CS 1502H 程序设计思想与方法 (C++)
Chapter 8. 结构体

陈东尧
上海交通大学
2025年秋季

```
int main()
{
	int **a, i, j, k = 0;
	a = new int *[3]; // 申请指针数组

	// 申请每一行空间
	for (i = 0; i < 3; ++i)
		a[i] = new int[4];
	for (i = 0; i < 3; ++i){ // 二维数组赋值
		for (j = 0; j < 4; ++j)
			a[i][j] = k++;
	}
	for (i = 0; i < 3; ++i) { // 二维数组输出
		cout << endl;
		for (j = 0; j < 4; ++j)
			cout << a[i][j] << '\t';
	}
	for (i = 0; i < 3; ++i) // 二维数组空间释放
		delete [] a[i];
	delete [] a;
	return 0;
}
```

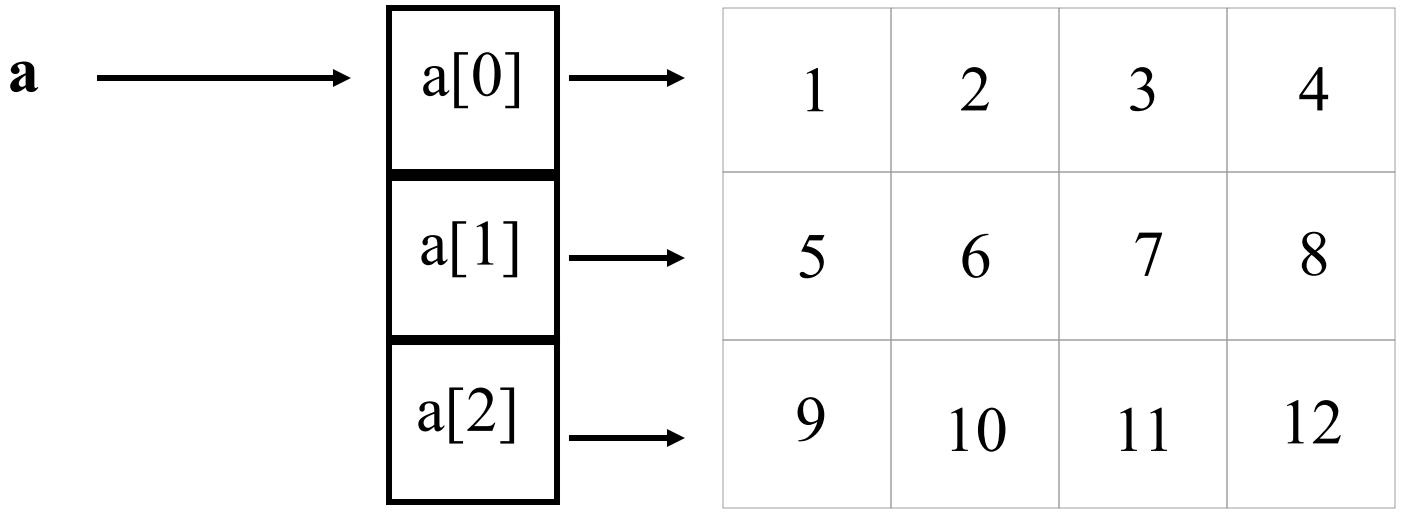
回顾

- 指针与数组的关系。
- 动态分配内存；堆和栈的概念。
- 用指针表示字符串及其相关操作。
- 指针与函数；将指针作为形式参数可以使一个函数与其调用函数共享数据。
- 引用类型作为函数的参数或返回值。
- 指针数组、多级指针与多维数组之间的关系。

回顾

- `int **a; a = new int*[3]; // 动态定义了3个指针变量`

  `for(int i=0; i<3; ++i) a[i] = new int[4];`



引用是如何在底层实现的？

```
void swap2(int *a,int *b)
{
    int temp;
     temp = *a;
     *a = *b;
     *b = temp;
}
```

```
void swap3(int &a,int &b)
{
      int temp;
      temp  = a;
      a = b;
      b = temp;
}
```

在引用的概念上，我们可以把 **int& r = x;** 理解为： **int* const r = &x;**

这可以解释引用的**很多性能**，比如：

- 为何引用不可以重定向，因此也可以实现隐式解引用（例如不需要写作*r）
- 为何引用可以起到指针专递的作用
- 不存在空的引用

为什么要学结构体？

- 学生成绩单

学号	姓名	数学	英语	程序设计
00001	张三	**96**	**94**	**88**
00002	李四	**89**	**70**	**76**
00003	王五	**90**	**87**	**78**

为什么要学结构体？

- 学生成绩单

学号	姓名	数学	英语	程序设计
00001	张三	**96**	**94**	**88**
00002	李四	**89**	**70**	**76**
00003	王五	**90**	**87**	**78**

统计每个学生的总分，如何在程序中表示学生信息？

可选方案探讨

int a[3][4];

1	**2**	**3**	**4**
5	**6**	**7**	**8**
9	**10**	**11**	**12**

scores[3][5];

00001	张三	**96**	**94**	**88**
00002	李四	**89**	**70**	**76**
00003	王五	**90**	**87**	**78**

可选方案探讨

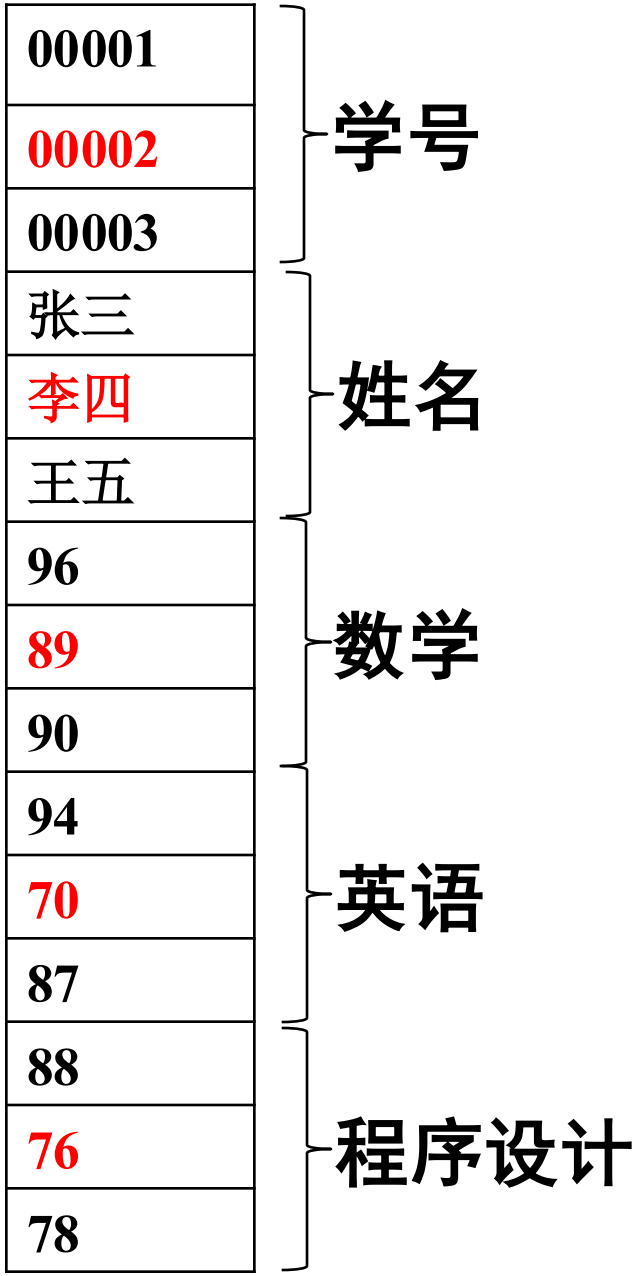
五个数组

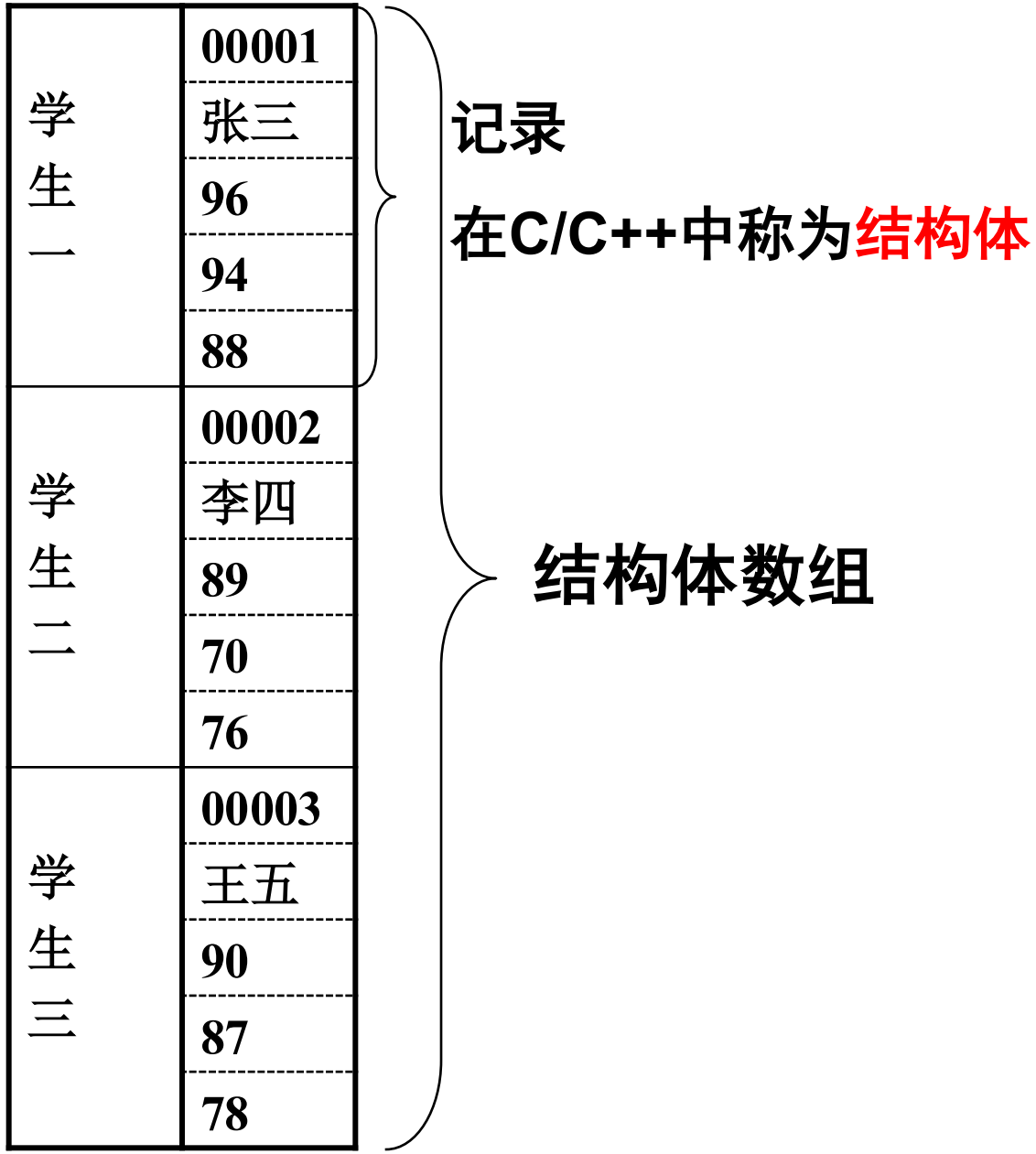
```
char id[3][14];
char name[3][12];
int math[3];
int english[3];
int programming[3];
```

可选方案探讨

五个数组

```
char id[3][14];
char name[3][12];
int math[3];
int english[3];
int programming[3];
```




学生一
00001
张三
96
94
88
学生二
00002
李四
89
70
76
学生三
00003
王五
90
87
78
记录
在C/C++中称为结构体
结构体数组

数据的封装-结构体

- 结构体的概述
- 结构体类型的定义
- 结构体类型的变量
- 结构体数组
- 结构体作为函数的参数

为什么要用结构体

- 将一个人的所有信息项放在一起，即保持相关性。

学号	姓名	数学	英语	程序设计
00001	张三	96	94	88
00002	李四	89	70	76
00003	王五	90	87	78

结构体类型作用

- 允许把一些分量聚合成一个整体，用一个变量表示
- 一个结构体的各个分量都有名字，把这些分量称为成员(member)
- 结构体的成员可以是各种类型，程序员能创建适合于问题的数据聚合

数据的封装-结构体

- 结构体的概述
- **结构体类型的定义**
- 结构体类型的变量
- 结构体数组
- 结构体作为函数的参数

结构体类型的定义

- 定义结构体类型中的成员
- 类型声明格式：

struct 结构体类型名{

成员声明；

}；

```
struct  studentT {
    char  id[14];
    char  name[12];
    int math;
    int english;
    int programming;
};
```

结构体类型的定义

- 定义结构体类型中的成员
- 类型声明格式：

  struct 结构体类型名{

  成员声明；

  } ;

注意分号不要漏！

```
struct  studentT {
    char  id[14];
    char  name[12];
    int math;
    int english;
    int programming;
 };
```

结构体成员的类型

- 可以是任何类型

```
struct studentT {
    char  id[14];
    char  name[12];
    int math;
    int english;
    int programming;
 };
```

```
struct dateT {
    int month;
    int day;
    int year;
};
```

```
struct student2T {
    char  id[14];
    char  name[12];
    dateT birthday;
    int math;
    int english;
    int programming;
 };
```

结构体成员的类型

- 可以是任何类型

```
struct studentT {
    char  id[14];
    char  name[12];
    int math;
    int english;
    int programming;
};
```

```
struct dateT {
    int month;
    int day;
    int year;
};
```

可以嵌套

```
struct student2T {
    char  id[14];
    char  name[12];
    dateT birthday;
    int math;
    int english;
    int programming;
};
```

结构体成员的类型

```
struct studentT {
    char  id[14];
    char  name[12];
    int math;
    int english;
    int programming;
 };
```

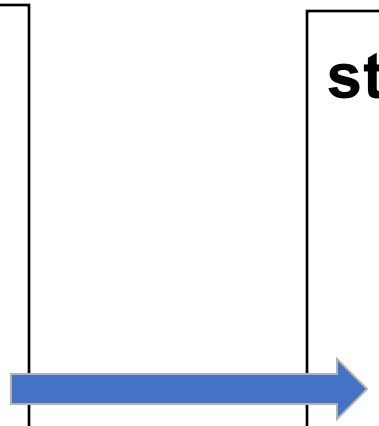
```
struct studentT {
    char  id[14];
    char  name[12];
    int math;
    int english;
    int programming;
    studentT  subStu;
 };
```

递归嵌套是不合法的

结构体成员的类型

- 20门课程（大学物理、工程导论、程序设计……）

```
struct studentT {
    char  id[14];
    char  name[12];
    int math;
    int english;
    int programming;
};
```



```
struct studentT {
    char  id[14];
    char  name[12];
    int scores[20];
};
```

数据的封装-结构体

- 结构体的概述
- 结构体类型的定义
- **结构体类型的变量**
- 结构体数组
- 结构体作为函数的参数

数据的封装-结构体

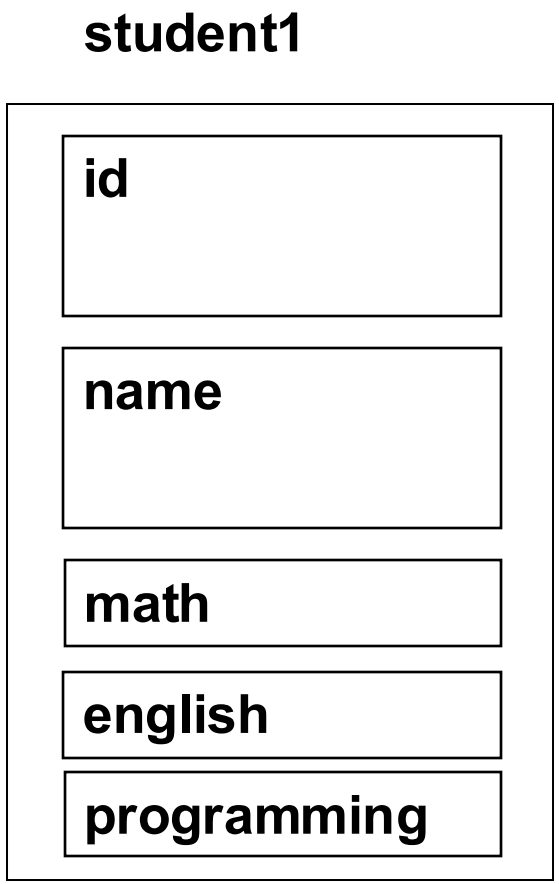
- 结构体的概述
- 结构体类型的定义
- **结构体类型的变量**
  - **结构体变量的定义→结构体类型的使用→指向结构体的指针→动态分配结构体的空间**
- 结构体数组
- 结构体作为函数的参数

结构体变量的定义

结构体类型　　变量名

studentT student1;

一旦定义了一个结构体类型的变量，系统在分配内存时就会**分配一块连续的空间**，依次存放它的每一个分量。这块空间总的名字就是结构体变量的名字，内部还有各个分量的名字 。



结构体变量的定义

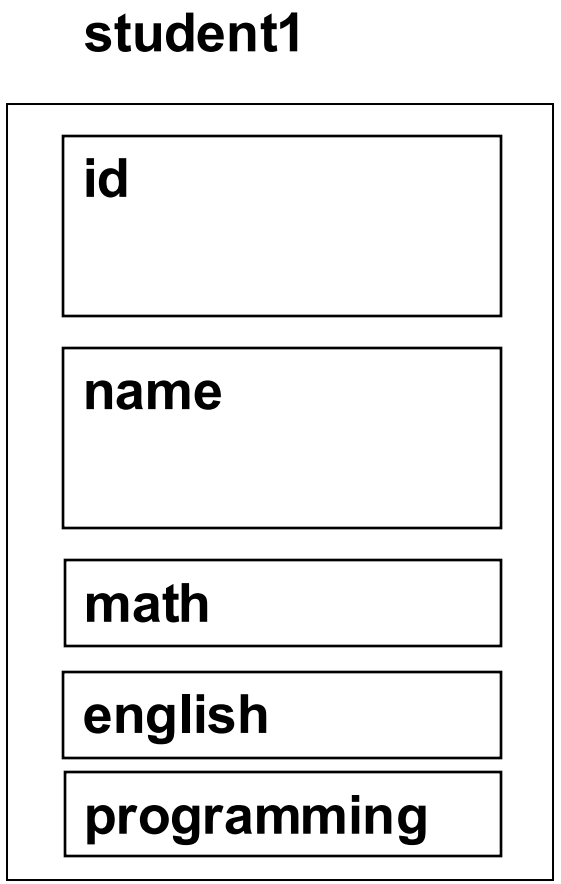
结构体类型　　变量名

studentT student1;

一旦定义了一个结构体类型的变量，系统在分配内存时就会**分配一块连续的空间**，依次存放它的每一个分量。这块空间总的名字就是结构体变量的名字，内部还有各个分量的名字 。

```
struct studentT {
    char  id[14];
    char  name[12];
    int math;
    int english;
    int programming;
 };
```

sizeof(student1)，应该有多大（bytes）？



结构体变量的定义

结构体类型　　变量名

studentT student1;

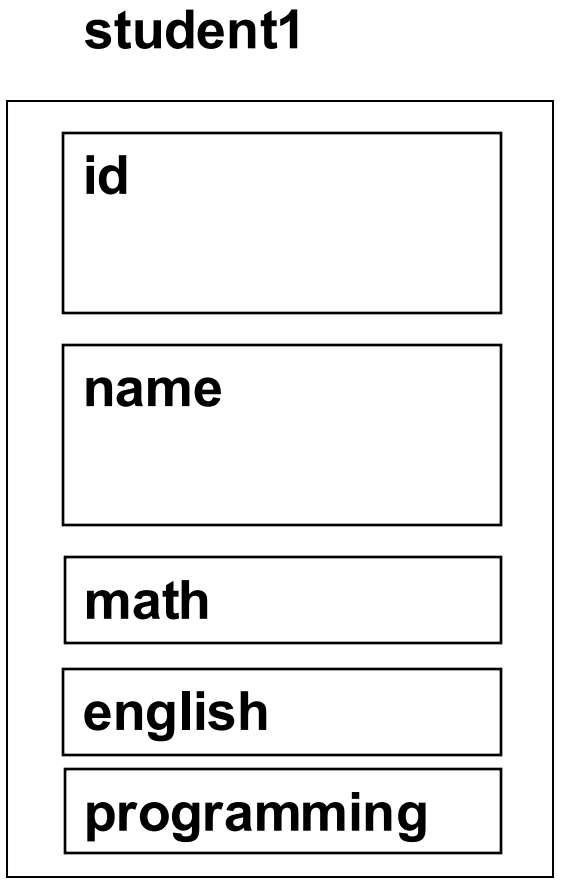
一旦定义了一个结构体类型的变量，系统在分配内存时就会**分配一块连续的空间**，依次存放它的每一个分量。这块空间总的名字就是结构体变量的名字，内部还有各个分量的名字 。

```
struct studentT {
    char  id[14];
    char  name[12];
    int math;
    int english;
    int programming;
 };
```

sizeof(student1)，应该有多大（bytes）？

14 + 12 + 4 + 4 + 4

=38



结构体变量的定义

结构体类型　　变量名

studentT student1;

一旦定义了一个结构体类型的变量，系统在分配内存时就会**分配一块连续的空间**，依次存放它的每一个分量。这块空间总的名字就是结构体变量的名字，内部还有各个分量的名字 。

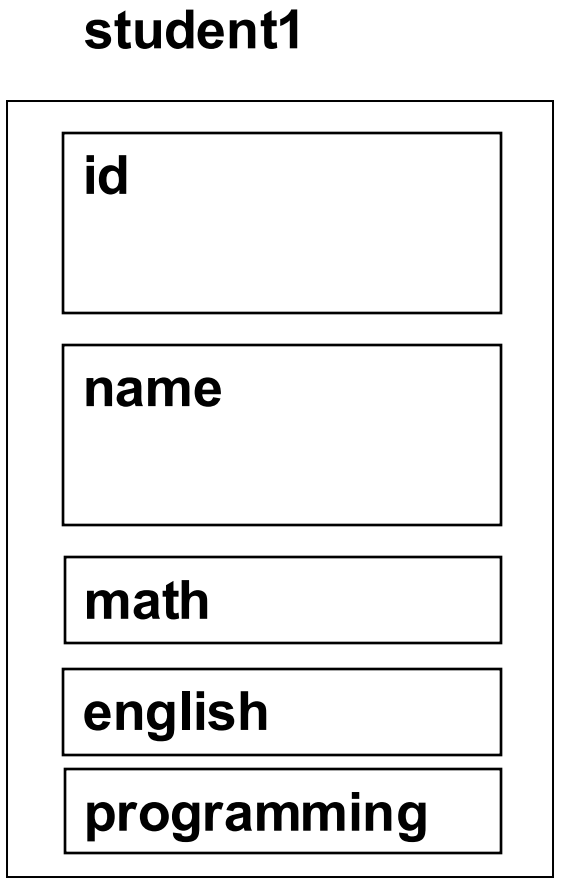
```
struct studentT {
    char  id[14];
    char  name[12];
    int math;
    int english;
    int programming;
 };
```

sizeof(student1)，应该有多大（bytes）？

14 + 12 + 4 + 4 + 4

= 38

你可能会发现不是这样的，很有可能是40



结构体变量在内存中的对齐和填充（Alignment and Padding）

```
struct  studentT {
    char id[14];
    char name[12];
    int math;
    int english;
    int programming;
 };

sizeof (studentT) == 40
//14对齐了16
```

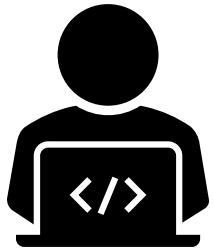
```
struct  studentT {
    char id[11];
    char name[11];
    int math;
    int english;
    int programming;
 };

sizeof (studentT) == 36
//两个11都向12对齐
```

int值被对齐在4字节地址边界，保证了一个int型数总能通过一次内存操作被访问到，每次内存访问是在4字节对齐的地址处读取。

【扩展阅读】C/C++中的结构体内存对齐（用于加快内存读取速度）
https://www.geeksforgeeks.org/c/structure-member-alignment-padding-and-data-packing/

结构体变量初始化

```
studentT s1 = {"00001", "张三" , 87, 90, 77};
studentT s1 = {.name = "Lee", .math = 100};
```



代码演示

chapter8-struct-initialization.cpp

数据的封装-结构体

- 结构体的概述
- 结构体类型的定义
- **结构体类型的变量**
  - **结构体变量的定义→结构体类型的使用→指向结构体的指针→动态分配结构体的空间**
- 结构体数组
- 结构体作为函数的参数

结构体变量的访问

- 访问它的成员

结构变量名.成员名

如：student1.name

- 如结构中还有结构，则一级一级用”.”分开

如：student1.birthday.year

结构变量的赋值

- 通过对它的每一个成员的赋值而实现

  如：输入student1的内容可用：

```
cin >> student1.id
   >> student1.name
   >> student1.math
   >> student1.english
  >> student1.programming;
```

结构变量的赋值

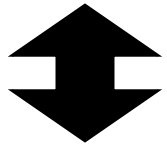
- 同类型的结构变量之间可以相互赋值，如

```
student1 = student2;//将student2的成员对应赋给student1的成员
```

结构变量的赋值

- 同类型的结构变量之间可以相互赋值，如

student1 = student2;//将student2的成员对应赋给student1的成员

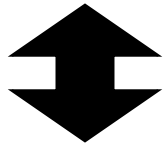


```
strcpy(student1.id, student2.id);
strcpy(student1.name,student2.name);
student1.math = student2.math;
student1.english = student2.english;
student1.programming = student2. programming;
```

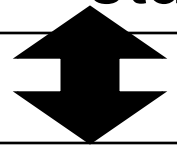
结构变量的赋值

- 同类型的结构变量之间可以相互赋值，如

student1 = student2;//将student2的成员对应赋给student1的成员



```
strcpy(student1.id, student2.id);
strcpy(student1.name,student2.name);
student1.math = student2.math;
student1.english = student2.english;
student1.programming = student2. programming;
```



```
memcpy(&student1, &student2, sizeof(studentT));
```

结构变量的输出

- 通常是通过输出它的每一个成员而实现

  如：输出student1的内容可用：

```
cout << student1.id
     << student1.name
     << student1.math
     << student1.english
     << student1.pogramming;
```

数据的封装-结构体

- 结构体的概述
- 结构体类型的定义
- **结构体类型的变量**
  - **结构体变量的定义→结构体类型的使用→指向结构体的指针→动态分配结构体的空间**
- 结构体数组
- 结构体作为函数的参数

指向结构的指针

- 直接定义指针变量　　studentT *sp;
- 给结构体指针赋值　　sp = &student1;

结构体指针的访问：

***sp.name是否正确？**

指向结构的指针

- 直接定义指针变量　　studentT *sp;
- 给结构体指针赋值　　sp = &student1;

结构体指针的访问：

***sp.name 错误！**

指向结构的指针

- 直接定义指针变量　　studentT *sp;
- 给结构体指针赋值　　sp = &student1;

结构体指针的访问：

***sp.name 错误** ✗

(*指针).成员　如：(*sp).name

指针->成员　如：sp->name

} 都可以输出**student1.name**

指向结构的指针

- 直接定义指针变量　　studentT *sp;
- 给结构体指针赋值　　sp = &student1;

结构体指针的访问：

***sp.name 错误！**

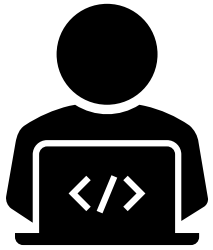
(*指针).成员　如：(*sp).name

指针->成员　如：sp->name

} 都可以输出**student1.name**

通常使用第二种方法

代码实例分析



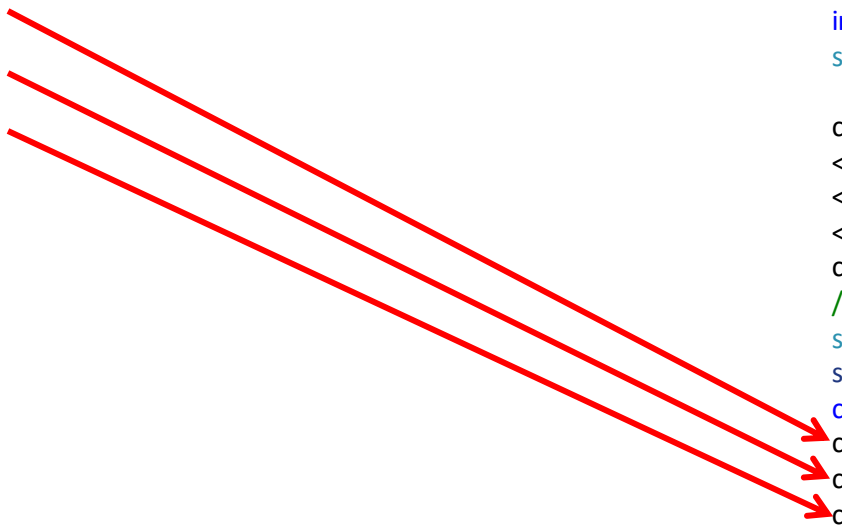
代码演示

chapter8-struct-initialization.cpp

```
struct studentT {
char id[14];
char name[8];
int math;
int english;
int programming;
};
struct tempT {
char id[5];
char name[5];
int math;
int english;
};

int main(){
studentT s1 = {.name = "Lee", .math = 100};

cout<<s1.name<<endl
<<s1.id<<endl
<<s1.math<<endl
<<s1.english<<endl;
cout<<"Size is:"<<sizeof(studentT)<<endl;
// 结构体指针赋值：
studentT *sp;
sp = &s1;
char c[] = {'a','b','d','\0'};
cout<<sp->name<<endl; //
cout<<(*sp).name<<endl; //
cout<< *sp->name<<endl; //!!!!! Possible test problem

//比较和下面代码输出的异同，分析原因
cout<<c<<endl;
cout<<*c<<endl;

//下面两个都是错误的代码，试分析原因
cout<< sp.name<<endl;//xxxx
cout<< (*sp)->name<<endl;//xxxx
return 0;
}
```

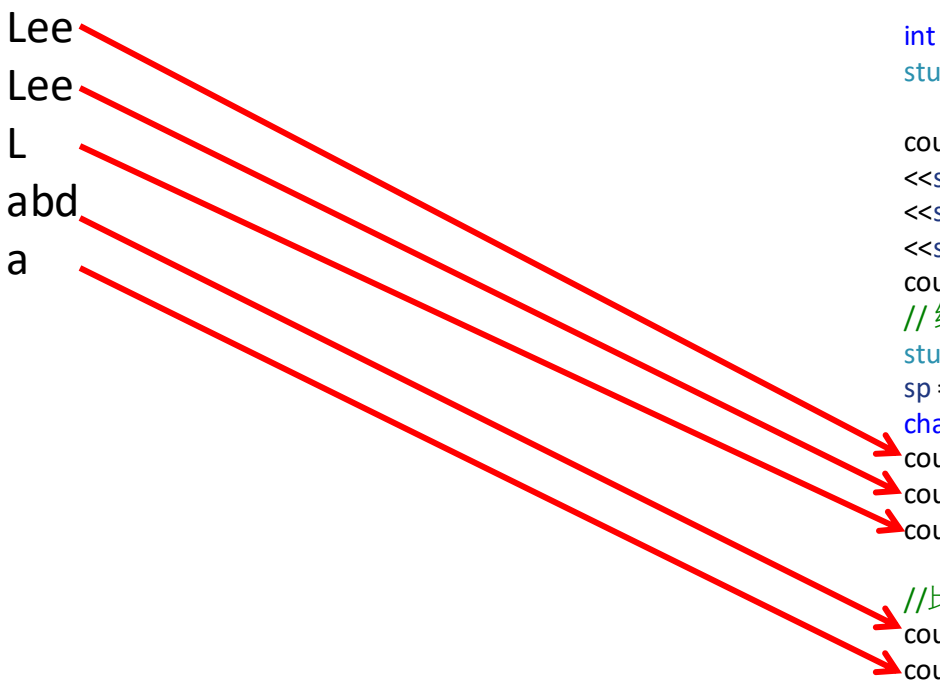
代码实例分析

Lee
Lee
L



```
struct studentT {
char id[14];
char name[8];
int math;
int english;
int programming;
};
struct tempT {
char id[5];
char name[5];
int math;
int english;
};

int main(){
studentT s1 = {.name = "Lee", .math = 100};

cout<<s1.name<<endl
<<s1.id<<endl
<<s1.math<<endl
<<s1.english<<endl;
cout<<"Size is:"<<sizeof(studentT)<<endl;
// 结构体指针赋值：
studentT *sp;
sp = &s1;
char c[] = {'a','b','d','\0'};
cout<<sp->name<<endl; //
cout<<(*sp).name<<endl; //
cout<< *sp->name<<endl; //!!!!! Possible test problem

//比较和下面代码输出的异同，分析原因
cout<<c<<endl;
cout<<*c<<endl;

//下面两个都是错误的代码，试分析原因
cout<< sp.name<<endl;//xxxx
cout<< (*sp)->name<<endl;//xxxx
return 0;
}
```

代码实例分析




```
struct studentT {
char id[14];
char name[8];
int math;
int english;
int programming;
};
struct tempT {
char id[5];
char name[5];
int math;
int english;
};

int main(){
studentT s1 = {.name = "Lee", .math = 100};

cout<<s1.name<<endl
<<s1.id<<endl
<<s1.math<<endl
<<s1.english<<endl;
cout<<"Size is:"<<sizeof(studentT)<<endl;
// 结构体指针赋值：
studentT *sp;
sp = &s1;
char c[] = {'a','b','d','\0'};
cout<<sp->name<<endl; //
cout<<(*sp).name<<endl; //
cout<< *sp->name<<endl; //!!!!! Possible test problem

//比较和下面代码输出的异同，分析原因
cout<<c<<endl;
cout<<*c<<endl;

//下面两个都是错误的代码，试分析原因
cout<< sp.name<<endl;//xxxx
cout<< (*sp)->name<<endl;//xxxx
return 0;
}
```

代码实例分析

```
struct studentT {
char id[14];
char name[8];
int math;
int english;
int programming;
};
struct tempT {
char id[5];
char name[5];
int math;
int english;
};

int main(){
studentT s1 = {.name = "Lee", .math = 100};

cout<<s1.name<<endl
<<s1.id<<endl
<<s1.math<<endl
<<s1.english<<endl;
cout<<"Size is:"<<sizeof(studentT)<<endl;
// 结构体指针赋值：
studentT *sp;
sp = &s1;
char c[] = {'a','b','d','\0'};
cout<<sp->name<<endl; //
cout<<(*sp).name<<endl; //
cout<< *sp->name<<endl; //!!!!! Possible test problem

//比较和下面代码输出的异同，分析原因
cout<<c<<endl;
cout<<*c<<endl;

//下面两个都是错误的代码，试分析原因
cout<< sp.name<<endl;//xxxx
cout<< (*sp)->name<<endl;//xxxx
return 0;
}
```

Lee
Lee
L
abd
a

error: member reference type 'studentT *' is a pointer; did you mean to use '->'?

error: member reference type 'studentT' is not a pointer; did you mean to use '.'?

数据的封装-结构体

- 结构体的概述
- 结构体类型的定义
- **结构体类型的变量**
  - **结构体变量的定义→结构体类型的使用→指向结构体的指针→动态分配结构体的空间**
- 结构体数组
- 结构体作为函数的参数

动态分配结构体的空间

- 指向结构体指针的另一种用法是存储动态申请到的内存的首地址。用法和申请普通的动态变量一样。如：

```
studentT  *sp;

sp = new studentT;

sp->math = 95;
```

数据的封装-结构体

- 结构体的概述
- 结构体类型的定义
- 结构体类型的变量
- **结构体数组**
- 结构体作为函数的参数

结构体数组

- 用于描述个体的集合
- 定义格式：

```
studentT   studentArray[30];

const int SIZE = 30;
studentT   studentArray[SIZE];
```

结构体数组的使用

- 使用数组的某一成员的成员

  studentArray[3].name

- 数组成员之间相互赋值

  studentArray[4] = studentArray[2]

- 结构数组的初始化

  studentT studentArray[3] = { {“00001”, “张三”, 80, 90,98 }, {“00002”, “李四”, 89, 70,76 }};

示例：录入学生信息

```cpp
#include <iostream>
using namespace std;

const int SIZE = 30;   // 学生人数
const int COURSES = 5; // 课程数量

struct studentT  // 声明结构体
{
    char no[14];   // 学号
    char name[12]; // 姓名
    int scores[COURSES]; // 各科成绩
};
```

手动录入信息

```cpp
int main()
{
    studentT students[SIZE];  // 定义结构体数组
    int total[SIZE] = {0};    // 定义存放总分数组

    for(int i = 0; i < SIZE; ++i) // 输入学生成绩
    {
        cout << "No Name: ";
        cin >> (1) ;      // 输入学号
        cin >> (2) ;      // 输入姓名
        cout << "Scores: ";
        for(int j = 0; j < COURSES; ++j)
            cin >> (3) ; // 输入课程成绩
        cout << endl;
    }

    for(int i = 0; i < SIZE; ++i) // 计算总分
    {
        for(int j = 0; j < COURSES; ++j)
            total[i] += (4) ;
    }
```

输出总分

```cpp
    cout << "No\tName\tTotal" << endl; // 输出总分
    for(int i =0; i < SIZE; ++i)
    {
        cout << (5) << "\t"          // 输出学号
             << (6) << "\t"          // 输出姓名
             << (7) << endl;         // 输出总分
    }
    return 0;
}
```

示例：录入学生信息

```cpp
#include <iostream>
using namespace std;

const int SIZE = 30;   // 学生人数
const int COURSES = 5; // 课程数量

struct studentT  // 声明结构体
{
    char no[14];   // 学号
    char name[12]; // 姓名
    int scores[COURSES]; // 各科成绩
};
```

手动录入信息

```cpp
int main()
{
    studentT students[SIZE];  // 定义结构体数组
    int total[SIZE] = {0};    // 定义存放总分数组

    for(int i = 0; i < SIZE; ++i) // 输入学生成绩
    {
        cout << "No Name: ";
        cin >> students[i].no;     // 输入学号
        cin >> students[i].name;   // 输入姓名
        cout << "Scores: ";
        for(int j = 0; j < COURSES; ++j)
            cin >> students[i].scores[j]; // 输入课程成绩
        cout << endl;
    }

    for(int i = 0; i < SIZE; ++i) // 计算总分
    {
        for(int j = 0; j < COURSES; ++j)
            total[i] += students[i].scores[j];
    }
```

输出总分

```cpp
    cout << "No\tName\tTotal" << endl; // 输出总分
    for(int i =0; i < SIZE; ++i)
    {
        cout << students[i].no << "\t"          // 输出学号
             << students[i].name << "\t"        // 输出姓名
             << total[i] << endl;               // 输出总分
    }
    return 0;
}
```

数据的封装-结构体

- 结构体的概述
- 结构体类型的定义
- 结构体类型的变量
- 结构体数组
- **结构体作为函数的参数**

函数参数 --- swap_num

```
int  main()
{
    int x = 4, y = 6;
    函数调用;
    cout  << “x=”  << x  <<“ y=” <<y<<endl;
    return 0;
 }
```

程序运行结果:

x=6 y=4

```
void swap1(int a,int b)
{
    int temp;
    temp  = a;
    a = b;
    b = temp;
}
```

```
void swap2(int *a,int *b)
{
   int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
```

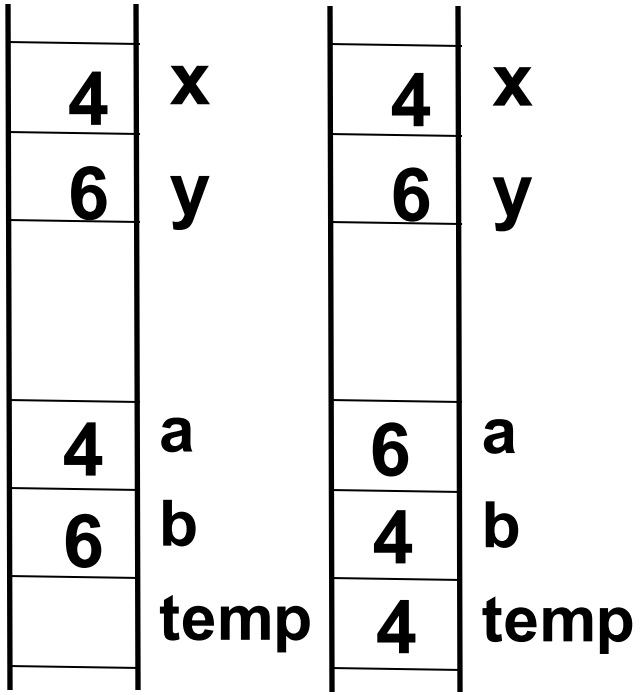
```
void swap3(int &a,int &b)
{
     int temp;
    temp  = a;
    a = b;
    b = temp;
}
```

```
int  main()
{   int x = 4, y = 6;
    swap1(x,y);
    cout  << "x="  << x  <<" y=" <<y<<endl;
    return 0;
 }
```

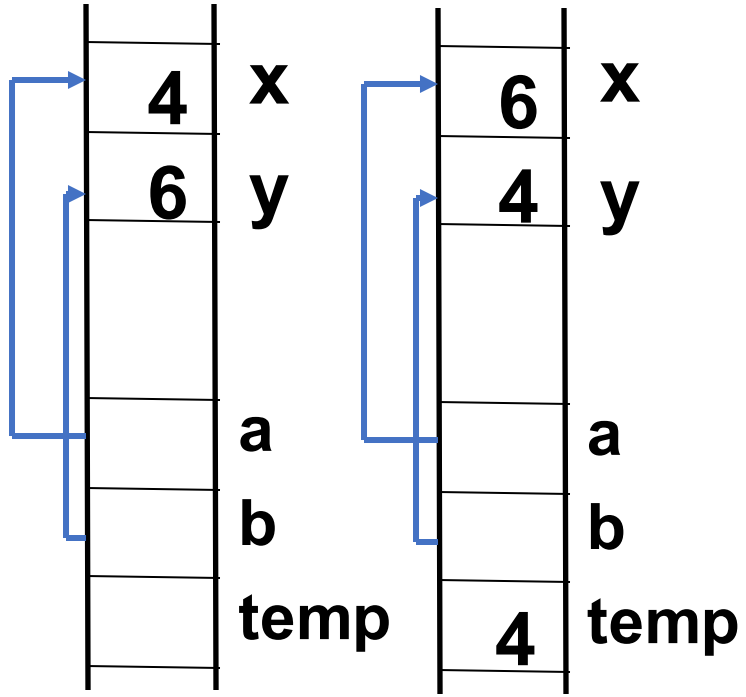
```
swap2(&x,&y);
```

```
swap3(x,y);
```

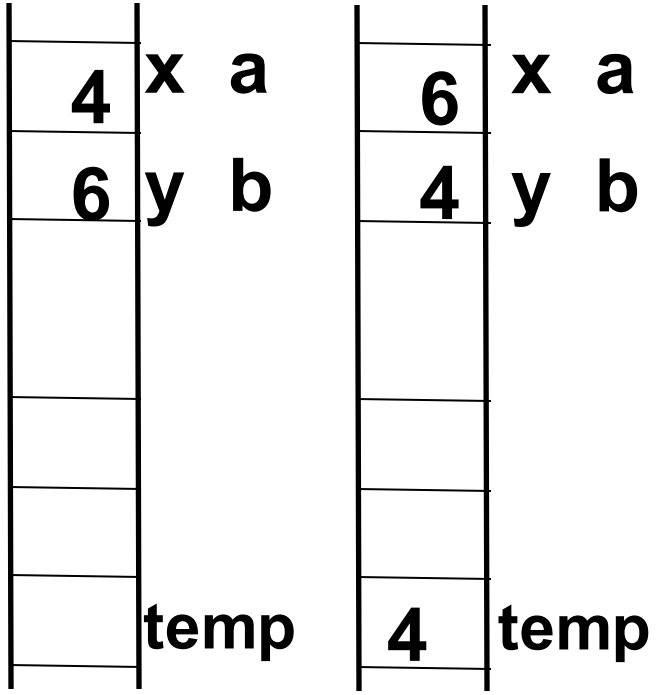
```
void swap1(int a,int b)
{
      int temp;
      temp  = a;
      a = b;
      b = temp;
}
```




```
void swap2(int *a,int *b)
{
      int temp;
      temp = *a;
      *a = *b;
      *b = temp;
}
```




```
void swap3(int &a,int &b)
{
      int temp;
      temp  = a;
      a = b;
      b = temp;
}
```



结构体作为参数传递

- 尽管结构体和数组一样也有许多分量组成，但结构体的传递和普通内置类型(int/double/char/bool)是一样的。
- 它是将实际参数中的每个分量复制到形式参数的每个分量中。

输入学生信息的示例

```
#include <iostream>
using namespace std;

const int SIZE = 2;
const int COURSES = 3;

struct studentT
{
        char id[14]; // 学号
        char name[12]; //姓名
        int scores[COURSES]; // 成绩
};
```

如何通过函数来模块化输入学生的信息？

```
int main() {
studentT students[SIZE];
int total[SIZE] = {0};

// Input students info
for (int i = 0; i < SIZE; i++){
        cout<<"student number and name:";
        cin>>students[i].id;
        cin>>students[i].name;
        cout << "Scores: ";
        for(int j = 0; j < COURSES; j++)
                cin >> students[i].scores[j];
        cout<<endl;
}

// Calculate total scores
for (int i = 0; i < SIZE; i++)
        for (int j = 0; j < COURSES; j++)
                total[i] += students[i].scores[j];

// Print students' info
cout<<"Number\tName\tTotal" <<endl;
for(int i = 0; i < SIZE; i++){
        cout<<students[i].id << "\t"
        <<students[i].name << "\t"
        <<total[i]<<endl;
}

return 0;
}
```

代码演示

chapter8-struct-reference

student.cpp

输入学生信息的示例

```
#include <iostream>
using namespace std;

const int SIZE = 2;
const int COURSES = 3;

struct studentT
{
        char id[14]; // 学号
        char name[12]; //姓名
        int scores[COURSES]; // 成绩
};
```

如何通过函数来模块化输入学生的信息？
例如设计：

inputStudent(students[i])

```
int main() {
studentT students[SIZE];
int total[SIZE] = {0};

// Input students info
for (int i = 0; i < SIZE; i++){
        cout<<"student number and name:";
        cin>>students[i].id;
        cin>>students[i].name;
        cout << "Scores: ";
        for(int j = 0; j < COURSES; j++)
                cin >> students[i].scores[j];
        cout<<endl;
}

// Calculate total scores
for (int i = 0; i < SIZE; i++)
        for (int j = 0; j < COURSES; j++)
                total[i] += students[i].scores[j];

// Print students' info
cout<<"Number\tName\tTotal" <<endl;
for(int i = 0; i < SIZE; i++){
        cout<<students[i].id << "\t"
        <<students[i].name << "\t"
        <<total[i]<<endl;
}

return 0;
}
```

代码演示

chapter8-struct-reference
student.cpp

inputStudent函数设计

- 方案一：值传递

```
void inputStudent(studentT s)
{
    cout << "No Name: ";
    cin >> s.no;
    cin >> s.name;
    cout << "Scores: ";
    for(int j = 0; j < COURSES; ++j)
        cin >> s.scores[j];
    cout << endl;
}
```

- 方案二：引用传递

```
void inputStudent(studentT &s)
{
    cout << "No Name: ";
    cin >> s.no;
    cin >> s.name;
    cout << "Scores: ";
    for(int j = 0; j < COURSES; ++j)
        cin >> s.scores[j];
    cout << endl;
}
```

哪个可行？

inputStudent函数设计

- 方案一：值传递

```
void inputStudent(studentT s)
{
    cout << "No Name: ";
    cin >> s.no;
    cin >> s.name;
    cout << "Scores: ";
    for(int j = 0; j < COURSES; ++j)
        cin >> s.scores[j];
    cout << endl;
}
```

错误

- 方案二：引用传递

```
void inputStudent(studentT &s)
{
    cout << "No Name: ";
    cin >> s.no;
    cin >> s.name;
    cout << "Scores: ";
    for(int j = 0; j < COURSES; ++j)
        cin >> s.scores[j];
    cout << endl;
}
```

正确

inputStudent函数设计

- 方案三：指针传递

```
void inputStudent(studentT *ps)
{
    cout << "No Name: ";
    cin >> ps->no;
    cin >> ps->name;
    cout << "Scores: ";
    for(int j = 0; j < COURSES; ++j)
        cin >> ps->scores[j];
    cout << endl;
}
```

- 方案二：引用传递

```
void inputStudent(studentT &s)
{
    cout << "No Name: ";
    cin >> s.no;
    cin >> s.name;
    cout << "Scores: ";
    for(int j = 0; j < COURSES; ++j)
        cin >> s.scores[j];
    cout << endl;
}
```

指针传递也是可行的

Print Student Info如何继续改进？

- 方案一：值传递

```
void printStudent(studentT s, int sum)
{
    cout << s.no << "\t"
         << s.name << "\t"
         << sum << endl;
}
```

Print Student Info如何继续改进？

- 方案一：值传递

```
void printStudent(studentT s, int sum)
{
    cout << s.no << "\t"
         << s.name << "\t"
         << sum << endl;
}
```

缺点：浪费空间，影响运行速度

Print Student Info如何继续改进？

- 方案一：值传递

```
void printStudent(studentT s, int sum)
{
    cout << s.no << "\t"
         << s.name << "\t"
         << sum << endl;
}
```

缺点：浪费空间，影响运行速度

- 改进方案

```
(const studentT &s, int sum)
```

是否需要const?

Print Student Info如何继续改进？

- 方案一：值传递

```
void printStudent(studentT s, int sum)
{
    cout << s.no << "\t"
         << s.name << "\t"
         << sum << endl;
}
```

缺点：浪费空间，影响运行速度

- 改进方案

```
(const studentT &s, int sum)
```

是否需要const?

const int *p;

不能通过p修改它指向的单元的内容，但p可以指向不同的变量

int *const p = &x;

可以通过p修改指向的对象，但p的值不能变

即p永远只能指向x。p定义时必须给出初值

const int *const p = &x;

p的值不能修改，*p的值也不能修改

C++的常规做法

特点：节约内存，提高函数调用速度，可靠

```
void inputStudent(studentT &s)
{
    cout << "No Name: ";
    cin >> s.no;
    cin >> s.name;
    cout << "Scores: ";
    for(int j = 0; j < COURSES; ++j)
        cin >> s.scores[j];
    cout << endl;
}
```

```
void printStudent(const studentT &s, int sum)
{
    cout << s.no << "\t"
         << s.name << "\t"
         << sum << endl;
}
```

总结：指向结构体的指针作为参数

- 由于结构体一般占用的内存量都比较大，值传递既浪费空间又浪费时间。因此可用指针传递或引用传递代替值传递
- 指针传递形式比较繁琐，所以C++通常用引用传递
- 引用传递的问题是函数中可以修改实际参数，要控制函数中不能修改实际参数，可以加**const**限定

返回结构体**类型**的函数

- 一个函数返回一个结构体，如：

```
studentT getStudentData( )
{
    studentT student;
    ……
    return(student);
}
```

- 返回的是一个结构体的复制(拷贝)

- 调用函数中可以有这样的程序段

```
int main()
{
    studentT s1;
    s1 = getStudentData();
     return 0;
}
```

返回结构体**引用**的函数

- 函数返回一个结构体的引用。如：

```
studentT  &getStudentData()
{
    studentT  *student  =  new  studentT;
    …….
    return(*student);
}
```

- 本质上返回的是结构体的地址
- 函数中返回的结构体不能是局部变量

- 调用函数中可以有这样的程序段

```
int main()
{
    studentT &s1 = getStudentData();
    ……

    //不需要的时候，需要回收变量空间
    delete &s1;
}
```

Quiz：选择题

对结构体变量stu1中成员age的非法引用是（ ）

```
struct studentT {
    int num;
    float age;
} ;
studentT stu1,  *p;
p=&stu1;
```

A. stu1.age　　B. studentT.age

C. p->age　　D. (*p).age

Quiz：选择题

对结构体变量stu1中成员age的非法引用是（B）

```
struct studentT {
    int num;
    float age;
};
studentT stu1,  *p;
p=&stu1;
```

A. stu1.age　　B. studentT.age

C. p->age　　D. (*p).age

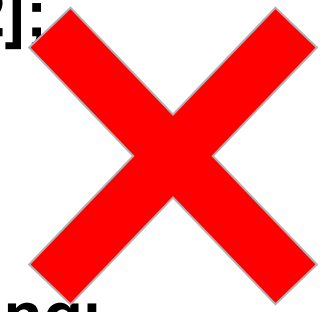
总结

- 结构体主要用于处理更复杂的数据
- 声明结构类型时，不产生内存的分配
- 定义结构变量时，才会分配内存空间
- 作为函数参数传递时，宜采用引用传递

扩展

- 我们提到结构体不可嵌套，但可以嵌套结构体指针

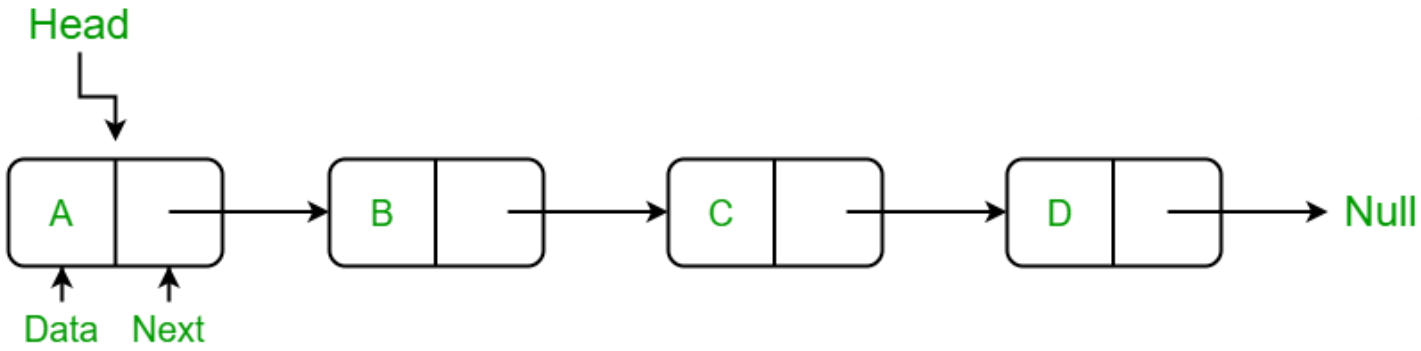
```
struct studentT {
    char  no[14];
    char  name[12];
    int math;
    int english;
    int programming;
    studentT  subStu;
};
```



```
struct studentLinkT {
    char  no[14];
    char  name[12];
    int math;
    int english;
    int programming;
    studentLinkT  *next;
};
```

扩展

- 我们提到结构体不可嵌套，但可以嵌套结构体指针
- 事实上，数据结构中的链表即是这样实现的

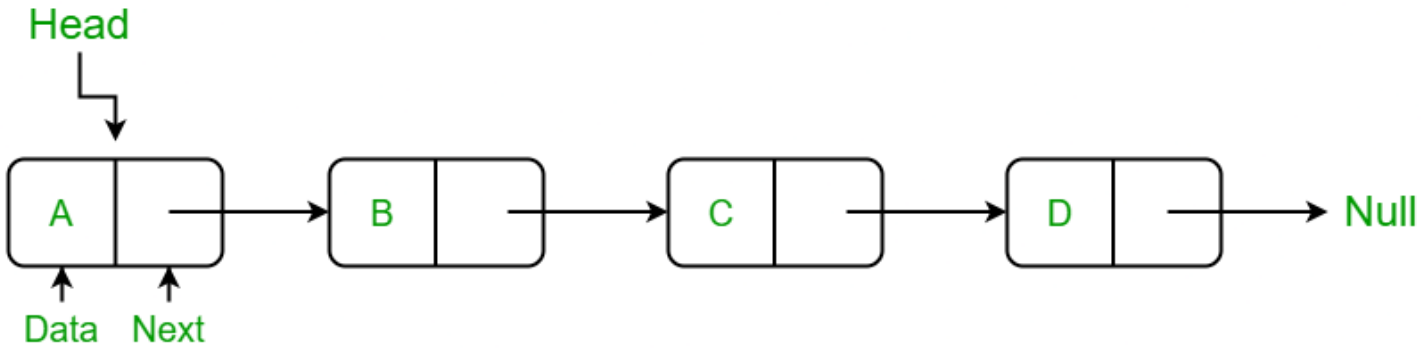



```
// C implementation
struct Node {
    int data;
    Node* next;
};
```

```
//C++ implementation
class Node {
public:
    int data;
    Node* next;
};
```

扩展

- 我们提到结构体不可嵌套，但可以嵌套结构体指针
- 事实上，数据结构中的链表即是这样实现的




```
// C implementation
struct Node {
    int data;
    struct Node* next;
};
```

```
//C++ implementation
class Node {
public:
    int data;
    Node* next;
};
```

第十章（类）见！