Dynamické proměnné

Programovací techniky

doc. Ing. Jiří Rybička, Dr.
Ústav informatiky
PEF MENDELU v Brně
rybicka@mendelu.cz

Statické proměnné

- **Terminologie**

Statické proměnné

- **Terminologie**
  - Pracovní definice: Statická proměnná = proměnná vzniklá deklarací a umístěná překladačem automaticky do úseku paměti pro tyto proměnné. Objekty: „statický“ má i jiný význam (viz ZOO).

Statické proměnné

- **Terminologie**
  - Pracovní definice: Statická proměnná = proměnná vzniklá deklarací a umístěná překladačem automaticky do úseku paměti pro tyto proměnné. Objekty: „statický“ má i jiný význam (viz ZOO).
  - Vzniká okamžikem deklarace a v tom okamžiku zabere paměť.

Statické proměnné

- **Terminologie**
  - Pracovní definice: Statická proměnná = proměnná vzniklá deklarací a umístěná překladačem automaticky do úseku paměti pro tyto proměnné. Objekty: „statický“ má i jiný význam (viz ZOO).
  - Vzniká okamžikem deklarace a v tom okamžiku zabere paměť.
  - Zaniká s koncem bloku, ve kterém byla deklarována, tehdy uvolní paměť.

Statické proměnné

- **Terminologie**
  - Pracovní definice: Statická proměnná = proměnná vzniklá deklarací a umístěná překladačem automaticky do úseku paměti pro tyto proměnné. Objekty: „statický“ má i jiný význam (viz ZOO).
  - Vzniká okamžikem deklarace a v tom okamžiku zabere paměť.
  - Zaniká s koncem bloku, ve kterém byla deklarována, tehdy uvolní paměť.
  - Patří sem i parametry podprogramu volané hodnotou (zápis parametrů má rovněž tvar deklarace).

Statické proměnné

- **Terminologie**
  - Pracovní definice: Statická proměnná = proměnná vzniklá deklarací a umístěná překladačem automaticky do úseku paměti pro tyto proměnné. Objekty: „statický“ má i jiný význam (viz ZOO).
  - Vzniká okamžikem deklarace a v tom okamžiku zabere paměť.
  - Zaniká s koncem bloku, ve kterém byla deklarována, tehdy uvolní paměť.
  - Patří sem i parametry podprogramu volané hodnotou (zápis parametrů má rovněž tvar deklarace).
  - Identifikátor proměnné představuje přímo adresu proměnné.

Statické proměnné

- **Terminologie**
  - Pracovní definice: Statická proměnná = proměnná vzniklá deklarací a umístěná překladačem automaticky do úseku paměti pro tyto proměnné. Objekty: „statický“ má i jiný význam (viz ZOO).
  - Vzniká okamžikem deklarace a v tom okamžiku zabere paměť.
  - Zaniká s koncem bloku, ve kterém byla deklarována, tehdy uvolní paměť.
  - Patří sem i parametry podprogramu volané hodnotou (zápis parametrů má rovněž tvar deklarace).
  - Identifikátor proměnné představuje přímo adresu proměnné.
  - Paměť je přidělována na systémovém zásobníku, jehož velikost je většinou omezena (relativně).

Statické proměnné

- **Použití**
  - „Menší“ množství dat (megabajty?)

Statické proměnné

- **Použití**
  - „Menší“ množství dat (megabajty?)
  - Přímý přístup k datům = maximálně rychlé.

Statické proměnné

- **Použití**
  - „Menší“ množství dat (megabajty?)
  - Přímý přístup k datům = maximálně rychlé.
  - Někdy nevýhodné hospodaření s pamětí (proměnná zabírá paměť o neměnné velikosti po celou dobu své existence).

Statické proměnné

- **Použití**
  - „Menší“ množství dat (megabajty?)
  - Přímý přístup k datům = maximálně rychlé.
  - Někdy nevýhodné hospodaření s pamětí (proměnná zabírá paměť o neměnné velikosti po celou dobu své existence).
  - „Omezení“ na předdefinované typy a struktury (všechny jednoduché typy, výčet, struct, union, objekt).

Statické proměnné

- **Použití**
  - „Menší“ množství dat (megabajty?)
  - Přímý přístup k datům = maximálně rychlé.
  - Někdy nevýhodné hospodaření s pamětí (proměnná zabírá paměť o neměnné velikosti po celou dobu své existence).
  - „Omezení“ na předdefinované typy a struktury (všechny jednoduché typy, výčet, struct, union, objekt).
  - POZOR! Pole *není* statická proměnná! Záludné, protože je deklarováno podobně jako ostatní typy.

Dynamické proměnné

- **Motivace**

Dynamické proměnné

- **Motivace**
  - Oblast paměti pro ukládání hodnot může kdykoliv vzniknout a kdykoliv zaniknout (lepší hospodaření s pamětí).

Dynamické proměnné

- **Motivace**
  - Oblast paměti pro ukládání hodnot může kdykoliv vzniknout a kdykoliv zaniknout (lepší hospodaření s pamětí).
  - Lze alokovat jen tolik, kolik aktuálně potřebujeme (opět lepší hospodaření s pamětí).

Dynamické proměnné

- **Motivace**
  - Oblast paměti pro ukládání hodnot může kdykoliv vzniknout a kdykoliv zaniknout (lepší hospodaření s pamětí).
  - Lze alokovat jen tolik, kolik aktuálně potřebujeme (opět lepší hospodaření s pamětí).
  - Alokace paměti je v oblasti, která je určena pro velké objemy dat.

Dynamické proměnné

- **Motivace**
  - Oblast paměti pro ukládání hodnot může kdykoliv vzniknout a kdykoliv zaniknout (lepší hospodaření s pamětí).
  - Lze alokovat jen tolik, kolik aktuálně potřebujeme (opět lepší hospodaření s pamětí).
  - Alokace paměti je v oblasti, která je určena pro velké objemy dat.
  - Možnost libovolné struktury svázané adresami.

Dynamické proměnné

- **Motivace**
  - Oblast paměti pro ukládání hodnot může kdykoliv vzniknout a kdykoliv zaniknout (lepší hospodaření s pamětí).
  - Lze alokovat jen tolik, kolik aktuálně potřebujeme (opět lepší hospodaření s pamětí).
  - Alokace paměti je v oblasti, která je určena pro velké objemy dat.
  - Možnost libovolné struktury svázané adresami.
- **Demotivace**

Dynamické proměnné

- **Motivace**
  - Oblast paměti pro ukládání hodnot může kdykoliv vzniknout a kdykoliv zaniknout (lepší hospodaření s pamětí).
  - Lze alokovat jen tolik, kolik aktuálně potřebujeme (opět lepší hospodaření s pamětí).
  - Alokace paměti je v oblasti, která je určena pro velké objemy dat.
  - Možnost libovolné struktury svázané adresami.
- **Demotivace**
  - Nepřímá adresace (časová režie).

Dynamické proměnné

- **Motivace**
  - Oblast paměti pro ukládání hodnot může kdykoliv vzniknout a kdykoliv zaniknout (lepší hospodaření s pamětí).
  - Lze alokovat jen tolik, kolik aktuálně potřebujeme (opět lepší hospodaření s pamětí).
  - Alokace paměti je v oblasti, která je určena pro velké objemy dat.
  - Možnost libovolné struktury svázané adresami.
- **Demotivace**
  - Nepřímá adresace (časová režie).
  - Uchovávání adres navíc (paměťová režie).

Dynamické proměnné

- **Motivace**
  - Oblast paměti pro ukládání hodnot může kdykoliv vzniknout a kdykoliv zaniknout (lepší hospodaření s pamětí).
  - Lze alokovat jen tolik, kolik aktuálně potřebujeme (opět lepší hospodaření s pamětí).
  - Alokace paměti je v oblasti, která je určena pro velké objemy dat.
  - Možnost libovolné struktury svázané adresami.
- **Demotivace**
  - Nepřímá adresace (časová režie).
  - Uchovávání adres navíc (paměťová režie).
  - Deklarace musí být doprovázena alokací (potenciální chyby).

Dynamické proměnné

- **Motivace**
  - Oblast paměti pro ukládání hodnot může kdykoliv vzniknout a kdykoliv zaniknout (lepší hospodaření s pamětí).
  - Lze alokovat jen tolik, kolik aktuálně potřebujeme (opět lepší hospodaření s pamětí).
  - Alokace paměti je v oblasti, která je určena pro velké objemy dat.
  - Možnost libovolné struktury svázané adresami.
- **Demotivace**
  - Nepřímá adresace (časová režie).
  - Uchovávání adres navíc (paměťová režie).
  - Deklarace musí být doprovázena alokací (potenciální chyby).
  - Lineární procházení struktur (časová režie, někdy značná).

Obecně

- Datový typ **ukazatel** = adresa dat

Obecně

- Datový typ **ukazatel** = adresa dat
- Ukazatel: určitý – ukazuje na konkrétní typ

Obecně

- Datový typ **ukazatel** = adresa dat
- Ukazatel: určitý – ukazuje na konkrétní typ
- obecný – ukazuje na cokoliv (obecná data)

Obecně

- Datový typ **ukazatel** = adresa dat
- Ukazatel: určitý – ukazuje na konkrétní typ
- obecný – ukazuje na cokoliv (obecná data)
- Univerzální kompatibilita ukazatelů

Obecně

- Datový typ **ukazatel** = adresa dat
- Ukazatel: určitý – ukazuje na konkrétní typ
- obecný – ukazuje na cokoliv (obecná data)
- Univerzální kompatibilita ukazatelů
- Podobně jako adresa dat existuje i adresa podprogramu

Obecně

- Datový typ **ukazatel** = adresa dat
- Ukazatel: určitý – ukazuje na konkrétní typ
- obecný – ukazuje na cokoliv (obecná data)
- Univerzální kompatibilita ukazatelů
- Podobně jako adresa dat existuje i adresa podprogramu
- Práce s adresou:

Obecně

- Datový typ **ukazatel** = adresa dat
- Ukazatel: určitý – ukazuje na konkrétní typ
- obecný – ukazuje na cokoliv (obecná data)
- Univerzální kompatibilita ukazatelů
- Podobně jako adresa dat existuje i adresa podprogramu
- Práce s adresou:
  - získání platné adresy jiného existujícího prvku (kdekoliv) v paměti

Obecně

- Datový typ **ukazatel** = adresa dat
- Ukazatel: určitý – ukazuje na konkrétní typ
- obecný – ukazuje na cokoliv (obecná data)
- Univerzální kompatibilita ukazatelů
- Podobně jako adresa dat existuje i adresa podprogramu
- Práce s adresou:
  - získání platné adresy jiného existujícího prvku (kdekoliv) v paměti
  - získání platné adresy alokací paměti

Obecně

- Datový typ **ukazatel** = adresa dat
- Ukazatel: určitý – ukazuje na konkrétní typ
- obecný – ukazuje na cokoliv (obecná data)
- Univerzální kompatibilita ukazatelů
- Podobně jako adresa dat existuje i adresa podprogramu
- Práce s adresou:
  - získání platné adresy jiného existujícího prvku (kdekoliv) v paměti
  - získání platné adresy alokací paměti
  - přiřazení konstanty prázdného ukazatele

Obecně

- Datový typ **ukazatel** = adresa dat
- Ukazatel: určitý – ukazuje na konkrétní typ
- obecný – ukazuje na cokoliv (obecná data)
- Univerzální kompatibilita ukazatelů
- Podobně jako adresa dat existuje i adresa podprogramu
- Práce s adresou:
  - získání platné adresy jiného existujícího prvku (kdekoliv) v paměti
  - získání platné adresy alokací paměti
  - přiřazení konstanty prázdného ukazatele
  - ukazatelová aritmetika: adresa $\pm$ celé číslo

Obecně

- Datový typ **ukazatel** = adresa dat
- Ukazatel: určitý – ukazuje na konkrétní typ
- obecný – ukazuje na cokoliv (obecná data)
- Univerzální kompatibilita ukazatelů
- Podobně jako adresa dat existuje i adresa podprogramu
- Práce s adresou:
  - získání platné adresy jiného existujícího prvku (kdekoliv) v paměti
  - získání platné adresy alokací paměti
  - přiřazení konstanty prázdného ukazatele
  - ukazatelová aritmetika: adresa $\pm$ celé číslo
  - porovnání ukazatelů, rozdíl ukazatelů

Jak na to v C++

- Deklarace dynamické proměnné: `typ *id;`

Jak na to v C++

- Deklarace dynamické proměnné: `typ *id;`
- Získání adresy jiné proměnné: `id = &proměnná;`

Jak na to v C++

- Deklarace dynamické proměnné: `typ *id;`
- Získání adresy jiné proměnné: `id = &proměnná;`
- Získání adresy alokací paměti: `id = new typ;`

Jak na to v C++

- Deklarace dynamické proměnné: `typ *id;`
- Získání adresy jiné proměnné: `id = &proměnná;`
- Získání adresy alokací paměti: `id = new typ;`
- Vložení prázdného ukazatele: `id = NULL;`

Krátký příklad

- Jednoduchá manipulace s dynamickou proměnnou:

```
int *a, b;          //deklarace
a = new int;        //alokace
*a = 10;            //přiřazení
b = *a;             //použití
cout<<a <<": " <<*a <<", " <<b <<endl;
delete a;           //uvolnění paměti
a = &b;             //získání adresy
b = 20;             //práce reference
cout<<a <<": " <<*a <<", " <<b <<endl;
```

Krátký příklad

- Jednoduchá manipulace s dynamickou proměnnou:

```
int *a, b;      //deklarace
a = new int;    //alokace
*a = 10;        //přiřazení
b = *a;         //použití
cout<<a <<": " <<*a <<", " <<b <<endl;
delete a;       //uvolnění paměti
a = &b;         //získání adresy
b = 20;         //práce reference
cout<<a <<": " <<*a <<", " <<b <<endl;
```

- `0x109f010: 10, 10`

Krátký příklad

- Jednoduchá manipulace s dynamickou proměnnou:

```
int *a, b;          //deklarace
a = new int;        //alokace
*a = 10;            //přiřazení
b = *a;             //použití
cout<<a <<": " <<*a <<", " <<b <<endl;
delete a;           //uvolnění paměti
a = &b;             //získání adresy
b = 20;             //práce reference
cout<<a <<": " <<*a <<", " <<b <<endl;
```

- `0x109f010: 10, 10`
- `0x7ffdeabbea54: 20, 20`

Dynamické struktury – seznam

- Seznamy, stromy, grafy, ...

Dynamické struktury – seznam

- Seznamy, stromy, grafy, ...
- Nejjednodušší strukturou je seznam

Dynamické struktury – seznam

- Seznamy, stromy, grafy, ...
- Nejjednodušší strukturou je seznam
- Elementem je záznam, jehož složkou jsou data a ukazatel na následníka

Dynamické struktury – seznam

- Seznamy, stromy, grafy, ...
- Nejjednodušší strukturou je seznam
- Elementem je záznam, jehož složkou jsou data a ukazatel na následníka
- Definice záznamu:

```
struct TypPrvek {  //vytvoření datového typu
   TypData Data; //typ dat není teď podstatný
   TypPrvek *Dalsi;
};
typedef TypPrvek *UkPrvek;  //ukazatel na prvek
```

Dynamické struktury – seznam

- Seznamy, stromy, grafy, ...
- Nejjednodušší strukturou je seznam
- Elementem je záznam, jehož složkou jsou data a ukazatel na následníka
- Definice záznamu:

```
struct TypPrvek {  //vytvoření datového typu
   TypData Data; //typ dat není teď podstatný
   TypPrvek *Dalsi;
};
typedef TypPrvek *UkPrvek;  //ukazatel na prvek
```

- Z těchto záznamů se tvoří posloupnost spojená ukazatelem na další prvek.

Seznam jako zásobník

- Operace: přidání (push), odebrání (pop)

```
UkPrvek Vrchol=NULL, Pom, Smazat;
TypData Vstup;
while (cin>>Vstup) { //všechna data ze vstupu
   Pom = new TypPrvek;
   (*Pom).Data = Vstup; //Pom->Data = Vstup
   (*Pom).Dalsi = Vrchol;
   Vrchol = Pom;  //nový je první
}
Pom = Vrchol; //výpis: pomocný na začátek seznamu
while (Pom != NULL) {
   cout << Pom->Data << endl;
   Smazat = Pom;   //nelze nyní delete Pom!!
   Pom = Pom->Dalsi;
   delete Smazat; //odstranění z paměti
}
```

Seznam jako fronta

- Operace: přidání (queup), odebrání (remove)

```
UkPrvek Hlava=NULL, Ocas=NULL, Pom, Smazat;
TypData Vstup;
while (cin>>Vstup) { //všechna data ze vstupu
   if (Hlava==NULL) { //fronta je prázdná
      Hlava = new TypPrvek;
      Ocas = Hlava;  //první je zároveň poslední
   } else {   //fronta není prázdná
      Ocas->Dalsi = new TypPrvek; //vazba na nový
      Ocas = Ocas->Dalsi; //nový konec
   }
   Ocas->Data = Vstup;  //naplnění daty
   Ocas->Dalsi = NULL; //nový je poslední
}
```

Seznam jako fronta

- Operace: přidání (queup), odebrání (remove)

```
UkPrvek Hlava=NULL, Ocas=NULL, Pom, Smazat;
TypData Vstup;
while (cin>>Vstup) { //všechna data ze vstupu
   if (Hlava==NULL) { //fronta je prázdná
      Hlava = new TypPrvek;
      Ocas = Hlava;  //první je zároveň poslední
   } else {   //fronta není prázdná
      Ocas->Dalsi = new TypPrvek; //vazba na nový
      Ocas = Ocas->Dalsi; //nový konec
   }
   Ocas->Data = Vstup;  //naplnění daty
   Ocas->Dalsi = NULL; //nový je poslední
}
```

- Výpis se provede stejně jako u zásobníku.

Obecný seznam a další varianty

- **Obecný** seznam – operace: přidání prvku do libovolného místa, odebrání prvku z libovolného místa, vyhledání pozice prvku.

Obecný seznam a další varianty

- **Obecný** seznam – operace: přidání prvku do libovolného místa, odebrání prvku z libovolného místa, vyhledání pozice prvku.
- Seznam **s aktivním prvkem** – má ukazatel na začátek a ukazatel na aktivní prvek (s ním se pracuje). Přidání za/před aktivního, odebrání aktivního (prvku za aktivním).

Obecný seznam a další varianty

- **Obecný** seznam – operace: přidání prvku do libovolného místa, odebrání prvku z libovolného místa, vyhledání pozice prvku.
- Seznam **s aktivním prvkem** – má ukazatel na začátek a ukazatel na aktivní prvek (s ním se pracuje). Přidání za/před aktivního, odebrání aktivního (prvku za aktivním).
- Práce s aktivitou: nastavení na začátek, přesun na další prvek, test aktivity.

Obecný seznam a další varianty

- **Obecný** seznam – operace: přidání prvku do libovolného místa, odebrání prvku z libovolného místa, vyhledání pozice prvku.
- Seznam **s aktivním prvkem** – má ukazatel na začátek a ukazatel na aktivní prvek (s ním se pracuje). Přidání za/před aktivního, odebrání aktivního (prvku za aktivním).
- Práce s aktivitou: nastavení na začátek, přesun na další prvek, test aktivity.
- **Dvousměrný** seznam – navíc ukazatel na předchůdce. Umožňuje procházení oběma směry. Jinak stejné operace jako jednosměrný seznam.

Obecný seznam a další varianty

- **Obecný** seznam – operace: přidání prvku do libovolného místa, odebrání prvku z libovolného místa, vyhledání pozice prvku.
- Seznam **s aktivním prvkem** – má ukazatel na začátek a ukazatel na aktivní prvek (s ním se pracuje). Přidání za/před aktivního, odebrání aktivního (prvku za aktivním).
- Práce s aktivitou: nastavení na začátek, přesun na další prvek, test aktivity.
- **Dvousměrný** seznam – navíc ukazatel na předchůdce. Umožňuje procházení oběma směry. Jinak stejné operace jako jednosměrný seznam.
- **Kruhový** seznam (jednosměrný): poslední prvek ukazuje zpět na první prvek. Ukazatel na libovolný prvek. Průchod: test místa, kde průchod začal.

O polích...

- Mají dvojí tvář: jsou to dynamické struktury, ale dá se s nimi pracovat jako se statickou proměnnou

O polích...

- Mají dvojí tvář: jsou to dynamické struktury, ale dá se s nimi pracovat jako se statickou proměnnou
- Deklarace (statická): `typ id[počet];`

O polích...

- Mají dvojí tvář: jsou to dynamické struktury, ale dá se s nimi pracovat jako se statickou proměnnou
- Deklarace (statická): `typ id[počet];`
- Deklarace (dynamická): `typ *id = new typ[počet];`

O polích...

- Mají dvojí tvář: jsou to dynamické struktury, ale dá se s nimi pracovat jako se statickou proměnnou
- Deklarace (statická): `typ id[počet];`
- Deklarace (dynamická): `typ *id = new typ[počet];`
- Přístup ke složce: `id[index]`, nebo `*id`

O polích...

- Mají dvojí tvář: jsou to dynamické struktury, ale dá se s nimi pracovat jako se statickou proměnnou
- Deklarace (statická): `typ id[počet];`
- Deklarace (dynamická): `typ *id = new typ[počet];`
- Přístup ke složce: `id[index]`, nebo `*id`
- Příklad průchodu polem:

```
int pole[300], *uk;
unsigned int pocet;
... //pole je naplněno ”pocet” hodnotami
uk = &pole[0];  //adresa 1. prvku
//lze také uk = pole;
for (int i=0; i<pocet; i++)
   cout << *(uk + i) << ” ”;
   //uk + i je adresa i-tého prvku
```

O polích...

- Pole versus dynamický seznam

O polích...

- Pole versus dynamický seznam
- Implementace fronty a zásobníku polem

O polích...

- Pole versus dynamický seznam
- Implementace fronty a zásobníku polem
- Indexace – indexový výraz:

$$A = PA + (I - I_0) \cdot V$$

O polích...

- Pole versus dynamický seznam
- Implementace fronty a zásobníku polem
- Indexace – indexový výraz:

$$A = PA + (I - I_0) \cdot V$$

- Strojová optimalizace: $I_0 = 0$ – případ polí v C/C++ (i jinde);
  $V = 2^n$ – aritmetický posun místo násobení

O polích...

- Pole versus dynamický seznam
- Implementace fronty a zásobníku polem
- Indexace – indexový výraz:

$$A = PA + (I - I_0) \cdot V$$

- Strojová optimalizace: $I_0 = 0$ – případ polí v C/C++ (i jinde);
  $V = 2^n$ – aritmetický posun místo násobení
- Algoritmy pro práci s polem: naplnění $\rightarrow$ poslední index, počet naplněných indexů (amatérská chyba $i - 1$)

Naplnění/výpis pole

- Typická úloha: vstup obsahuje řadu hodnot (např. celá čísla), je potřebné naplnit pole. Vstup může být i zcela prázdný.

```
const int MaxRozmer = 1000;
int pole [MaxRozmer], pom;
unsigned int index = 0;
while (cin>>pom and index<MaxRozmer)
    pole[index++] = pom;
//počet prvků = index
//poslední obsazený prvek = index - 1!!
for (int i=0; i<=index-1; i++)  //i<index;
    cout << pole[i] << ” ”;
```