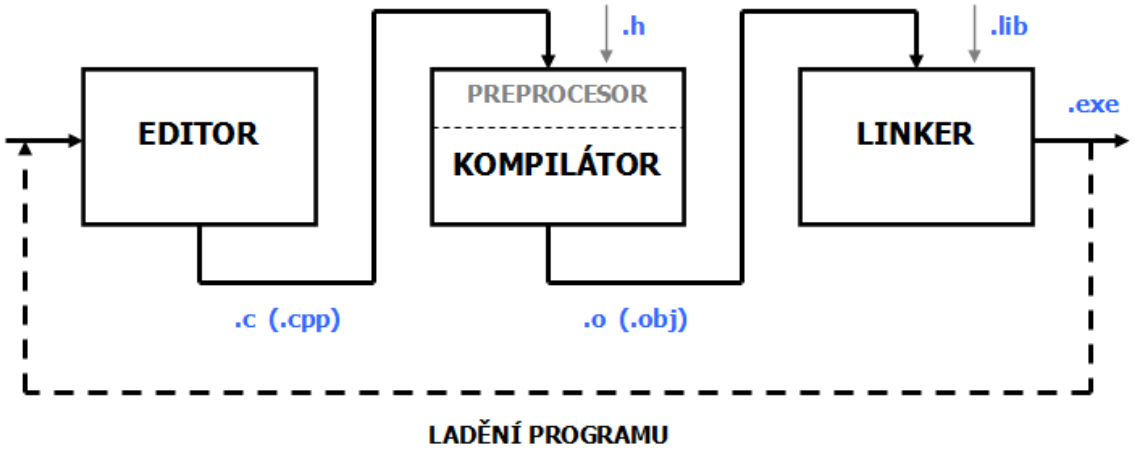
PROGRAMOVACÍ JAZYK C

- **Vývojové prostředí Dev C++, compiler a linker**
- **Datové typy**
  - Definice vs. Inicializace
- **Přetypování**
- **Standartní vstupy a výstupy**
- **Řídící struktury**
  - Větvení
  - Cykly
- **Práce s polem**
  - Jednorozměrné, dvourozměrné
  - Načtení, výpis, manipulace s prvky
- **Algoritmy třídění pole**
- **Funkce**
  - Popis a použití
  - Definice, tělo, prototyp
  - Bez parametrů, s parametrem
- **Struktury a ukazatele**
- **Práce se soubory**
  - Typy souborů
  - Deklarace proměnných pro práci se souborem
  - Přístupové funkce
- **Knihovny**

1. Vývojové prostředí Dev C++, compiler a linker



EDITOR - program v jazyce C, tzv. zdrojový kód, můžeme napsat v jednoduchém textovém editoru. Jeho výstupem je soubor s příponou. c (případně.cpp). Je vhodné nepoužívat v názvech souborů české znaky a mezery.

KOMPILÁTOR - PŘEKLADAČ - textový soubor nejdřív zpracuje preprocesor, který zdrojový kód částečně upraví. Vloží hlavičkové soubory, odstraní komentáře, nahradí symbolické konstanty apod. Preprocesor je součástí překladače. Potom překladač přeloží příkazy, vznikne tzv. relativní kód (s příponou.obj). V případě, že se v programu vyskytnou formální chyby, je nutné je opravit a kód znovu zkompilovat.

LINKER - SESTAVOVACÍ PROGRAM - poslední fází je propojení relativního kódu a knihovních souborů (.lib). Výsledkem je spustitelný soubor (ve Windows s příponou.exe).

Pokud spustitelný program nefunguje podle našich představ - dává např. špatné výsledky, musíme program přepsat. Procesu hledání logických chyb říkáme ladění programu. K tomu je možné využít ladící program tzv. debugger.

2. Datové typy

Datový typ označuje druh dat, proměnná je paměťový prostor. Při vytváření (definici) proměnné musíme stanovit její datový typ

Typ žádné hodnoty - (Void) - Podprogram nic nevrací

Název	Bitů	Význam	Příklad
char	8	celé číslo, znak	0, 255, 'a', 'A', 'e'
short[1)]	16	krátké celé číslo	65535, -32767
int	16/32	celé číslo	--||--
long[1)]	32	dlouhé celé číslo	--||--
long long[1)]	64	ještě delší celé číslo	9223372036854775807, -9223372036854775807
enum	8/16/32	výčtový typ	
float	32	racionální číslo	5.0, -5.0
double	64	racionální číslo s dvojitou přesností	5.0l, 5l, -5.0L
long double	80	velmi dlouhé racionální číslo	5.5L, 5l, -5.0l
pointer	16/32/64	ukazatel	

Inicializace vs. Deklarace

Pokud chceme nějakou proměnnou použít, musíme ji nejdříve deklarovat. Deklarace proměnné udává její název a datový typ. Je vhodné uvádět deklarace na začátku bloku příkazů. (Blok příkazů je posloupnost příkazů uzavřená v { }).

Příklady deklarace proměnných uvádějí různé možnosti, jak deklarovat proměnnou. Proměnná, která při deklaraci není inicializována t.j. nemá zároveň nastavenou hodnotu, obsahuje náhodnou hodnotu.

```
int x = 1, y;                    //celočíselná proměnná x s nastavenou hodnotou 1
                                 //celočíselná proměnná y s neurčenou hodnotou
char posl_znak = '*';            //proměnná posl_znak typu char (pro znaky)
float cislo1, cislo2 = 0.0;      //dvě reálné proměnné cislo1 a cislo2
                                 //proměnná cislo2 nastavena na 0, používáme desetinnou tečku
```

3. Přetypování

Přetypování je způsob, jak změnit interpretaci nějakého datového typu, ať již proměnné, či konstanty. Například, když dělíte dvě celá čísla, výsledkem je zase celé číslo. To se vám nemusí vždy hodit.

```
Double a = 5,3;
Int b;
b= (int) a;          //b==5
```

4. Standartní vstupy a výstupy

%c	znak
%d	celé desítkové číslo
%f	reálné číslo (float)
%s	řetězec

// výpis textu a odřádkovaní:

printf("Příklady výpisu hodnot na obrazovku: \n\n");

// jednoduchý výpis celočíselné proměnné:

printf("%d", x);

// výpis celočíselné proměnné s textem:

printf("Hodnota proměnné x je %d \n", x);

// výpis hodnot proměnné použitím vypočtu:

printf("Součet: %d + %d = %d \n", a, b, a+b);

// výpis reálné proměnné:

printf("Hodnota reálné proměnné y je %f \n", y);

// výpis reálné proměnné se zaokrouhlením na 1 desetinné místo:

printf("Hodnota reálné proměnné y na 1 desetinné místo je %.1f \n", y);

// výpis reálné proměnné - vhodný pro tabulkový výpis

printf("%10.2f \n", y);

Pro použití funkcí pro načtení a výpis musíme vložit hlavičkový soubor **stdio.h**

Pro načtení hodnot z klávesnice používáme funkce: **scanf();**

fscanf(); fgetc(); fgets();	//text soubor
gets(); scanf();	//řetězec
fread();	//bin soubor

Pro výpis hodnot na obrazovku používáme funkce: **printf();**

fprintf(); fputc(); fputs();	//text soubor
puts(); printf();	//řetězec
fwrite();	//bin soubor

5. Řídící struktury

- **Větvení (IF a SWITCH)**

if (log. výraz) příkaz_1; else příkaz_2;

```
switch(volba){                    //”volba” je celočíslená hodnota
        case 1: break;
        default:
}
```

- **Cykly se známým počtem průchodů**

For (počáteční_výraz; koncový výraz; iterace);

```
For(i=0;i<10;i++)
{
        Printf(“%d”, i);
}
```

- **Cykly s podmínkou na začátku**

While (výraz)
příkaz;

```
While(i<5)
i=i+1;
```

- **Cykly s podmínkou na konci**

Do{
Příkazy;
}while(výraz);

```
Do{
i=i+1;
}while(i<5);
```

6. Práce s polem

- Jednorozměrné a vícerozměrné pole
- Je to datová struktura obsahující prvky stejného datového typu
- Statická struktura, tzn je určen maximální počet prvků
- Při práci potřebujeme znát aktuální počet prvků
- Aktuální počet nesmí, přesáhnou maximální počet prvků
- Jazyk C rozsah nekontroluje, nutné aby zajistil programátor
- Pořadí prvků je dáno indexem, který začíná vždy od 0
- V paměti je pole uloženo jako souvislá oblast, velikost je určena rozsahem a datovým typem

```
Int pole[10];                 //pole muže obsahovat max. 10 celočíselných hodnot (0-9)
int pole[5]={1,2,3,4,5};      //deklarace s inicializací
Int pole[0]=1;                //na první index (0) uloží číslo 1
Int pole2[5][5];              //deklarace dvojrozměrného pole

scanf("%d", &pole[i]);        //jednoduché načtení pole
```

Pro naplnění pole používáme cyklus for:

```
Int pole[10], i;
for(i=0 ; i<10 ; i++)
{
   pole[i]= i+1;
}
```

Pro výpis celého pole používáme cyklus for pro procházení pole a pak jej vypíšeme

```
for(i=0 ; i<10 ; i++)
{
   printf("%d", pole[i];
}
```

7. Algoritmy třídění pole

Algoritmus pro hledání minimální hodnoty:

- Za minimum považujeme první prvek v poli
- Procházíme postupně hodnoty v poli, pokud je nějaký prvek menší než aktuální minimum, považujeme ho za nové minimum

```
float pole[100], min;
int i, n, imin;

min=pole[0];                    //nastavení počáteční hodnoty minima
imin=0;                         //nastavení indexu minima
for(i=1; i<n; i++)              //procházení hodnot v poli
if (pole[i]<min)                //když je hodnota z pole menší než aktuální minimum
{
min=pole[i];                    //přepíšeme hodnotu minima
imin=i;                         //nastavení indexu minima
}
printf("\n Minimální hodnota je %.2f, je to prvek s indexem %d\n", min, imin);
```

Algoritmus pro hledání maximální hodnoty:

```
float pole[100], max;
int i, n, imax;

max=pole[0];                    //nastavení počáteční hodnoty minima
imax=0;                         //nastavení indexu minima
for(i=1; i<n; i++)              //procházení hodnot v poli
{
  if (pole[i]>max)              //když je hodnota z pole menší než aktuální minimum
  {
    max=pole[i];                //přepíšeme hodnotu minima
    imax=i;                     //nastavení indexu minima
  }
}
printf("\n Maximální hodnota je %.2f, je to prvek s indexem %d\n", max, imax);
```

Bubble sort

- Bublinkové řazení
- Vhodný pro malé počty hodnot
- Založen na porovnávání sousedících hodnot
- Při každém průchodu „probublá“ maximální hodnota na konec posloupnosti
- Procházíme postupně hodnoty v poli a porovnáváme vždy dvě sousedící hodnoty, když je první z nich větší než druhá, hodnoty zaměníme a nastavíme, zda byla provedena záměna
- Opakujeme, dokud byla při posledním průchodu nastavena alespoň jedna záměna

```
float cisla[100], pom;
int i, pocet, zamena;

do
{
zamena=0;                       //nastaveni proměnné pro sledovaní záměny
                                //záměna není, na začátku každého průchodu polem
                                //procházení pole do předposledního prvku
 for (i = 0; i < pocet-1 ; i++)
{
   if ( cisla[i] > cisla[i+1] )
   {                            //záměna sousedících hodnot

        pom = cisla[i];
        cisla[i] = cisla[i+1];
        cisla[i+1] = pom;
        zamena=1;               //nastaveni proměnné pro sledovaní záměny
                                //záměna nastala

      }

    }

 }while(zamena == 1);
```

Selection sort

- vhodný pro malé množství dat
- data jsou rozdělena na seřazenou a neseřazenou část
- dvě varianty, výběrem maxima nebo minima
- najdeme prvek s nejmenší hodnotou v neseřazené části posloupnosti
- minimum zaměníme s prvkem na první pozici v neseřazené části
- posuneme hranici seřazené části a neseřazené části o 1 pozici vpravo
- opakováním předchozích kroků seřadíme celou posloupnost

Další třídící algoritmus např.: Quick sort

8. Funkce

Popis a použití

- Každý program obsahuje alespoň jednu funkci – main()
- Funkce jsou základním nástrojem pro strukturování programu
- Využíváme je, když se nějaká činnost v programu opakuje, když chceme složitý program rozdělit na dílčí úkoly, když potřebujeme opakovat nějakou činnost vícekrát v programu

Definice funkce

- Definice funkce obsahuje kompletní zápis funkce, tj hlavičku i tělo funkce
- Uvádí se za funkcí main();
- Pro název funkce platí stejná pravidla jako název proměnné

Prototyp funkce (deklarace)

- Obsahuje název funkce, typ návratové hodnoty a typy parametrů
- Končí středníkem a píše se na začátek programu před funkci main()

Návratová hodnota

- typ návratové hodnoty uvádíme před názvem funkce v její deklaraci a definici

```
typ nazev_fce(typ parametr1, typ parametr2,...)      //hlavička funkce
{
   typ prom;                                         //deklarace lokálních proměnných
   prikazy                                           //tělo funkce
   return hodnota;                                   //použijeme pro případ vraceni hodnoty
}
```

Parametry funkce

- uvádí se za názvem funkce v kulatých závorkách
- umožňují předávání hodnot mezi funkcí a ostatními částmi program
- funkce nemusí mít žádné parametry, potom se uvádí prázdné závorky

Rekurzivní funkce

- volají samy sebe
- je nutné dobře stanovit ukončovací podmínku – jinak hrozí přeplnění paměti a zatuhnutí programu

9. Struktura a ukazatelé

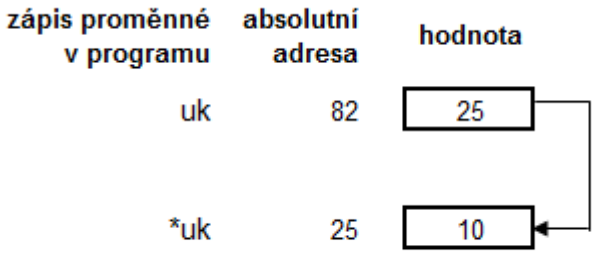
Struktura

- Používáme pro uložení údajů různých datových typů
- Všechny informace máme v jedné proměnné
- Tato definice se uvádí globálně, mimo jakoukoliv funkci na začátku programu
- K jednotlivým položkách se dostaneme přes operátor = tečka

```
Typedef struct{
   typ položka1;
} Tnazev;

Tnazev nazev_promenne;

Printf("nazev knihy: ");
gets(knihy[i].nazev);                    //načtení

Printf("%d %s", knihy[i].cislo, knihy[i].nazev);    //výpis
```

Ukazatel

- Dá se použít jako parametr ve funkcích, při práci s polem, při dynamické alokaci paměti
- Je to proměnná jako ostatní, pouze hodnota v ni uložená má jiný význam
- Deklarujeme ho jako ukazatel na určitý datová typ
- Obsahem ukazatele je adresa v paměti
- Při práci s ukazatelem nás ve většině případů zajímá hodnota uložená na dané adrese



- Proměnná uk je ukazatel, hodnota v něm uložená je 25, jedná se o adresu, na které je uložená hodnota, se kterou budeme pracovat, na absolutní adrese 25 je hodnota 18, ukazatel ukazuje na hodnotu 18, ale sám má hodnotu 25

Deklarace ukazatelů:

```
int *uk1, číslo=5;          //ukazatel na celé číslo
float *uk2;                 //ukazatel na reálnou hodnotu
FILE *f;                    //ukazatel pro práci se souborem

Uk=&číslo;
```

- ukazatel uk nastaví hodnotu na adresu proměnné číslo, pokud chceme pracovat s hodnotou, na kterou ukazatel ukazuje, použijeme v programu zápis *uk

Ukazatel jako parametr funkce

- Použijeme, pokud potřebujeme z funkce vrátit více výsledků, ty uvedeme jako parametry funkce
- Příklad: Do funkce jdou 2 hodnoty, ale z funkce potřebujeme dostat taky 2 hodnoty. Nepotřebujeme ale 4 parametry (2 pro vstup a 2 pro výstup), protože hodnoty, které vstupují do funkce, potřebujeme zároveň vrátit z funkce změněné.

```
#include <stdio.h>
  void zamena(int *x, int *y);                    //zamění hodnoty 2 proměnných
  int main()
  {
    int a,b;
    printf("\nZadejte dve cisla : ");
    scanf("%d%d",&a, &b);
    getchar();
    printf("\nPred zamenou : a = %d  b = %d\n", a, b);

    zamena(&a, &b);
    printf("Po zamene : a = %d  b = %d\n", a, b);

    getchar();
    return 0;
  }

  void zamena(int *x, int *y)
  {
    int pom;
    pom = *x;
    *x = *y;
    *y = pom;
  }
```

10. Práce se soubory

- Pokud chceme pracovat se souborem, musíme ho nejprve otevřít a po ukončení práce s ním je potřebné ho uzavřít

Typy souborů – textový a binární

- Textové soubory obsahují řádky textu, jejich obsah si můžeme prohlídnout, případně změnit v textovém editoru
- Binární soubor obsahuje informace zapsány binárně (zakódovaně)

Deklarace proměnných pro práci se souborem:

*FILE *soubor;*
*FILE *fw, *fr;*

Otevření a uzavření souborů

- Pro otevření souboru používáme funkci fopen();, má dva parametry – název souboru a režim pro práci s daty
- Soubor uzavřeme funkcí fclose();

Režimy otevření textového souboru:

- r= pro čtení
- w= pro zápis, pokud soubor neexistuje -> vytvoří se, jinak se data přepisují
- a= pro zápis na konec souboru, pokud neexistuje soubor -> vytvoří se

Režimy otevření binárního souboru:

- rb, wb, ab

Funkce pro práci s textovým souborem

- pro čtení slouží *fscanf();*
- pro zápis slouží *fprintf();*
- obě funkce mají tři parametry – proměnná typu FILE *, zbylé dvě jsou klasické
- pro čtení řetězce slouží *fgets();* a pro zápis *fputs();* nebo klasicky *fprintf();*
- cyklus pro čtení ze souboru až do jeho konce -> *while(fscanf(f,"%f",&x) != EOF)*

Funkce pro práci s binárním souborem

- pro čtení dat slouží funkce *fread();*
- pro zápis dat slouží funkce *fwrite();*
- obě funkce mají čtyři parametry – adresa proměnné, velikost bloku v Bytech (sizeof), počet položek, proměnná typu FILE *

fwrite(&x, sizeof(Tdata), 1, f);
fread(&x, sizeof(Tdata), 1, f);

11. Knihovny

- Základní součást programování
- Obsahují sadu funkcí a struktur
- Linker připojuje naši aplikaci ke knihovnám
- Na knihovny se musíme na začátku každé aplikaci nejdříve odvolat

#include<stdio.h>

- Standart input/output knihovna
- Printf,scanf,gets,getchar,fopen,

#include<stdlib.h>

- Standartní knihovna
- Rand, srand,systém („pause“), malloc, free, exit, atoi

#include<string.h>

- Knihovna pro práci s řetězcem
- Strcmp,strlen,strstr,strcpy

#include<math.h>

- Knihovna obsahující matematické operace
- Pow, sqrt, round, sin, exp, log

#include<time.h>

- Umožňuje zjistit aktuální čas
- Localtime, time