动态内存管理

COMP250205：计算机程序设计

李昊

hao.li@xjtu.edu.cn

西安交通大学计算机学院

数据生存周期与自动变量分配

全局变量和局部变量 - 1

全局变量：定义在函数之外的变量

程序启动时自动分配，程序结束时销毁

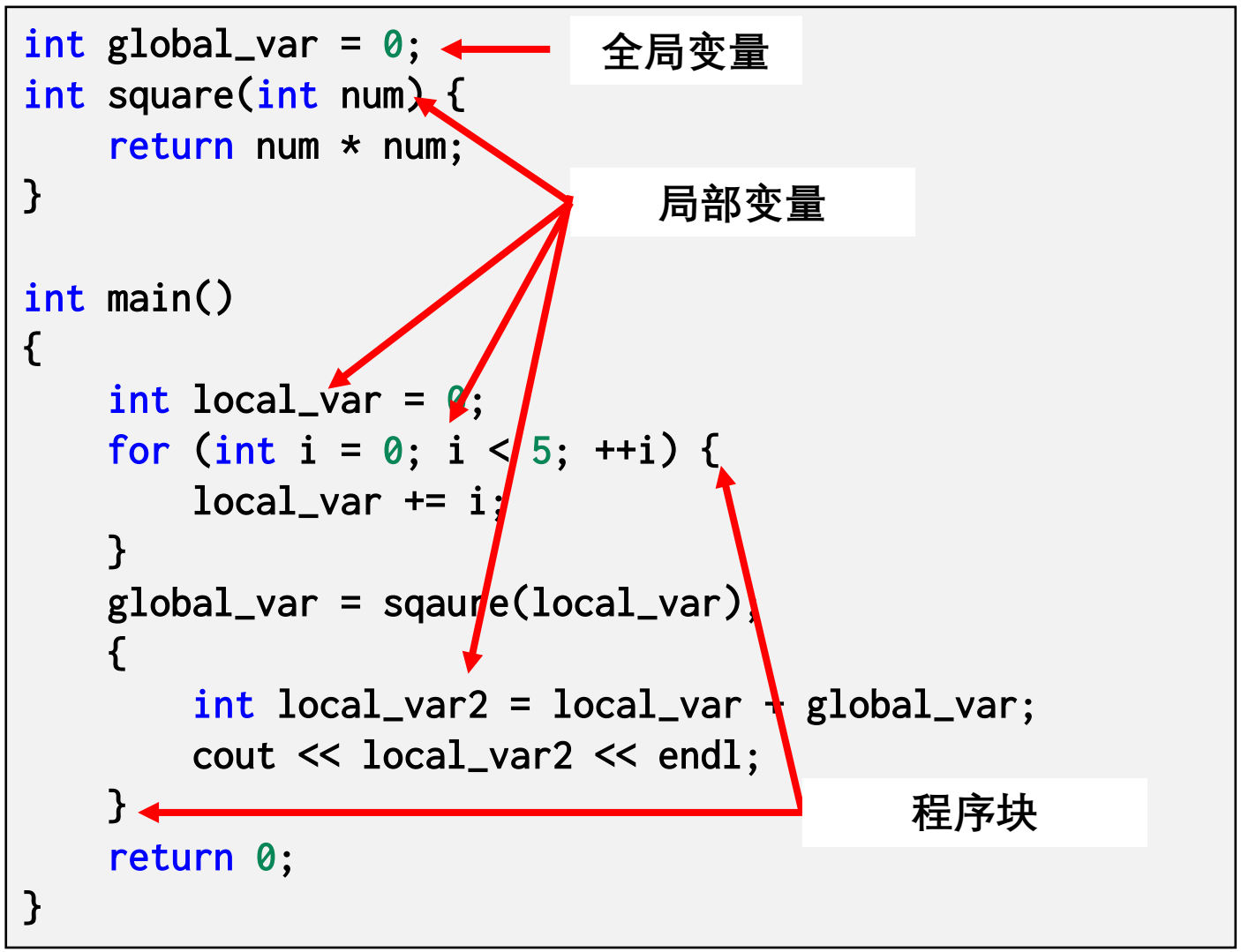
局部变量：定义在程序块内的变量

进入程序块时自动分配，离开程序块时销毁

全局变量和局部变量 - 2

```
int global_var = 0;
int square(int num) {
    return num * num;
}

int main()
{
    int local_var = 0;
    for (int i = 0; i < 5; ++i) {
        local_var += i;
    }
    global_var = sqaure(local_var);
    {
        int local_var2 = local_var + global_var;
        cout << local_var2 << endl;
    }
    return 0;
}
```



变量的存储位置

静态内存：存储全局变量、字符串常量

栈内存（stack）：存储局部变量

```
int global = 0;
int main()
{
    int local = 0;
    for (int i = 0; i < 5; ++i) {
        local += i;
    }
    {
        int local2 = local;
        global += local2;
    }
    return 0;
}
```

自动变量分配

将变量名与内存绑定，自动的分配和回收内存

这部分内存也被称为“命名内存”

优势：内存管理对程序员透明

缺陷：内存管理灵活性有限

自动变量分配的问题 - 1

无法灵活管理生存周期

```
char* hello()
{
    char hello[10] = "hello";
    return hello;
}

int main()
{
    char* p = hello()
    cout << p;
    return 0;
}
```

局部自动变量

错误！返回了一个将被回收的内存位置！

指向了一个未知的区域

使用全局变量可以解决这个问题，
但会造成内存浪费

自动变量分配的问题 - 2

无法高效使用内存

必须明确预留空间

```
int main()
{
    char name[10];
    cin >> name;
    cout << name;
    return 0;
}
```

空间不够 vs. 浪费空间

动态内存管理

动态内存分配

由程序运行时分配的内存空间：堆内存（heap）

该空间不与任何变量名绑定：匿名内存

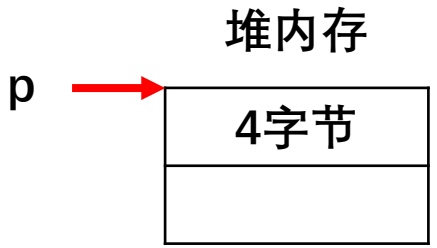
使用地址（指针）访问该空间

使用对应类型的指针指向该空间

新空间的数据类型

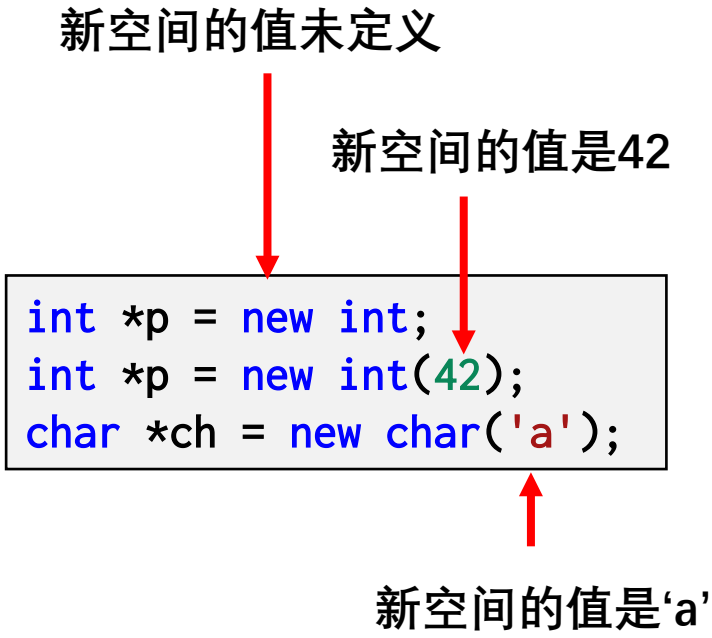
```
int *p = new int;
```

new运算符，分配一片新空间



初始化动态分配空间

动态内存的初始化方式与自动变量类似



回收动态分配的内存

动态分配的内存不会自动回收：需要显式回收

使用delete显式回收用new分配的内存

```
int main()
{
    int *p = new int(42);
    cout << *p << endl;
    delete p;
    return 0;
}
```

使用new分配了新空间

使用delete回收该片空间

使用动态分配内存

```
#include <iostream>
int* create_int()
{
    int *p = new int;
    return p;
}
int main()
{
    using namespace std;
    int *pt = create_int();
    *pt = 1001;

    cout << "some int is ";
    cout << *pt << " at location " << pt << endl;

    delete pt;
    return 0;
}
```

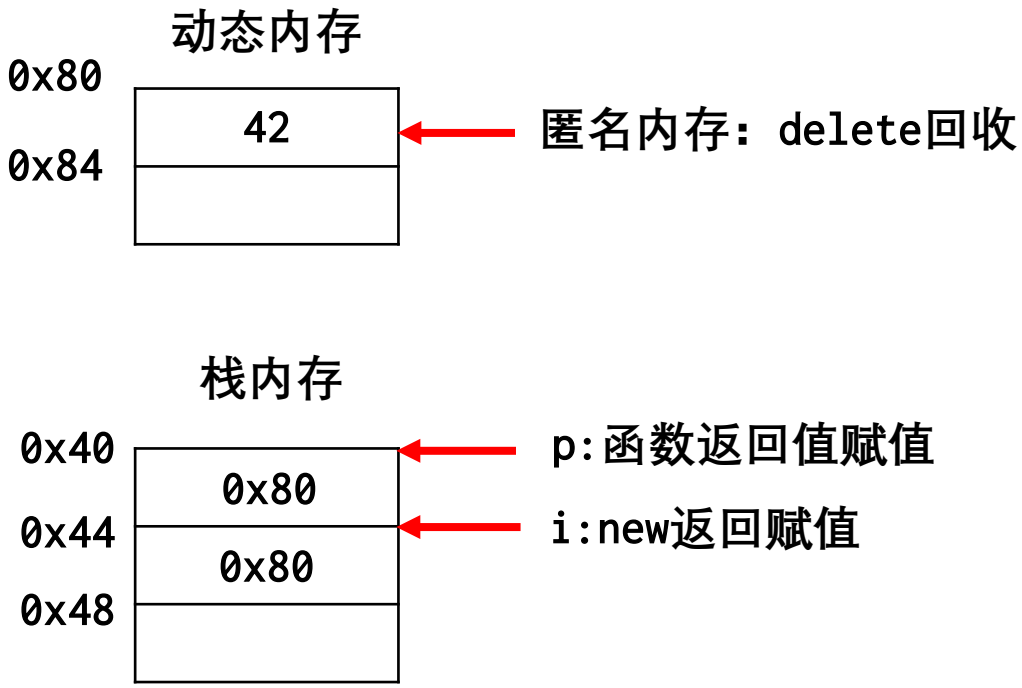
动态分配内存的生存周期

从new开始，直到delete结束

动态内存的生命周期和其指针的生命周期不同

```
int* create_int()
{
    int *i = new int(42);
    return i;
}

int main()
{
    int *p = create_int();
    cout << *p << endl;
    delete p;
    return 0;
}
```



动态分配内存的生存周期 – QUIZ – 1

使用new分配了整型空间

错误！下一次循环时，之前分配的空间将丢失！

只回收了最后一次指向的空间

```
int* create_int(int var)
{
    int *p = new int(var);
    return p;
}

int main()
{
    int *ps;
    for (int i=0; i<5; ++i) {
        ps = create_int(i);
        cout << *ps << endl;
    }
    delete ps;
}
```

动态分配内存的生存周期 – QUIZ – 2

```
int* create_int()
{
    int *p = new int();
    return p;
}
int main()
{
    int *num1 = create_int();
    int *num2 = create_int();
    cin >> num1 >> num2;
    if (*num1 > *num2) {
        num2 = num1;
    }
    cout << "the larger number is " << *num2;
    delete num1;
    delete num2;
    return 0;
}
```

num2原来分配的空间丢失!

同一片空间回收两次!
double free!

动态分配内存的生存周期 – QUIZ – 3

```
int main()
{
    int *pa[10] = {nullptr};
    for (int i=0; i<10; ++i) {
        if (!pa[i]) {
            pa[i] = create_int(i);
        }
        if (i % 2 == 0) {
            cout << *pa[i] << endl;
            delete pa[i];
        }
    }
    for (int i=0; i<10; ++i) {
        if (!pa[i]) continue;
        cout << *pa[i] << endl;
        delete pa[i];
    }
    return 0;
}
```

回收了偶数下标空间

加入pa[i] = nullptr;

错误！回收后指针不一定是nullptr！

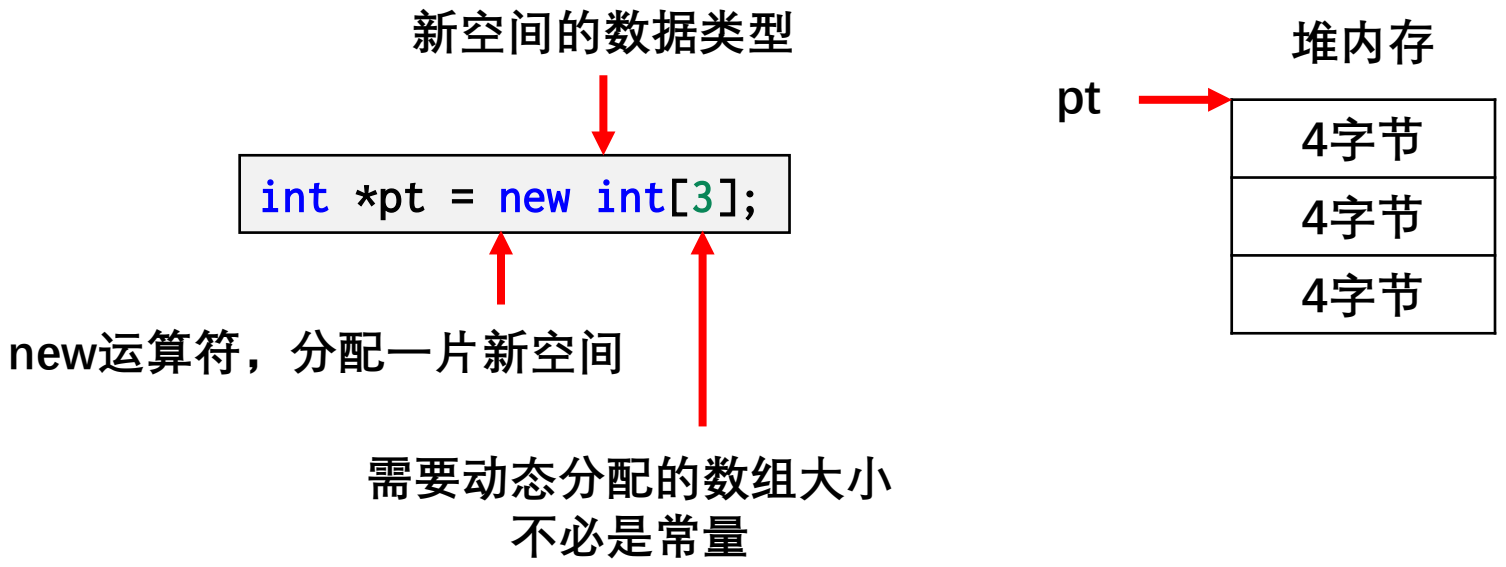
有可能导致二次回收

动态数组

创建动态数组

动态分配一组大小一致的空间

使用new分配空间，返回指向第一个元素的指针



回收动态数组

动态数组使用delete []来回收

单一整型空间 →
动态数组空间 →

回收单一整型空间 →
注意！也仅回收单一整型空间 →

显式回收整个数组 →

```
int *pt = new int;
int *ps = new int[500];

delete pt;
delete ps;

delete [] ps;
```

new和delete的使用方式要对应一致

使用动态数组

接收3个键盘输入的字符串，存储在动态数组中

```
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
    char *strs[3];
    for (int i = 0; i < 3; ++i) {
        char str[100];
        cin >> str;
        strs[i] = new char[strlen(str)];
        strcpy(strs[i], str);
        cout << strs[i] << endl;
    }
    return 0;
}
```

临时存储在栈上

分配堆空间

拷贝字符串内容

错误！内存没有回收！

错误！分配空间不够！

使用动态数组（进阶）

在上面的程序中，如果单次输入超过100怎么办?

无论准备多大的缓冲区，都有可能超过

考虑到输入可能来自文件、网络数据流

内容总结

动态内存分配：运行时在堆上分配的空间

- 匿名内存，仅返回空间地址，使用指针访问
- 必须显式回收，并且分配一次回收一次
- 注意内存泄漏、悬挂指针、二次回收问题
- new和delete必须配对

动态数组：一次分配多个连续同类型空间

- 使用new []分配，必须使用delete []回收