Dynamická alokace paměti

doc. Mgr. Jiří Dvorský, Ph.D.

Katedra informatiky
Fakulta elektrotechniky a informatiky
VŠB – TU Ostrava

Prezentace ke dni 12. února 2020





Osnova přednášky

Dynamická alokace paměti

- Podobenství o žlutém lístečku a kastrolu s gulášem
- Podobenství o kabátu v šatně na plese
- Pointery v C++

Podobenství o žlutém lístečku a kastrolu s gulášem

Pravidla hry:

- žlutý nalepovací lísteček – lísteček je volný nebo je na něm napsáno, kde je guláš nebo že guláš není
- psací stůl – lísteček je nalepen na vašem psacím stole
- kastrolek s gulášem v lednici – obsahuje lahodný pokrm, který lze konzumovat
- lednice – na základě lístečku vydá guláš, bez lístečku není guláš, lednici nelze prohledávat

Hra samotná – maminka uvaří guláš, uschová jej do lednice a synáčkovi napíše na lísteček, kde se guláš ukrývá.

Podobenství o žlutém lístečku a kastrolu s gulášem

Co můžeme říci o žlutém lístečku:

- žlutý lísteček určitě není guláš samotný
- žlutý lísteček udává **nepřímo** polohu guláše
- žlutý lísteček existuje sám o sobě, zabírá jakési malé množství paměti
- žlutý lísteček není v lednici, je jinde
- obsah žlutého lístečku – prázdný, „Guláš je v lednici“ nebo „Guláš není!“
- obsah žlutého lístečku si můžu opsat na jiný – dva lístečky odkazují na totožný guláš
- na jiný oranžový lísteček si můžu napsat kde leží žlutý lísteček
- na zelený lísteček si můžu napsat, kde leží oranžový lísteček

Podobenství o žlutém lístečku a kastrolu s gulášem

Lednice a kastrol s gulášem:

- na základě žlutého lístečku lednice vydá guláš, s gulášem lze manipulovat, například přiostřit; co se stane když si guláš vyzvednu se zeleným lístečkem?
- vydaný guláš můžu sníst; co když přijdu se zeleným lístečkem
- co když žlutý lísteček zahodím? guláš zůstane v lednici navěky
- na nový lísteček si můžu napsat že v lednici mám nový kastrol, na základě žlutého lístečku si vyzvednu kastrol s gulášem a naleji ho i do nového kastrolu a mám dva exempláře guláše (Tady dochází k porušení zákona zachování hmoty – vytváříme guláš z ničeho)

Podobenství o žlutém lístečku a kastrolu s gulášem

Informatický výklad podobenství:

- psací stůl – paměťový prostor, datový segment, místo pro uložení proměnných vzniklých v době kompilace
- lednice – paměťový prostor, halda, prostor pro dynamickou alokaci paměti
- kastrol s gulášem – dynamicky alokovaná data
- žlutý lísteček – ukazatel nebo-li pointer. Smyslem pointeru je **nepřímá adresace**. Pointer není objektem samotným (objekt ne nutně ve smyslu OOP), ale je informací, kde se objekt nalézá.

Podobenství o žlutém lístečku a kastrolu s gulášem

Možné stavy žlutého lístečku:

- čistý lísteček – neinicializovaný pointer, pointer se teď chová jako bežná proměnná
- „Guláš je v lednici“ – platný odkaz na guláš, lze provést dereferenci
- „Guláš není!“ – zvláštní, vyhrazená, hodnota. Pointer ukazuje „nikam“. Nelze zaměňovat s čistým lístečkem. Pointer nikam lze testovat ale nelze provést dereferenci.
-

```
1 DatovyTyp *ZlutyListecek;
2 int *p;
3 int *q = NULL;
4 int *r = nullptr;
```

Podobenství o žlutém lístečku a kastrolu s gulášem

Uvaření nového guláše:

- uvaření guláše lze chápat jako dynamickou alokaci objektu v paměti
- alokaci provedeme operátorem `new`

```
1 struct Kastrol
2 {
3    // gulas
4 };
5 Kastrol *ZlutyListecek = new Kastrol;
```

Podobenství o žlutém lístečku a kastrolu s gulášem

Manipulace s gulášem:

- na základě žlutého lístečku lze získat z lednice kastrol s gulášem a s tímto gulášem manipulovat
- operace získání guláše se nazývá dereference, „odpointrování“

```
1 Kastrol K = *ZlutyListecek;
2 K += LzicePapriky;
```

nebo přímo

```
1 *ZlutyListecek += LzicePapriky;
```

Dereference neinicializovaného pointeru nebo `nullptr`

Neexistující objekty nelze dereferencovat! Fatální chyba.

Podobenství o žlutém lístečku a kastrolu s gulášem

Likvidace guláše:

- likvidaci guláše (snězení, vylití) lze chápat jako delokaci objektu v paměti
- delokaci provedeme pomocí operátoru `delete`

```
1 delete ZlutyListecek;
```

Memory leak

Každý dynamicky alokovaný objekt musí být uvolněn z paměti! Jinak dojde k úniku paměti. Nepoužívané objekty za vás nikdo neuklízí.

Podobenství o žlutém lístečku a kastrolu s gulášem

Operace s pointery:

- **derefence** – na základě žlutého lístečku dostanu kastrol s gulášem, „odpointrování“
- **reference** – získání umístění objektu, poloha guláše v lednici, výsledek lze zapsat na žlutý lísteček
- reference a dereference jsou inverzní operace

```
1 struct Kastrol
2 {
3    // nejaka data
4 };
5 Kastrol JinyHotovyGulas;
6 Kastrol *ZlutyListecek = &JinyHotovyGulas;
```

Podobenství o žlutém lístečku a kastrolu s gulášem

Operace s pointery:

```
1 Kastrol *ZlutyListecek;
2 Kastrol *ZelenyListecek;
```

- přiřazení pointerů `ZelenyListecek = ZlutyListecek`, ve výsledku oba lístečky ukazují na tentýž kastrol, nebo na nic, nebo mají shodnou neinicializovanou hodnotu
- přiřazení objektů na které pointery ukazují
  `*ZelenyListecek = *ZlutyListecek`, ve výsledku se obsah guláše přelil do dašího kastrolu

Co pointer není

- Pointer není adresa v paměti. Ano pointer musí nutně skončit jako nějaká binární hodnota, kterou lze interpretovat jako číslo.
- Nelze si ale představovat, že na daném bajtu v paměti bychom nějakým přístojem našli binární ekvivalent odkazovaných dat.
- `NULL` není nula ani jiné přirozené číslo. Možná konverze na `int` je vražedným dědictvím ANSI C. Fuj! Nebrat!
- Jazyk C++ nově (od verze C++ 11) obsahuje `nullptr`.

Podobenství o kabátu v šatně na plese

Situace:

- Vesnický hasičský, sokolský či jiný ples.
- Šatna, kde odevzdáváte kabáty je vybavena pojízdnými věšáky.
- Odevzdáte kabát a dostanete papírový lísteček s číslem například 118.
- Po plese si na základě lístečku vyzvednete kabát.
- Jestliže se na jeden věšák vejde 20 kabátů a začaly se věšáky plnit zleva, tak by šlo usuzovat, že váš kabát 118 je na šestém věšáku osmnáctý v pořadí.
- Vzhledem k tomu, že si šatnářky mohou věšáky přerovnávat, tak se nedá říci, kde se věšák v vašim kabátem v šatně nachází.
- Ale víme, že na základě lístečku kabát dostanu. Zbytek je záležitost šatnářky.
- Šatna plní roli paměti, šatnářka je memory manager, heap manager.
- Přesuny věšáků – stránkování paměti, virtuální paměť atd.

Konstantní pointery

- Co lze označit za konstantní: polohu a obsah.
- Celkem mohou nastat 4 možnosti:
- `int* p;` – vše proměnné
- `const int* p;` – konstantní obsah
- `int* const p;` – konstantní poloha
- `const int* const p;` – konstantní obsah i poloha.

Dynamická alokace pole v C++

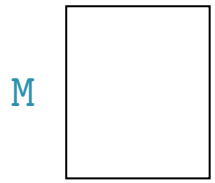
- Identifikátor pole je pointer na první prvek tohoto pole.
- Jednorozměrné pole

```
1 int *A = new int[5];
2 delete []A;
```

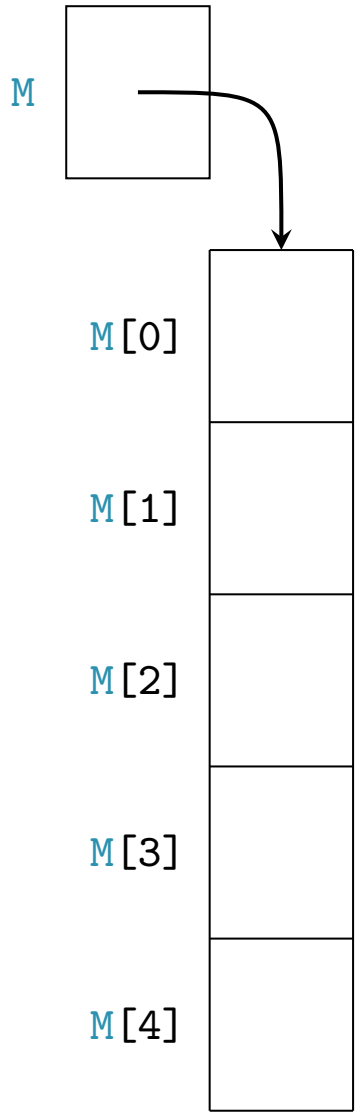
- Dvourozměrné pole

```
1 double **M = new double*[R];
2 for (int i = 0; i < R; i++)
3    M[i] = new double[C];
4
5 for (int i = 0; i < R; i++)
6    delete[]M[i];
7 delete []M;
```

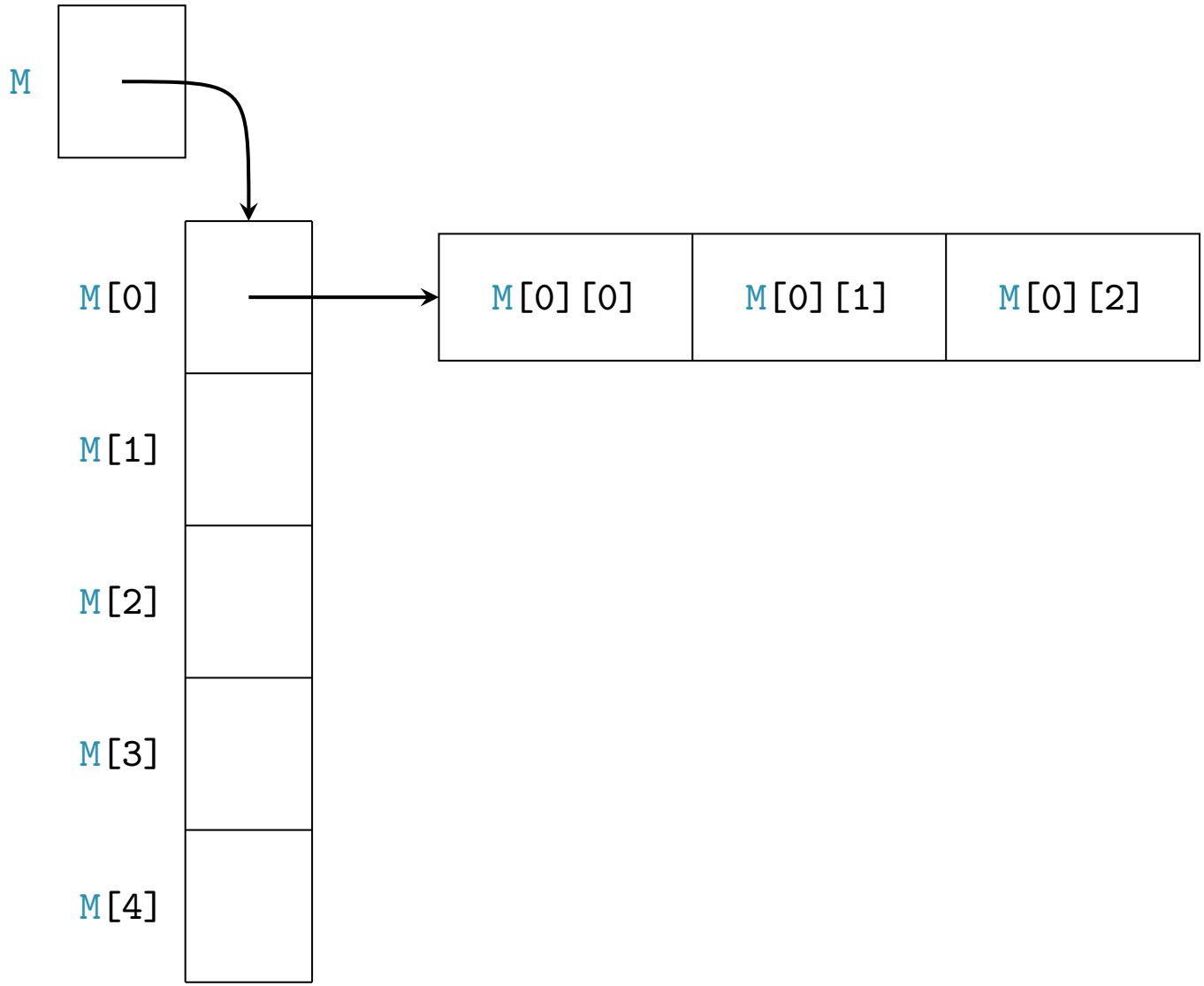
Dynamická alokace dvourozměrného pole v C++



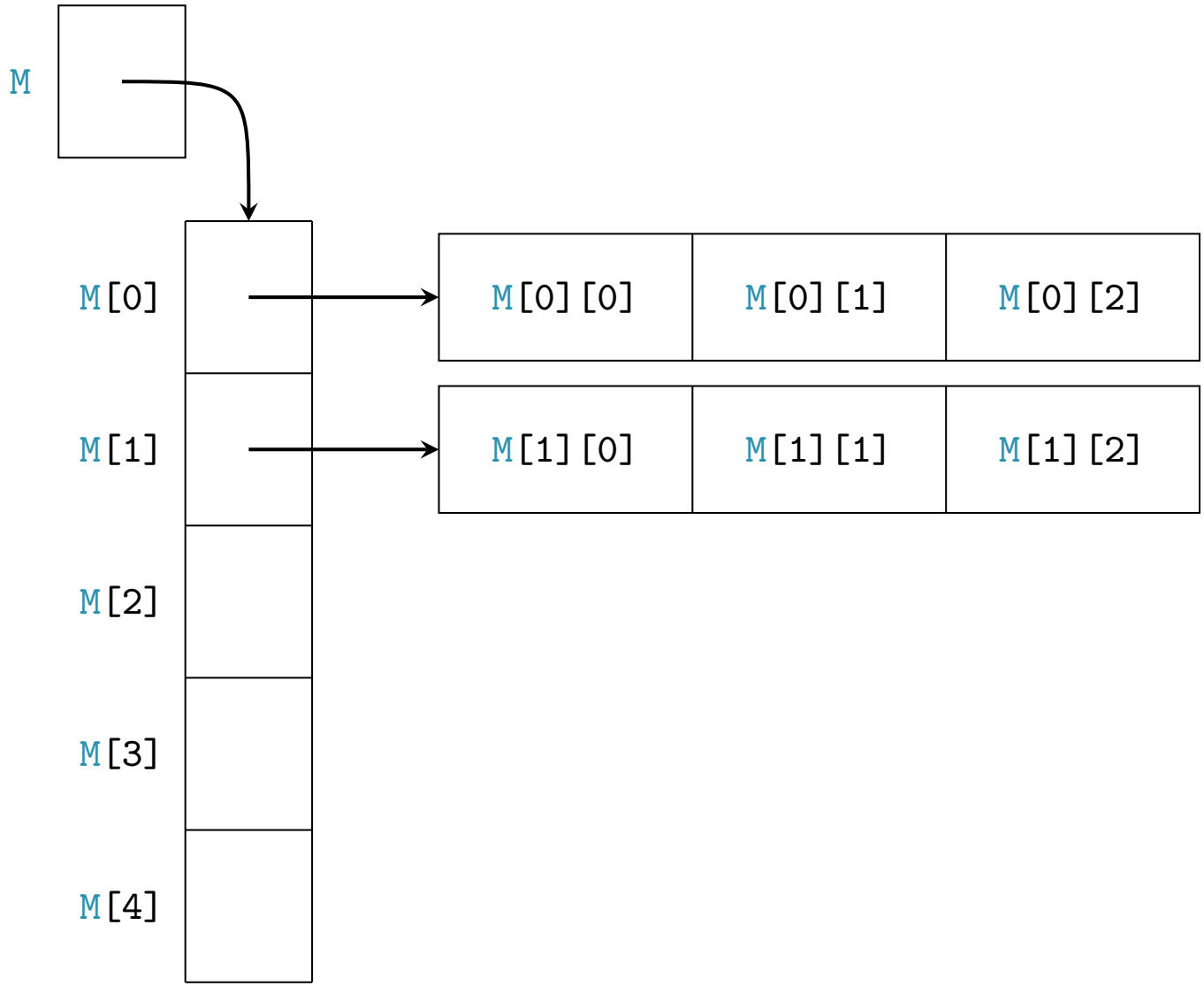
Dynamická alokace dvourozměrného pole v C++ (pokrač.)



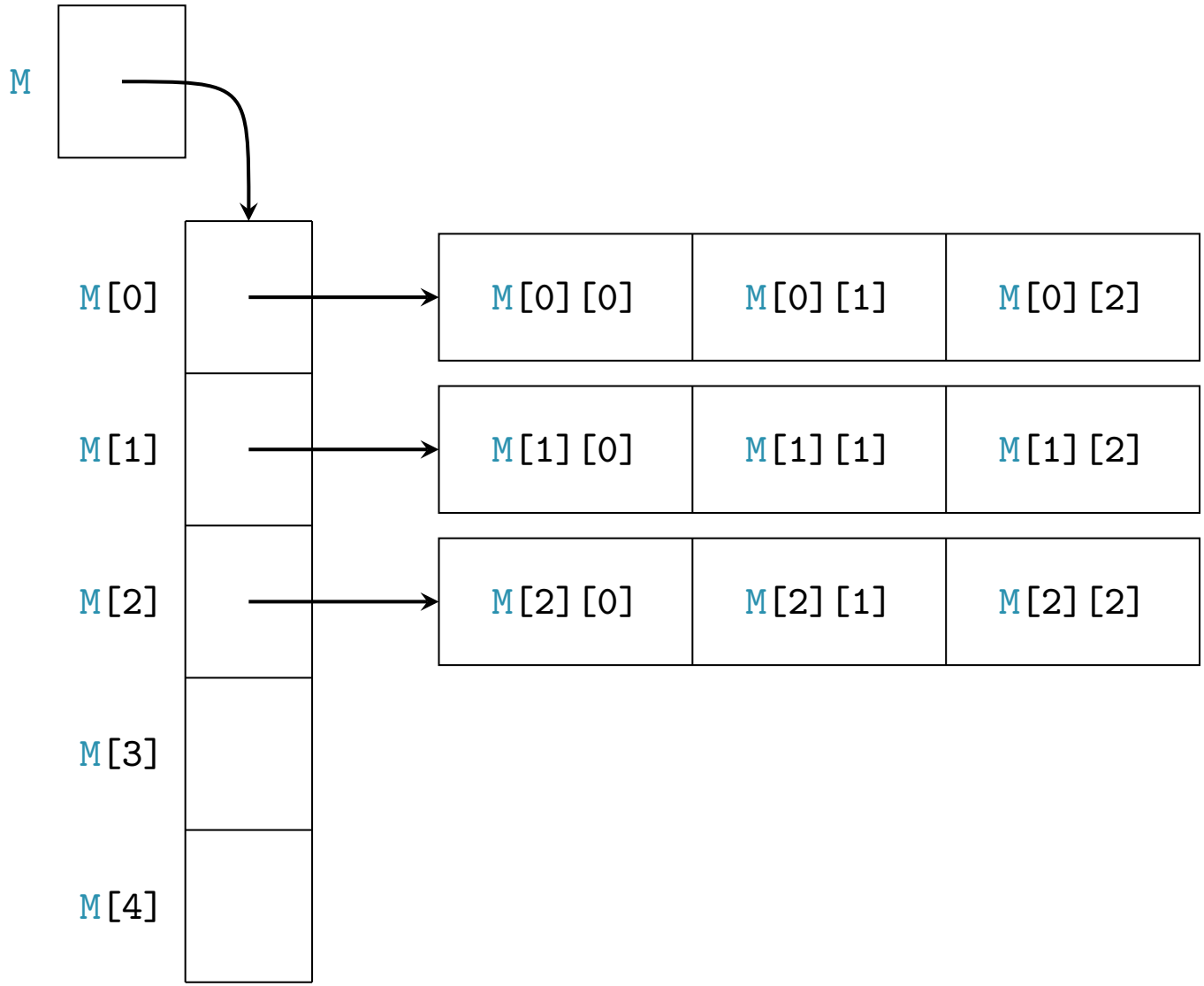
Dynamická alokace dvourozměrného pole v C++ (pokrač.)



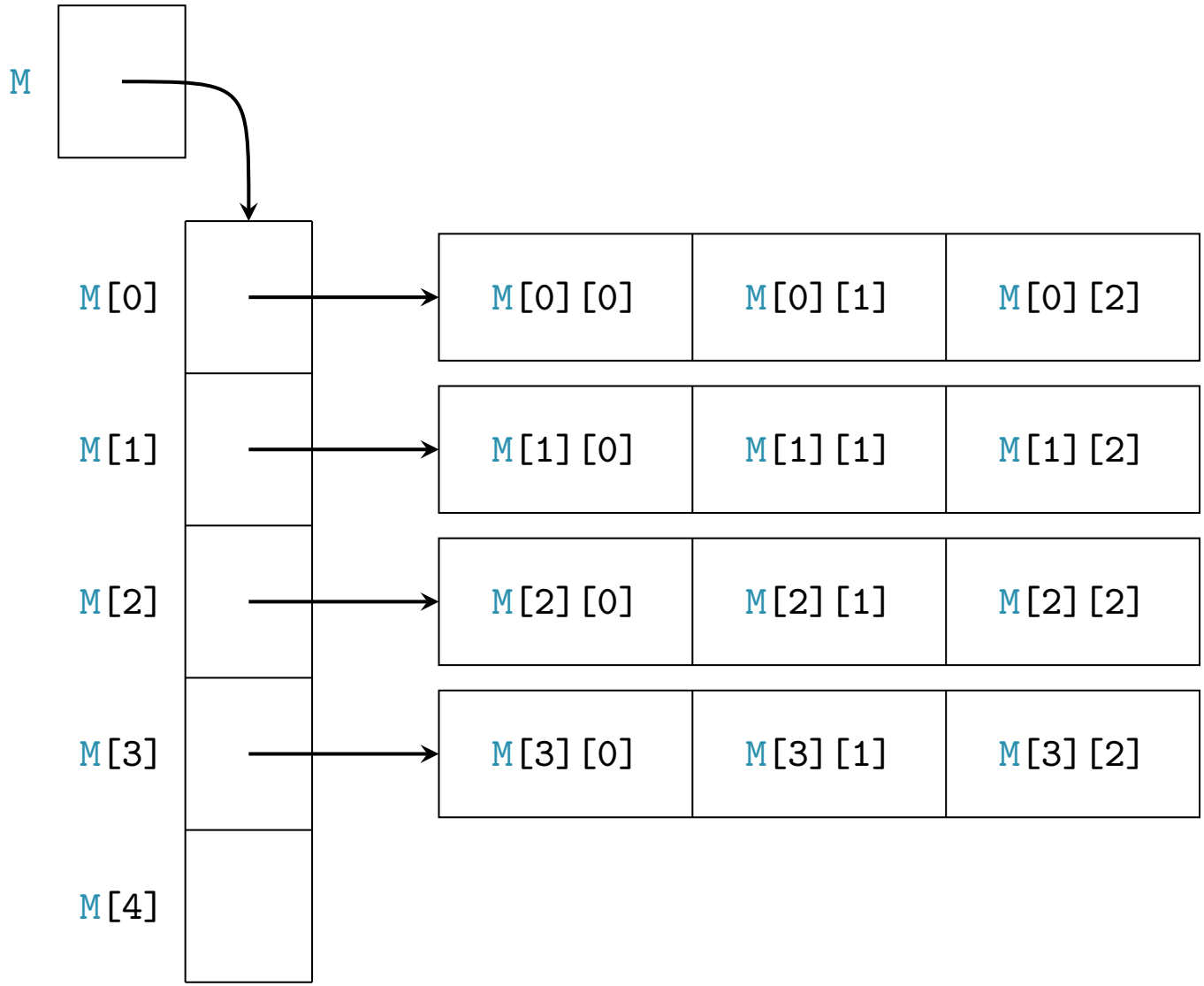
Dynamická alokace dvourozměrného pole v C++ (pokrač.)



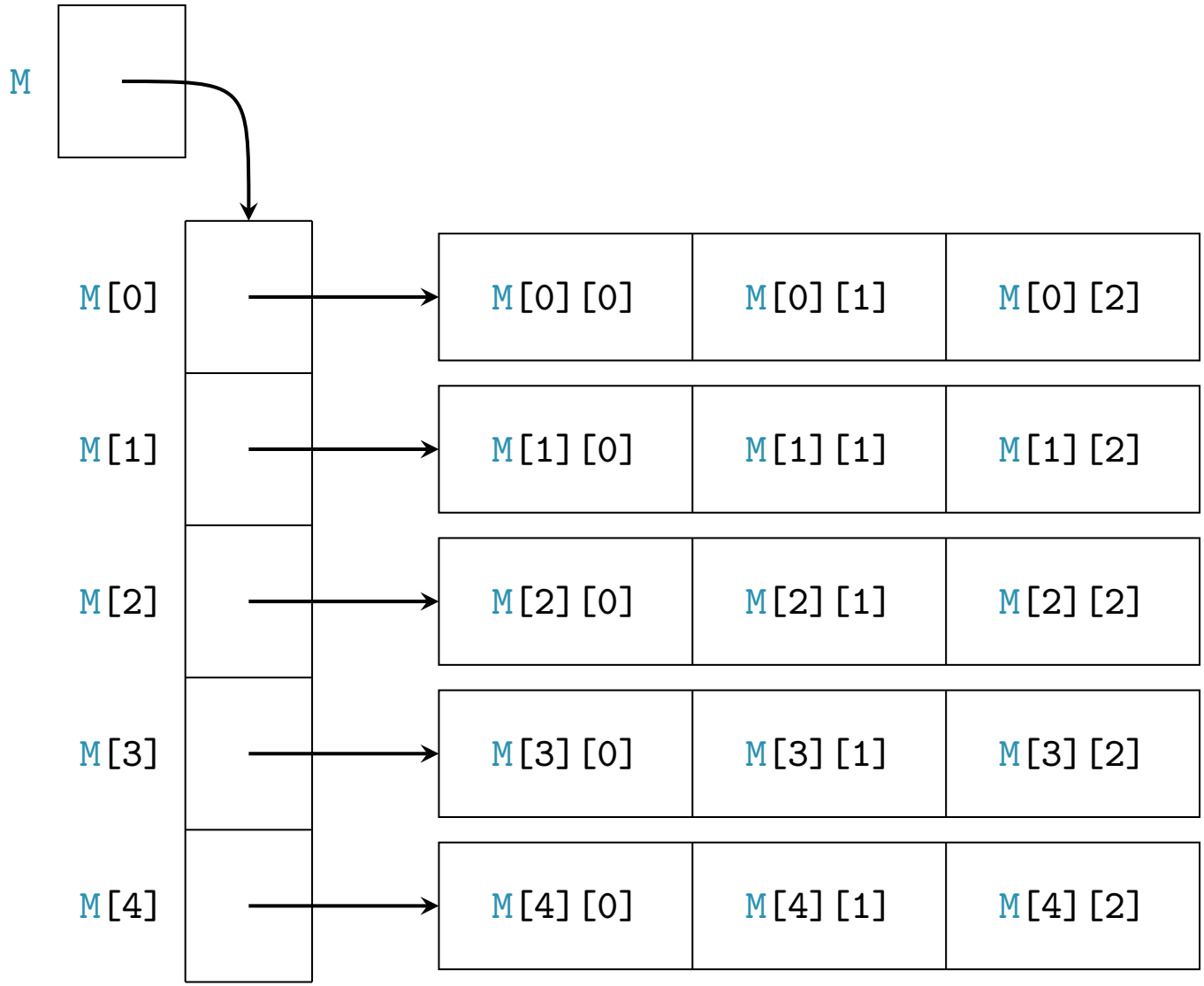
Dynamická alokace dvourozměrného pole v C++ (pokrač.)



Dynamická alokace dvourozměrného pole v C++ (pokrač.)



Dynamická alokace dvourozměrného pole v C++ (pokrač.)



Možnosti předávání parametrů funkcím

Rozlišujeme tyto způsoby předávání parametrů:

- hodnotou – vznik lokální kopie parametru
- pointerem – předáváno, kopírováno, umístění objektu
- odkazem – předáváno, kopírováno, umístění objektu. Obdoba volání pomocí pointeru, pointer samotný je před programátorem utajen kompilátorem.

Předávání polí:

- Pole jsou vždy předávána jako pointer.
- `void F(int A[])` nebo `void F(int* A)`
- Pozor na chování operátoru `sizeof`

Možnosti předávání parametrů funkcím (pokrač.)

```
1 void F(int* B)
2 {
3    cout << sizeof(B);
4 }
5
6 int A[5];
7 cout << sizeof(A);
8 F(A);
```

Pointerová artimetika

```
1 void CopyArray1(int* SourceArray, int* DestinationArray, const int N)
2 {
3    for (int i = 0; i < N; i++)
4    {
5       DestinationArray[i] = SourceArray[i];
6    }
7 }
```

Pointerová artimetika

```
1 void CopyArray2(int* SourceArray, int* DestinationArray, const int N)
2 {
3    int* SPtr = SourceArray;
4    int* DPtr = DestinationArray;
5    int* EndPtr = SourceArray + N;
6    while (SPtr != EndPtr)
7    {
8       *DPtr = *SPtr;
9       SPtr += 1;
10      DPtr += 1;
11   }
12 }
```

Pointerová artimetika

```
1 void CopyArray2a(int* SourceArray, int* DestinationArray, const int N)
2 {
3    for (int *SPtr = SourceArray, *DPtr = DestinationArray; SPtr !=
         SourceArray + N; *DPtr++ = *SPtr++);
4 }
```

Kontrolní otázky

1. Co je to pointer? Jaký je význam dat obsažených v pointeru? Nápověda: zaměřte se na vysvětlení pojmu nepřímého adresování.
2. Jak deklarujeme pointer `p` na typ `int`?
3. Jak se nazývá pointer ukazující „nikam“?
4. Co je to operátor reference? Uveďte příklad použití.
5. Co je to operátor dereference? Uveďte příklad použití.
6. Předpokládejme, že máme dánu proměnnou `int a = 8;` a pointer `int *p;`. Jakým způsobem zařídíte, aby pointer `p` ukazoval na proměnnou `a`?
7. Předpokládejme, že máme deklarovány dva pointery `int* p;` a `int* q;`. Co se stane po vyhodnocení výrazu `p = q;`?
8. Předpokládejme, že máme deklarovány dva pointery `int* p;` a `int* q;`. Co se stane po vyhodnocení výrazu `*p = *q;`? Jaké zde můžou nastat problémy?

Kontrolní otázky (pokrač.)

9. Deklarujte pointer `d` na typ `double`. Proveďte alokaci paměti a dealokaci paměti. Jaké operátory na to použijete?
10. Máte dynamicky alokovat pole `a` s prvky typu `int`, které bude mít 20 prvků. Jak to provedete? Pomocí jakého operátoru?
11. Je dáno dynamicky alokované pole `a` s prvky typu `int`, které má 20 prvků. Máte toto pole dealokovat. Jak to provedete? Pomocí jakého operátoru?
12. Máte za úkol dynamicky alokovat dvourozměrné pole `a`, které bude bude mít `R` řádků a `S` sloupců. Prvky budou typu `int`. Jak bude vypadat deklarace pointeru na takové pole? Jak se provede jeho alokace v paměti? Lze to provést jednorázovou alokací?
13. Máte za úkol uvolnit z paměti, dealokovat, dvourozměrné pole `a`, které má `R` řádků a `S` sloupců. Prvky jsou typu `int`. Jak to provedete? Lze to provést jednorázovou dealokací?

Kontrolní otázky (pokrač.)

14. Máte pointer daný následující deklarací `int* p;`. Co je na tomto pointeru konstantního a co proměnného?
15. Máte pointer daný následující deklarací `const int* p;`. Co je na tomto pointeru konstantního a co proměnného?
16. Máte pointer daný následující deklarací `int* const p;`. Co je na tomto pointeru konstantního a co proměnného?
17. Máte pointer daný následující deklarací `const int* const p;`. Co je na tomto pointeru konstantního a co proměnného?
18. Popište tři způsoby předávání parametru do funkce.
19. Do funkce `F` předáváte pole pomocí pointeru. Ve funkci `F` na takto předané pole (na parametr z hlavičky funkce) aplikujete operátor `sizeof`? Jaký výsledek dostanete?

Kontrolní otázky (pokrač.)

20. Jaké operace jsou povoleny provádět s pointery v pointerové aritmetice? Za jakého předpokladu mají tyto operace smysl? Nápověda: Mají tyto operace smysl pokud pointery ukazují na libovolné místo v paměti? Nebo musí být s něčím svázány?
21. Jak je možné chápat pole z pohledu pointeru? Co vlastně z tohoto pohledu představuje identifikátor pole?
22. Máte dáno pole `int a[10];` a ukazatel `int* p = a;`. Čemu odpovídá výraz `*(p+3)`?
23. Máte dáno pole `int a[10];` a ukazatel `int* p = a;`. Kam bude ukazovat pointer `p` po vyhodnocení výrazu `p+=2;`?
24. Máte dva pointery `int *p, *q;`. Jaký je smysl výrazu `p == q`?
25. Máte dáno pole `int a[10];` a dva ukazatele `int *p, *q;`. Oba tyto ukazatele ukazují na některý prvek v poli `a`;. Jaký je smysl výrazů `p-q` a `*p-*q`?

Děkuji za pozornost