba v prípade, že je podmienka pravdivá. Podmienka v zátvorke je pravdivá práve vtedy keď je nenulová. Tento kód ilustruje vytvorenie bloku kódu pomocou podmienky **if**, ktorý sa vykoná vždy:

```
if (2) {
    printf("Vykonam sa vzdy\n");
}
```

Ak je výraz v zátvorke nulový, podmienka sa nevykoná. Nasledovný blok kódu v podmienke **if** sa nevykoná nikdy:

```
if (0) {
    printf("Nevykonam sa nikdy\n");
}
```

Namiesto hodnoty môžeme do zátvorky napísať výraz, ktorý sa vyhodnotí na nulovú alebo nenulovú hodnotu:

```
if (1 == 2) {
    printf("Nevykonam sa lebo 1 sa nerovna 2\n");
}
```

Podmienka **if** môže byť nasledovaná blokom **else**, ktorý sa vykoná iba v prípade, že podmienka nie je splnená:

```
if (1 == 2) {
    printf("Nevykonam sa lebo 1 sa nerovna 2\n");
}
else {
    printf("Vzdy viem ze 1 sa nerovna 2\n");
}
```

Operátor `==` znamená operáciu porovnania, ktorá ak je pravdivá tak vracia nenulovú hodnotu a ak je nepravdivá, tak vracia nulu. Operácia `!=` (nerovná sa) sa správa opačne ako operácia `==`, vráti *1* v prípade, že dve hodnoty nie sú rovnaké a nulu v iných prípadoch.

Operátor porovnania a operátor priradenia

Operácia priradenia vždy vracia nenulovú hodnotu a ako vedľajší efekt spôsobí kopírovanie do premennej na ľavej strane. Nasledovný kód sa teda nebude správať podľa očakávania:

```
if (pocet_hodnot = 1){ ①
    printf("Hodnota pocet_hodnot je 1\n",a);
}
```

① Výraz `pocet_hodnot = 1` vráti nenulovú hodnotu a jeho vedľajší efekt je priradenie hodnoty do premennej.

Pozor, v jazyku C je operátor porovnania `==` rozdielny ako operátor priradenia `=`.

Napísaná správa síce bude pravdivá, ale spôsobí to, že blok kódu v podmienke sa vykoná vždy a prepíše sa hodnota premennej. Takto napísaný kód je nesprávny. Oveľa lepšie je vždy v podmienke if využívať operáciu porovnania:

```
if (pocet_hodnot == 1){
    printf("Hodnota pocet_hodnot je 1\n",a);
}
```

Úlohy na precvičenie:

Vykoná sa nasledovný blok kódu?

```
if (-1) {
    printf("Je -1 pravdivy vyraz?\n");
}
```

Čo vypíše takéto volanie funkcie `printf`?

```
printf("Hodnota pravdiveho vyrazu je %d",1==1);
```

Vstup s ošetrením

Už vieme napísať kód, ktorý sa vykoná iba ak je vstup od používateľa nesprávny. Operátor porovnania môže využívať aj premenné, takže môžeme napísať:

```c
#include <stdio.h>

int main(){
    printf("Súčtová kalkulačka\n");
    printf("Prosím zadajte prvý argument\n");
    int a = 0;
    // Nacita hodnotu pod pouzivatela do premennej a
    // a vrati pocet nacitanych hodnot do pocet_hodnot
    int pocet_hodnot = scanf("%d",&a);
    printf("Pocet uspesne nacitanych hodnot je %d\n",pocet_hodnot);
    if (pocet_hodnot != 1){
        printf("")
    }
    printf("Nacitana hodnota je %d\n",a);
    return 0;
}
```

V tomto príklade vieme vykonať nejaký blok príkazov iba v prípade, že bola zadaná nesprávna hodnota. Celý príklad s ošetrením vstupu bude trochu zložitejší:

```c
#include <stdio.h>

int spocitaj(int a,int b){
    return a + b;
}

void vypis_sucet(int a,int b){
    int vysledok = spocitaj(a,b);
    printf("Vysledok spocitania %d + %d je %d\n",a,b,vysledok);
}

int main(){
    printf("Súčtová kalkulačka\n");
    printf("Prosím zadajte prvý argument\n");
    int a = 0;
    int pocet_hodnot = 0;
    pocet_hodnot = scanf("%d",&a);
    if (pocet_hodnot == 1){
        printf("Prosím zadajte druhý argument\n");
        int b = 0;
        int pocet_hodnot = 0;
        pocet_hodnot = scanf("%d",&b);
        if (pocet_hodnot == 1){
            vypis_sucet(a,b);
        }
        else {
            printf("Zle ste zadali druhy argument. Cakal som cele cislo.\n");
        }
    }
    else {
        printf("Zle ste zadali prvy argument. Cakal som cele cislo.\n");
    }
    return 0;
}
```

Tento program má o niečo lepšie vlastnosti. Pomocou vetvenia **if-else** sme dosiahli, že funkcia na sčítanie sa bude volať iba v prípade, že obidve hodnoty boli zadané správne. Zabráni sa "pokazeniu" kalkulačky - vylúčili sme situáciu, keď bude program fungovať, ale nesprávne. Vďaka tomu si môžeme byť istý, že ak dostaneme výsledok tak bude správny.

Úlohy na precvičenie

Upravte kalkulačku tak, aby vedela pracovať aj s číslami s desatinnou čiarkou.

Bicyklová reťaz: cykly a reťazce

Pomocou našich zázračných programátorských schopností sme boli schopní vo veľmi krátkom čase navrhnúť a implementovať kalkulačku vhodnú aj pre netechnické typy. Stále tam však zostávajú viaceré nedostatky. V prípade, že zadáme nesprávny vstup, musíme celý proces začať od znova.

Cyklus while

V tejto kapitole upravíme kalkulačku tak, aby sa nevzdávala pri prvom neúspechu, ale vytrvalo prosila o správny vstup až kým sa to nepodarí. Na to využijeme cyklus typu **while**, ktorý bude prebiehať dovtedy, pokiaľ je splnená podmienka:

```
while(1){
    printf("Ja neprestanem\n");
}
```

Zápis cyklu typu **while** je veľmi podobný podmienke **if**, hlavný rozdiel je v tom, že telo cyklu môže byť vykonané viac krát (pokiaľ platí podmienka). Podmienka je splnená ak je vyhodnotená na nenulovú hodnotu, takže takýto cyklus bude prebiehať do nekonečna (alebo pokiaľ nezrušíme celý proces).

Ak vieme napísať cyklus, nebude problém modifikovať program tak, aby požiadavka na vstup prebiehala dovtedy, pokiaľ nebude vstup správny. Inými slovami, ak bude vstup nesprávny tak sa bude požiadavka opakovať. Aby nám cyklus prebehol aspoň raz, nastavíme počiatočnú hodnotu premennej `pocet_hodnot` na nulovú hodnotu.

```c
#include <stdio.h>

int spocitaj(int a,int b){
    return a + b;
}

void vypis_sucet(int a,int b){
    int vysledok = spocitaj(a,b);
    printf("Vysledok spocitania %d + %d je %d\n",a,b,vysledok);
}

int main(){
    printf("Súčtová kalkulačka\n");
    int pocet_hodnot = 0;
    int a = 0;
    while (pocet_hodnot != 1){
        printf("Prosím zadajte prvý argument\n");
        pocet_hodnot = scanf("%d",&a);
    }
    int b = 0;
    pocet_hodnot = 0;
    while (pocet_hodnot != 1){
        printf("Prosím zadajte druhý argument\n");
        pocet_hodnot = scanf("%d",&b);
    }
    vypis_sucet(a,b);
    return 0;
}
```

Úloha na precvičenie

Vyskúšajte si tento program a skúste zistiť, prečo sa program v prípade nesprávneho vstupu správa nečakane. Modifikujte tento program tak, aby v prípade nesprávneho vstupu o tom vypísal správu.

Načítanie reťazca

Zistili sme, že takto napísaný program síce funguje skvele, ale iba v prípade že sa nevyskytnú "neočakávané" okolnosti (nesprávny vstup od používateľa). Dôvodom je to, že funkcia `scanf` nie je veľmi užitočná. Obsahuje vnútorný buffer (pomocné pole znakov), ktorý sa vyprázdni iba v prípade, že vstup z klávesnice bol správny. V prípade, že bol vstup nesprávny tak tam nesprávna hodnota ostane, až pokiaľ nebude spracovaná. Ďalšie volania funkcie `scanf` potom namiesto vstupu od používateľa stále pracujú s pôvodným, nesprávnym vstupom.

Vyplýva z toho, že funkcia `scanf` je použiteľná iba na veľmi jednoduché príklady, ale nie je vhodná na reálne použitie. Našťastie, riešenie je pomerne jednoduché - naprogramovať si vlastný "buffer", do ktorého budeme ukladať vstup od používateľa. Premenu vstupu z klávesnice na číslo budeme vykonávať osobitne na vlastnom buffri.

Zmena nášho programu bude taká, že namiesto celého čísla, ktoré je možné zapísať nesprávne budeme očakávať všeobecnejší typ **reťazec**. Do reťazca si poznbačíme všetko, čo používateľ zadal (ktoré klávesy stlačil) a premenu na celé číslo vykonaáme neskôr.

Reťazec je sada znakov ukončená nulou. Každý znak je v pamäti reprezentovaný jedným kódom (číslom v rozsahu 0 až 255). Posledným znakom je vždy nula ktorá vyznačuje koniec reťazca. Hodnoty znakov sú zakódované do číselnej podoby pomocou ASCII tabuľky.

Klasické celočíslené premenné v jazyku C umožňujú uloženie iba jednej hodnoty. Aby sme si mohli do premennej uložiť viac hodnôt, musíme o tom prekladaču povedať. Napríklad ak chceme "rozšíriť" celočíselnú premennú na viacero miest, môžeme napísať:

```
int pole[4];
```

Vyhradeniu viacerých pamäťových miest rovnakého typu vedľa seba vravíme **pole**. Celé si to vieme predstaviť ako rebrík:

Pole v pamäti

```
    int pole[4] = {4,3,2,1};


    index:    0     1       2       3
              +------+------+-------+-------+
    hodnota: |  4   |  3   |  2    |   1   |
              +------+------+-------+-------+
    adresa:  \#10   \#14   \#18    \#22
```

Rozdiel od klasického zápisu deklarácie premennej (príkazu na vyhradenie pamäťového miesta) je použitie hranatých zátvoriek. Pomocou hranatých zátvoriek v deklarácii premennej vravíme prekladaču, aby vyhradil viac miest naraz. Pole si môžeme zostaviť z ľubovoľného dátového typu, napr. `float`. Na uloženie znaku je v jazyku C najvhodnejší typ `char` (znakový typ). Keďže reťazec je pole znakov zakončené nulou, pamäť pre uchovanie desať znakového reťazca si vyhradíme takto:

```
char retazec[11];
```

Posledné políčko poľa musíme vyhradiť pre zápis nuly na konci.

Ak si chceme zapamätať ľubovoľný vstup od používateľa (menší ako *100* znakov) a uložiť ho do poľa, môžeme zapísať:

```
char retazec[100];
printf("Zadajte hocijaký vstup:");
fgets(retazec,100,stdin);
printf("Napísali ste: %s\n");
```

Prvým argumentom funkcie `fgets()` je názov poľa, kam sa má uložiť výsledok. Druhým argumentom je veľkosť poľa, ktoré máme k dispozícii. Tretím argumentom je súbor z ktorého sa má načítavať, v tomto prípade načítavame z klávesnice ako keby to bol otvorený súbor. Slabinou tohto programu je, že používateľ nesmie zadať viac ako *99* znakov, inak sa zvyšné znaky zapíšu mimo vyhradenej pamäte.

Reťazec v pamäti. Posledná hodnota s indexom 3 môže byť ľubovoľná.

```
    char reazec[4] = "14";

    index:    0     1       2       3
              +------+------+-------+-------+
    hodnota: |  '1' |  '4' |   0   |   x   |
              +------+------+-------+-------+
    adresa:  \#10   \#11   \#12    \#13
```

Konverzia reťazca na číslo

Zistili sme, že v jazyku C sú reťazec a číslo dve rozdielne veci. Zjednodušene môžme povedať, že reťazec je viac hodnôt vedľa seba, číslo je iba jediná hodnota. Na to aby sme so zadaným reťazcom vedeli vykonávať matematické operácie, musíme si ho premeniť na číslo. Tento krok do teraz robila za nás funkcia `scanf`, ktorá premieňala vstup z klávesnice vo forme reťazca bez toho, aby sme o nejakých reťazcoch tušili. Zistili sme ale, že má určité obmedzenia, ktoré je potrebné obísť.

Riešenie je našťastie jednoduché - konverziu na číslo je možné jednoduchú vykonávať nie len z klávesnice, ale aj z hodnôt, ktoré sme si poznačili pred tým do poľa. Slúži na to funkcia `sscanf`, ktorá sa správa rovnako ako funkcia `scanf`, ale namiesto klávesnice pracuje s reťazcom, ktorý jej zadáme ako argument. Netrpí teda problémom so zasekávajúcim sa buffrom.

Konverziu reťazca na číslo pomocou funkcie `sscanf` vykonávame takto:

```
int cislo = 0;
int pocet_hodnot = 0;
char retazec[10] = "12";
pocet_hodnot = sscanf(retazec,"%d",&cislo);
printf("Vase cislo je %d",);
```

Číslo v pamäti

```
int cislo = 14;

              +------+
    hodnota: | 14   |
              +------+
    adresa:  \#10   \#14
```

Celý program

Celý program bude potom vyzerať takto:

```c
#include <stdio.h>

int spocitaj(int a,int b){
    return a + b;
}

void vypis_sucet(int a,int b){
    int vysledok = spocitaj(a,b);
    printf("Vysledok spocitania %d + %d je %d\n",a,b,vysledok);
}

int main(){
    printf("Súčtová kalkulačka\n");
    int pocet_hodnot = 0;
    int a = 0;
    char buffer[100];
    while (pocet_hodnot != 1){
        printf("Prosím zadajte prvý argument\n");
        fgets(retazec,100,stdin);
        pocet_hodnot = sscanf(buffer,"%d",&a);
    }
    int b = 0;
    pocet_hodnot = 0;
    while (pocet_hodnot != 1){
        printf("Prosím zadajte druhý argument\n");
        fgets(retazec,100,stdin);
        pocet_hodnot = sscanf(buffer,"%d",&b);
    }
    vypis_sucet(a,b);
    return 0;
}
```