程序设计
Programming

Lecture 8: 结构

基本数据类型与构造数据类型

- 基本数据类型
  - ✓基本算数类型：整数，浮点数
  - ✓char，short，int，long，float，double等等

- 构造数据类型
  - ✓使用基本数据类型或构造数据类型进行组合构造的类型
  - ✓数组：一组相同数据类型元素的有序组合
  - ✓指针：存放其他变量地址的数据类型
  - ✓结构、联合：把不同数据类型结合成一个整体的数据类型，需要程序员自定义
  - ✓枚举：一组整型常量的集合

基本数据类型与构造数据类型

- 基本数据类型
  - ✓基本算数类型：整数，浮点数
  - ✓char，short，int，long，float，double等等

- 构造数据类型
  - ✓使用基本数据类型或构造数据类型进行组合构造的类型
  - ✓数组：一组相同数据类型元素的有序组合
  - ✓指针：存放其他变量地址的数据类型
  - ✓结构、联合：把不同数据类型结合成一个整体的数据类型，需要程序员自定义
  - ✓枚举：一组整型常量的集合

结构的定义

```
struct  结构名
{
        类型名  结构成员名1；
        类型名  结构成员名2；
        。。。。。。
        类型名  结构成员名n；
};
```

- struct关键字
- 结构名和结构成员名都是C语言标识符
- 类型名可以是基本数据类型，也可以是构造数据类型
- 以分号结束，因为struct的声明是一个语句

```c
struct score
{
    int math;
    int chinese;
    int english;
};
```

```c
struct student
{
    char *name;
    char gender[6];
    int age;
    struct score gaokao_score;
};
```

结构变量的定义

用法相当于int, double, char等

- 单独定义：先定义结构，再用结构定义结构变量

struct 结构名 结构变量名;

例如：struct score gaokao_score;

结构变量的定义

- 单独定义：先定义结构，再用结构定义结构变量

struct 结构名 结构变量名;
例如：struct score gaokao_score;

```
struct student
{
    char *name;
    char gender[6];
    int age;
    struct score gaokao_score;
};
```

结构变量的定义

- 单独定义：先定义结构，再用结构定义结构变量

  struct 结构名 结构变量名;
  例如：struct score gaokao_score;

```
struct student
{
    char *name;
    char gender[6];
    int age;
    struct score gaokao_score;
};
```

```
int main()
{
    struct student stu1;

    return 0;
}
```

结构变量的定义

- 单独定义：先定义结构，再用结构定义结构变量

单独定义

```
struct score
{
    int math;
    int chinese;
    int english;
};
```

struct 结构名 结构变量名;

例如：struct score gaokao_score;

结构变量的定义

- 单独定义：先定义结构，再用结构定义结构变量

  struct 结构名 结构变量名;
  例如：struct score gaokao_score;

单独定义

```
struct score
{
    int math;
    int chinese;
    int english;
};
```

- 混合定义：同时定义结构和结构变量

混合定义

```
struct score
{
    int math;
    int chinese;
    int english;
}gaokao_score;
```

结构变量的定义

- 单独定义：先定义结构，再用结构定义结构变量

  struct 结构名 结构变量名;
  例如：struct score gaokao_score;

- 混合定义：同时定义结构和结构变量
- 无类型名定义：省略结构名，直接定义结构变量
  - ✓无法再次用此结构单独定义其他新变量

单独定义

```
struct score
{
    int math;
    int chinese;
    int english;
};
```

无类型名定义

```
struct
{
    int math;
    int chinese;
    int english;
}gaokao_score;
```

结构变量的定义

- 单独定义：先定义结构，再用结构定义结构变量

struct 结构名 结构变量名;
例如：struct score gaokao_score;

单独定义

```
struct score
{
    int math;
    int chinese;
    int english;
};
```

```
struct
{
    int english;
}gaokao_score;
```

一般推荐使用单独定义

义结构变量

定义

义结构变量

✓无法再次用此结构单独定义其他新变量

结构变量的初始化和赋值

- 声明结构变量时初始化
  - ✓利用初始化表将各个成员顺序赋值，并保持数据类型与成员类型一致
- 先声明变量，后对其成员赋值
  - ✓结构变量.成员变量=数值

结构变量的初始化和赋值

- 声明结构变量时初始化
  - ✓利用初始化表将各个成员顺序赋值，并保持数据类型与成员类型一致
- 先声明变量，后对其成员赋值
  - ✓结构变量.成员变量=数值

```
struct score
{
    int math;
    int chinese;
    int english;
};
```

```
struct student
{
    char *name;
    char gender[6];
    int age;
    struct score gaokao_score;
};
```

结构变量的初始化和赋值

- 声明结构变量时初始化
  - ✓利用初始化表将各个成员顺序赋值，并保持数据类型与成员类型一致
- 先声明变量，后对其成员赋值
  - ✓结构变量.成员变量=数值

```c
struct score
{
    int math;
    int chinese;
    int english;
};
```

```c
int main()
{
    struct score score1 = {140, 120, 130};  //利用初始化表顺序赋值

    struct student stu1;  //先声明结构变量，再对其成员赋值
    stu1.name = "Bob";
    strcpy(stu1.gender, "male");
    stu1.age = 20;
    stu1.gaokao_score = score1;

    return 0;
}
```

```c
struct student
{
    char *name;
    char gender[6];
    int age;
    struct score gaokao_score;
};
```

结构变量的使用

- 使用结构变量操作符 . 来访问结构中的成员变量

结构变量的使用

- 使用结构变量操作符 . 来访问结构中的成员变量
- 结构变量可以作为函数的参数或者返回值

结构变量的使用

- 使用结构变量操作符.来访问结构中的成员变量
- 结构变量可以作为参数或者返回值

```
struct student newStu(char *name,
                      char *gender,
                      int age,
                      struct score score1)
{
    struct student stu1;
    stu1.name = name;
    strcpy(stu1.gender, gender);
    stu1.age = age;
    stu1.gaokao_score = score1;

    return stu1;
}
```

结构变量的使用

- 使用结构变量操作符“.”来访问结构中的成员变量

```c
struct student newStu(char *name,
                      char *gender,
                      int age,
                      struct score score1)
{
    struct student stu1;
    stu1.name = name;
    strcpy(stu1.gender, gender);
    stu1.age = age;
    stu1.gaokao_score = score1;

    return stu1;
}
```

```c
int main()
{
    struct score score1 = {140, 120, 130};

    struct student stu1 = newStu("Bob", "male", 18, score1);
    printf("%s %s %d %d %d %d\n", stu1.name,
                                  stu1.gender,
                                  stu1.age,
                                  stu1.gaokao_score.math,
                                  stu1.gaokao_score.chinese,
                                  stu1.gaokao_score.english);

    return 0;
}
```

结构数组

- 同一结构类型的变量组成的数组

```
struct  student  my_students[40];
```

- mystudents[i]就是一个student结构的结构变量，用法和单独的student结构变量一致

结构数组

```
int main()
{
    struct score score1 = {140, 120, 130};
    struct student my_students[40];
    for (int i = 0; i < 40; i++)
        my_students[i] = newStu("Bob", "male", 18, score1);

    return 0;
}
```

结构指针

- 指向结构类型变量的指针

  struct 结构名 * 结构指针变量名
  struct student stu1;
  struct student *sp;
  sp = &stu1; //stu1的开始地址，即sp指向结构变量stu1

- 用结构指针间接访问结构变量
  - ✓用间接访问符*: (*sp).name = “Bob”;
  - ✓用指针运算符->: sp->name = “Bob”;

结构指针

```c
int main()
{
    struct score score1 = {140, 120, 130};
    struct student stu1 = newStu("Bob", "male", 18, score1);
    struct student *sp = &stu1;
    printf("%s %s %d\n", stu1.name, sp->gender, (*sp).age);

    return 0;
}
```

结构指针作为函数参数

- 传参效率
  - ✓使用结构体传参，需要把结构体实参每个变量传递给形参相应变量，耗时且耗空间
  - ✓使用结构指针作为参数传递给函数可以提高传参效率

结构指针作为函数参数

- 传参效率
  - ✓使用结构体传参，需要把结构体实参每个变量传递给形参相应变量，耗时且耗空间
  - ✓使用结构指针作为参数传递给函数可以提高传参效率

```
struct score
{
    int math;
    int chinese;
    int english;
};

struct student
{
    char *name;
    char gender[6];
    int age;
    struct score *gaokao_score;
};
```

结构指针作为函数参数

```
struct student newStu(char *name,
                      char *gender,
                      int age,
                      struct score *score1)
{
    struct student stu1;
    stu1.name = name;
    strcpy(stu1.gender, gender);
    stu1.age = age;
    stu1.gaokao_score = score1;

    return stu1;
}
```

```
struct score
{
    int math;
    int chinese;
    int english;
};

struct student
{
    char *name;
    char gender[6];
    int age;
    struct score *gaokao_score;
};
```

结构指针作为函数参数

```
struct student newStu(char *name,
{
}
```

```
struct score
};
```

```
int main()
{
    struct score score1 = {140, 120, 130};
    struct student stu1 = newStu("Bob", "male", 18, &score1);
    struct student *sp = &stu1;
    printf("%s %s %d\n", stu1.name, sp->gender, (*sp).age);

    return 0;
}
```

结构指针作为函数参数

- 传参效率
  - ✓使用结构体传参，需要把结构体实参每个变量传递给形参相应变量，耗时且耗空间
  - ✓使用结构指针作为参数传递给函数可以提高传参效率
- 传递结构指针可以改变结构的成员变量的值

参数

- 实

数

- 成

```c
#include <stdio.h>

struct student
{
    int age;
};

void chage(struct student st)
{
    st.age = 10;
}

int main()
{
    struct student st;
    st.age = 5;
    chage(st);
    printf("%d\n", st.age);
    return 0;
}
```

参数

- 实

- 数

- 成

```c
#include <stdio.h>

struct student
{
    int age;
};

void chage(struct student st)
{
    st.age = 10;
}

int main()
{
    struct student st;
    st.age = 5;
    chage(st);
    printf("%d\n", st.age);
    return 0;
}
```

```c
#include <stdio.h>

struct student
{
    int age;
};

void chage(struct student *st)
{
    st->age = 10;
}

int main()
{
    struct student st;
    st.age = 5;
    chage(&st);
    printf("%d\n", st.age);
    return 0;
}
```

什么时候需要用到结构？

- 将一些有共同特征或者属于共同对象的变量封装打包成一个整体，以便后续操作

- 面向对象编程的雏形
  - ✓类似于C++/Java/Python中的class
  - ✓但是缺少面向对象的另外两个基本特征：继承和多态

typedef用于结构

- 类型重命名

```
typedef struct student
{
    成员;
}student;
```

随后就可以使用student作为类型名，声明结构变量！

```c
typedef struct score
{
    int math;
    int chinese;
    int english;
}score;

typedef struct student
{
    char *name;
    char gender[6];
    int age;
    struct score *gaokao_score;
}student;
```

dent

声明结构变量！

```c
typedef struct score
{
    int math;
    int chinese;
    int english;
}score;

typedef struct stu
{
    char *name;
    char gender[6]
    int age;
    struct score *
}student;
```

```c
int main()
{
    score score1 = {140, 120, 130};
    student stu1 = newStu("Bob", "male", 18, &score1);
    struct student *sp = &stu1;
    printf("%s %s %d\n", stu1.name, sp->gender, (*sp).age);

    return 0;
}
```

联合

- 联合union
  - ✓用法和struct类似

```
union 联合名
{
        成员1；
        成员2；
        ……
        成员n;
};
```

  - ✓区别是union的所有成员共享内存开始地址

联合

- 联合union
  - ✓用法和struct类似

```
union 联合名
{
        成员1；
        成员2；
        ……
        成员n;
};
```

  - ✓区别是union的所有成员共享内存开始地址

```c
typedef struct score
{
    int math;
    int chinese;
    int english;
}score;

union student
{
    char name[10];
    char gender[6];
    int age;
};
```

联合

```c
int main()
{
    union student stu1;
    strcpy(stu1.name, "Bob");
    printf("%s %s %d\n", stu1.name, stu1.gender, stu1.age);
    strcpy(stu1.gender, "male");
    printf("%s %s %d\n", stu1.name, stu1.gender, stu1.age);
    stu1.age = 18;
    printf("%s %s %d\n", stu1.name, stu1.gender, stu1.age);

    return 0;
}
```

```c
typedef struct score
{
    int math;
    int chinese;
    int english;
}score;

union student
{
    char name[10];
    char gender[6];
    int age;
};
```

枚举

- 枚举enum
  - ✓用法和struct类似

```
enum 枚举名
{
        成员1；
        成员2；
        ……
        成员n;
};
```

  - ✓一组int型常量的集合（可以视作多个#define）
  - ✓成员1默认值为0，之后的成员依次累加1
  - ✓如果某个成员被赋予特定值，则之后的成员从此特定值开始累加1

```c
enum day {Mon, Tue, Wed, Thu, Fri, Sat, Sun};

int main()
{
    printf("%d, %d, %d, %d, %d, %d, %d\n", Mon, Tue, Wed, Thu, Fri, Sat, Sun);
    return 0;
}
```

```
{
        成员1；
        成员2；
        ……
        成员n;
};
```

```
0, 1, 2, 3, 4, 5, 6


...Program finished with exit code 0
Press ENTER to exit console.
```

✓一组int型常量的集合（

✓成员1默认值为0，之后

✓如果某个成员被赋予特定值，则之后的成员从此特定值开始累加1

```
enum day {Mon, Tue=2, Wed, Thu, Fri, Sat, Sun};

int main()
{
    printf("%d, %d, %d, %d, %d, %d, %d\n", Mon, Tue, Wed, Thu, Fri, Sat, Sun);
    return 0;
}
```

```
{
        成员1；
        成员2；
        ……
        成员n;
};
```

```
0, 2, 3, 4, 5, 6, 7


...Program finished with exit code 0
Press ENTER to exit console.
```

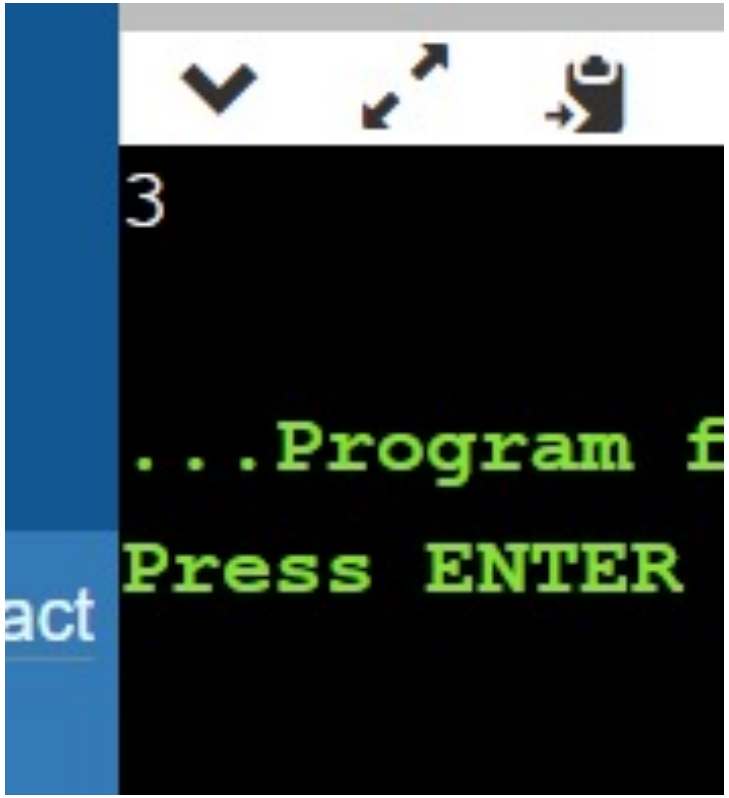
✓一组int型常量的集合（
✓成员1默认值为0，之后的成员依次累加1
✓如果某个成员被赋予特定值，则之后的成员从此特定值开始累加1

枚举的使用

```c
#include <stdio.h>

enum DAY
{
      MON=1, TUE, WED, THU, FRI, SAT, SUN
};

int main()
{
    enum DAY day;
    day = WED;
    printf("%d",day);
    return 0;
}
```



什么时候需要用到联合和枚举？

- 联合
  - ✓当成员变量基本不需要同时使用的时候，使用联合可以节约内存开销
- 枚举
  - ✓当某变量具有固定取值范围的时候，使用枚举可以将其声明为特殊的常量，增强代码可读性

小结

- 结构的定义
  - ✓结构变量的定义、赋值和使用
- 结构数组
- 结构指针
- 联合、枚举（了解）
  - ✓与结构的区别