Jazyk C

Funkce

- dělí se na
  - standardní funkce - gets(), scanf(), sqrt(), ...
  - funkce main - hlavní funkce
  - vlastní funkce - (lokální a globální proměnné)
- **Popis a použití**
  - jsou to bloky kódu, které můžou být volany s různých částí programu
  - funkce umožňují organizovat kód do menších částí
  - funkce umožňují opakovaně používat určité úseky kódu
  - funkce umožňují snadnější orientaci v kódu
- **Definice, tělo, prototyp**
  - **Prototyp**
    - musí být před funkcí main
    - obsahuje
      - název funkce a typ návratové hodnoty
      - názvy parametrů a jejich datové typy
    - říká kompilátoru, že funkce existuje, ale neřeší její obsah
  - **Definice**
    - za funkcí main
    - obsahuje tělo funkce, tedy to, co se má stát, kdy funkci zavoláš
  - **Tělo**
    - celý kód uvnitř {}, který se provede při zavolání funkce
- **Bez parametru, s parametrem**
  - bez parametru - void sort**()**;
  - s parametrem - void searchByID**(int id)**;
  - s návratovou hodnotou - **int** getID();

Struktura a ukazatelé

- Efektivní manipulace s daty a alokací paměti

- Struktura - "Balík proměnných"
  - je to vlastní datový typ, který seskupuje různé proměnné do jednoho objektu
  - k jednotlivým položkám se dostanu přes tečku - *mojeAuto.rok* ...
- Ukazatele - "Šipky na paměť"
  - je proměnná, která obsahuje adresu jiné proměnné
    - místo hodnoty uchovává "šipku" na místo v paměti

```
printf("Hodnota: %d\n", *ptr); // Dereference (vypíše 10)
printf("Adresa: %p\n", ptr); // Vypíše adresu v paměti
```

Práce se soubory

- umožňuje čtení a zápis dat ze souborů na disku
- **Typy souborů**
  - Textové
    - ukládají data jako čitelný text (.txt, .csv)
    - možnost otevřít v poznámkovém bloku, nebo editoru třeba
    - "r" - čtení
    - "w" - přepsání souboru
    - "a" - přidání na konec souboru
  - Binární
    - ukládají data v surové binární (paměťové) podobě (.dat, .bin)
    - jsou rychlejší a zabírají méně místa
    - "rb" - čtení
    - "wb" - přepsání souboru
    - "ab" - přidání na konec souboru
- **Deklarace proměnných pro práci se souborem**
  - je třeba deklarovat proměnnou typu FILE - představuje ukazatel na soubor

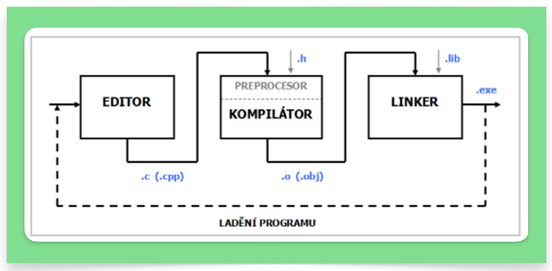
    FILE *f;
- **Přístupové funkce**
  - fopen() - otevření souboru
  - fclose() - uzavření souboru
  - fread() - čtení ze souboru
  - fwrite() - zápis do souboru

Knihovny

- obsahují předdefinované funkce, které můžeme použít v programu
- můžeme je taky nainstalovat
- musí mít příponu .h
- pro možnost používání v kódu je musíme includovat - #include <stdio.h>
- pro includování vlastní funkce používáme místo ostrých závorek uvozovky - #include "funkce.h"
- různé knihovny
  - stdio.h
  - stdlib.h
  - math.h
  - string.h
  - stdbool.h
  - time.h
  - ...

MAIN INFO

- starý ale stále hodně používaný programovací jazyk
- 70.léta - Dennis Ritchie a Ken Thompson
  - Byl vytvořen hlavně proto aby pomohl s vývojem operačního systému UNIX.
- postupně se rošířil a dnes se s ním programují nejen operační systémy ale i Embedded systémy (domací spotřebiče) a superpočítače či aplikace
- je to kompilační jazyk - vyžaduje kompilátor (překladač)



Vývojové prostření Dev C++, compiler a linker

- Dev C++ - Editor
  - je jednoduché (IDE) integrované vývojové prostředí určené pro programování C/C++
  - umožňuje psání, kompilaci a ladění kódu
  - je to program v jazyce C/C++, tzn. zdrojový kód, který můžeme napsat v jednoduchém textovém souboru
  - výstupem je soubor s příponou .c/.cpp
- Kompilátor - překladač (překlad do assembleru)
  - má za úkol převést zdrojový kód na spustitelný program, kterému rozumí procesor
  - překládá kód do binární podoby, který procesor dokáže vykonávat
  - vznikne 'relativní kód' s příponou .o/.obj - binární instrukce
- Linker - Sestavovací program
  - propojí 'relativní kód' s knihovnami (.lib)
  - výsledkem je spustitelný soubor (ve Windows .exe)

Pokud spustitelný program nefunguje podle naších představ je třeba přepsat kód - ladění
- k tomu je možné využít debugger

Editor, překladač a linker jsou většinou přímo součástí vývojového prostředí
- VS Code, NetBeans, ...

Datové typy

- určují jaký druh hodnoty může proměnná obsahovat a kolik paměti na ní bude potřeba
  - prostě, když vytvoříme proměnnou musíme určit jestli bude obsahovat
  - tomu se říka INICIALIZACE proměnné
    - čísla - int - celá čísla (24, -8, 1234)
    - znaky - char - jeden znak ('A', '*', '7'), char[] nebo string - řetězec znaků ("Hello")
    - destinné hodnoty - float, double (přesnější)
    - logické hodnoty - bool - true (1), false (0)
- **Definice**
  - říkáš programu "Tady je místo v paměti pro nějakou hodnotu", ale zatím do ní nic neukládáš
  - řekneš, že proměnná existuje, ale zatím do ní nic neukládáš
  - př. - int cislo;
- **Inicializace**
  - proměnnou vytvořím a rovnou jí přidám hodnotu
  - př. - int cislo = 24;

Proměnná - místo v paměti, jeho obsah se může měnit když něco děláme

Přetypování - Type Casting

- znamená, že změníš datový typ proměnné na jiný
- např. float na int
- existují i předepsané funkce pro změnu typu proměnné
  - atoi() - text na celé číslo
  - atof() - text na des. číslo (float)
  - strtod() - text na des. číslo (double)
- jak?

  když *a* a *b* jsou typu int

  float vysledek = (float)a / b;
  - říkám "ať se *a* chová jako float"

  když ale *a* je typu float a *a = 9.99*

  int vysledek = (int)a;
  - vysledek bude 9 (desetinná část se zahodí)

Existuje implicitní a explicitní
- Implicitní (automatické) - převadíme DT s nižší prioritou na DT s vyšší prioritou
  - třeba s int na float

    když *a* je int (*a=3*) a *b* je float (*b=1.123*)

    float cislo = a + b; //a se převede na 3.000
- Explicitní (vynucené) - převádíme DT s vyšší prioritou na DT s nižší prioritou
  - je třeba dávat pozor na ztrátu dat

    když je *a* float (*a=1.123*) a *b* je int

    int b = (int)a; //do proměnné se uloží oříznuté a, tj. 5

Standardní výstupy a vstupy

- to zajišťuje knihovna stdio.h
- scanf()
  - načte hodnoty z klávesnice

    scanf("%d", &a); // *&* - protože přistuúujeme k adrese proměnné
    // *%a* - protože se jedná a int
- printf()
  - vypis hodnot z konzole
  - pro zobrazení proměnné - použijeme *%d*, a pak na konci za čárkou jméno proměnné
    - *printf("%d ... ", cislo);*

- pro řetězce je dobré používat
  - gets() - načtení řetězce
  - puts() - vypis řetězce
  - z knihovny string.h

- formatovací znaky
  - \n - odřádkování
  - \t - tabulátor

Řídící struktury

- určují tok programu
- používají
  - logické operátory (< | > | <= | >= | == | != | && | II | !)
  - operátory inkrementace a dekrementace (**i++** | **i--**)
  - ternární operátor (***if(i<5) ? x:y***) - pokud ano vrácí hodnotu *x*, jinak *y*

Větvení

- if, else, else if
  - pokud je podmínka v závorkách platná provede se kód v {}
  - když ale pridáme za } else - provede se v případě nepravdivé podmínky v závorkách kód v elsu
- switch

```
switch(volba){
case 1: printf(„a“);break;
case 2: printf(„b“);break;
default: printf(„0“);break;
}
```

  - pokud se volba v závorkách bude rovnat některému z caseu - provede se ten daný case
  - jinak se provede default

Cykly

- Cykl/Smyčka se opakuje dokud je podmínka pravdivá
  - for - *for(int i=0;i<n;i++){...}*
    - kód se bude provádět dokud *i* bude menší než *n*
  - while - *while(a<b){...}*
    - kód se bude provádět dokud bude *a* menší než *b*
  - do while - *do{...}while(a<b)*
    - smyčka provede poprvé kód bez toho aníž by pohlížela na podmínku
    - pak pojede dokud bude *a* menší než *b*

- příkazy:
  - *break* - vyskočí ze smyčky
  - *continue* - přeskočí na další cyklus

Práce s polem

- Pole je druh proměnné, ve které se nachází více než 1 hodnota stejného datového typu
- deklarace - do hranatých závorek za název proměnné se vloží jeho velikost
  - ***int a[5];***
  - nebo - ***int a[5] = {2,3,4,5,6};*** - když při deklaraci chceme naplnit pole hodnotami
- pro zápis nebo výpis pole potřebujeme smyčku s indexem - pole začíná od 0
- můžeme také přistoupit a přepsat přímo hodnotu na daném indexu
  - ***a[3]=5;*** //na čtvrté pozici v poli se přepiše hodnota na pětku

Jenorozměrné pole
- pracujeme s ním pomocí jednoho indexu (pozice)

```
int a[5] = {2,3,4,5,6};
```

Dvojrozměrné pole
- pracujeme s ním pomocí dvou indexů (souřadnice)
- můžeme si představit jako tabulku (matici)
  - každý 'prvek' má dva indexy
    - první index - řádek | druhý index - sloupec

```
int a[3][3] = {
        '1','2','3',
        '4','5','6',
        '7','8','9'
    };
```

- při procházení se musí použít 2 smyčky
  - jedna pro řádek, druhá pro sloupec

Může být i 3-rozměrné pole - ***int a[3][3][3] = {...};***

```
float cisla[100], pom, min;
int i, pocet, imin, zac;

//uvod programu
//nacteni hodnot do pole, urceni aktualniho poctu prvku
//pripadne dalsi prikazy

for ( zac = 0; zac < pocet - 1; zac++)          //posun zacatku
{
   min = cisla[zac];       //zacatek algoritmu pro urceni minimalni hodnoty
   imin = zac;
   for ( i = zac; i < pocet; i++)
          {
       if ( cisla[i] < min )
       {
            min = cisla[i];
            imin = i;
       }
    }
   pom = cisla[zac];                            //zamena s prvnim prvkem
   cisla[zac] = cisla[imin];
   cisla[imin] = pom;
}
```

Algoritmy třídění pole

- algoritmy, které uspořádájí prvky pole do určitého pořadí
- Bubble sort
  - řazení záměnou
  - jednoduchý algoritmus, který porovnává 2 sousedící hodnoty
    - po každém průchodu se největší hodnota dostane na konec
- Selection sort
  - řazení výběrem minima
    - Na začátku si nastavíme min hodnotu na první hodnotu v poli a pak procházíme pole. Když najdeme novou min hodnotu uložíme si její index a přepíšeme min hodnotu. Po doběhnnutí smyčky zaměníme počáteční hodnotu s naší nalezenou minimální.
  - řazení výběrem maxima
    - Zde je postup obrácen jako je popsáno v předchozím kroku pro lepší představení.

```
float cisla[100], pom, max;
int i, pocet, imax, kon;

//uvod programu
//nacteni hodnot do pole, urceni aktualniho poctu prvku
//pripadne dalsi prikazy

for ( kon = pocet - 1; kon > 0; kon-- )      //posun konce
{
   max = cisla[0];     //zacatek algoritmu pro urceni maximalni hodnoty
   imax = 0;
   for ( i = 0; i < kon + 1; i++)
          {
       if ( cisla[i] > max )
       {
          max = cisla[i];
          imax = i;
       }
     }
   pom = cisla[kon];       //zamena s poslednim prvkem
   cisla[kon] = cisla[imax];
   cisla[imax] = pom;
}

//pokracovani programu
```