2. Programovací jazyk – jazykové konstrukce

Základní pojmy:

— **Datový typ** — je definován:
- předpis
  - jak
  - vyjádřit
  - pojmu
  - — **názvem**
  - — **množinou hodnot**
  - — **operacemi, které lze nad hodnotami provádět**
- — slouží pro vyhodnocení (deklarace a definice) proměnných
- — říká překladači:
  - — jak velká je paměťová buňka pro proměnnou
  - — jaké operace jde s proměnnou dělat

— **Proměnná** = Pojmenovaná paměťová buňka
- — má přiřazen datový typ
- — slouží k uchování hodnot
- — může se měnit v průběhu chodu programu

Inicializace proměnné
— **první přiřazení hodnoty** (při definici)
- — neinicializované hodnoty mají nedefinovanou hodnotu (smetí po předchozích proměnných)
- — proměnné některých typů lze jako celek nastavit jinou při inicializaci

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

Příklady deklarací
- — `int (*x)[]` – ukazatel na pole čísel typu int (nebo pole polí)
- — `int (*(*x)[])[]` – ukazatel na pole ukazatelů na pole typu int
- — `void (*f)(void)` – ukazatel na funkci bez parametrů vracející void
- — `void (*f[])(void)` – pole ukazatelů na funkce bez parametrů vracející void
- — v jazyce C nelze vracet pole!

Vyjádření složitějších typů
- — typedef specifikace typu nový-identifikátor;

```
int mojeFce(int a, int *out);
typedef int (*FUNC)(int, int *);
....
FUNC fce = mojeFci;
int x = fce(10, &y);
```

```
int *cislo; // neinicializovaný ukazatel na int
Tnode *ohnok; // ukazatel na hodnotu typu Tnode
FUNC fce = mojeFce; // ukazatel na funkci mojeFce
void *ptr; // netypový ukazatel
```

Deklarace proměnné (přísluší překladači)
> 90% proměnné spojení a s definicí!
- — syntaktická konstrukce
- — říká překladači: "Tam se ní, že bude existovat toho proměnná takto typu a jména"
- — např. deklarace parametrů funkce

`int mojeFce(FILE *a, int b);`

Definice Proměnné
- — příkaz, který vytvoří proměnnou v paměti
- — **Lokální** ⇒ uvnitř funkce (blokové příkazy), na zásobníku, zaniká na konci funkce (bloku)
- — **Globální** ⇒ mimo funkce, v datové oblasti programu, zaniká na konci programu

`int x;`

Jednoduché datové typy
- — proměnná obsahuje jednu hodnotu daného typu (není dělitelná na části)
- — Celočíselné: char, short, int, long, long long, (bool)
- — Racionální: float, double, long double

Složené datové typy
- — skládá se z jiných datových typů
- — umožňují přiřadit k jednomu proměnnému více hodnot
- — proměnná složeného typu (skládá se ze složek, stejného nebo různých proměnných do jedné)
- — složka funguje jako samostatná proměnná

Homogenní (jednoduchý, návrh)
- — pole / vícerozměrné pole (pole polí)
- — každá složka (prvek) je stejného typu
- → bázový typ (základní typ int, char)

Heterogenní
- — struktura (návrh)
- — každá složka může být jiného typu
- — rozsáhlá abstrakce

Ukazatelové typy
- — proměnná obsahuje adresu paměťových míst
- — Rozlišení: TYP UKAZATEL a UKAZATEL JAKO PROMĚNNÁ
- • **Typový ukazatel**
  - — Odkazované paměťové místo má velikost daného bázového typu
  - — Lze vyjádřit ukazatelový typy na jakýkoliv datový typ
- • **Netypový ukazatel**
  - — Jinak se pracovat příslušnost typu void
  - — Před pracováním proměnných je nutné je přetypovat na typový ukazatel

2. Programovací jazyk – jazykové konstrukce

Pole

- jednorozměrné – souvislý blok paměti
  - homogenní
  - přístup k prvkům pomocí [ ]

- vícerozměrné pole – pole polí
  ↳ při dynamické alokaci nemusí být jak celek souvislé

// vícedimenzionální pole ukazatelů na funkce

`FUNC operace [2] = { moje Fu 1, moje Fu 2};`

Struktura

- skládá se ze složek libovolných typů
- složka může být i struktura (kromě vlastního typu)
- složka může být i ukazatel na svůj vlastní typ

přístup ke složkám – přes ukazatel na strukturní proměnné

ukazatel → složka } stejné
(*ukazatel). složka }

```
Tosta clock = { .jmeno = "Jozef"};
Tosta *prom = & osta;
prom → role = 2.3;
```

Výrazy (expression)

- výraz má nebo reprezentuje hodnotu konkrétního typu

L – hodnota – proměnné

- výraz reprezentující místo v paměti, kam lze zapisovat
- může se ocitnout na levé/pravé straně přiřazovacího operátoru

např. $X = a + b;$ $\Delta \to N[i].u = \&u;$

Výsledek výrazu je hodnota

P – hodnota – hodnota

- výraz reprezentující hodnotu konkrétního typu, ale NE konkrétní místo v
- NEMŮŽE SE vyskytnout na levé straně přiřazovacího operátoru

např. $X = a + b;$ ale $a + b = 10;$ toto nejde !

Dereferenční operátor *

- vrací hodnotu, na niž proměnná typu ukazatel odkazuje
- typ celého výrazu odpovídá cílovému typu ukazatele
- celý výraz je L – hodnotou
- např. float $y = *pv;$

Operátor volání funkce ( )

- příkaz(); sin()
- máme-li funkci provedený lze operátor volání je chápán jako
  výraz typu ukazatel na funkci
- sin; // nejde o syntaktickou chybu, ale o prázdný příkaz !

— volání funkce (podprogramu)

  `int vysledek = soucet (3, 4);`

Podprogram

- úsek kódu, který je definován jménem, aby ho lze po opakovaně provádět

Funkce – vrací hodnotu

Procedura – nevyhodnocuje návratové hodnoty

Referenční operátor &

- & proměnná; & (L-hodnota) (reference)
- vrací adresu proměnné nebo výrazu, který je L – hodnota
- typ výrazu odpovídá typu ukazatel na typ proměnné nebo výrazu
- celý výraz je P – hodnota ! Je to hodnota adresy, na proměnné kterou odkazovaná bude adresou

Příkaz (jméno výrazu) (statement) — Výsledek příkazu je la činnost

- je instrukce nebo akce, kterou program provádí (určuje co program má dělat v určitý okamžik)

např. – přiřazovací příkaz (int $x = 10;$)

- podmíněný příkaz `if (x > 10) { printf (....)`
  ↳ vnořené `}else if (x < 3) { .....`
  `}else { ..}`
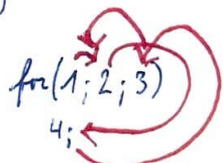
CYKLUS
- cyklus for `(int i = 0; i < m; i++) { ...}`
- cyklus while `(X >) { ...}`
- cyklus do-while: `do { i++; } while (i < 5);`
  ↳ vykoná blok kódu aspoň jednou a poté hodnotu podmínku

— switch: `switch (day) { case 1: ____;`
  `case 2: ____;`
  `3 default: ____;`

break = vyskočení z cyklu
continue = jde na další průchod
return = vyskočení funkce, vrací hodnotu

$for(1; 2; 3)$
$4;$