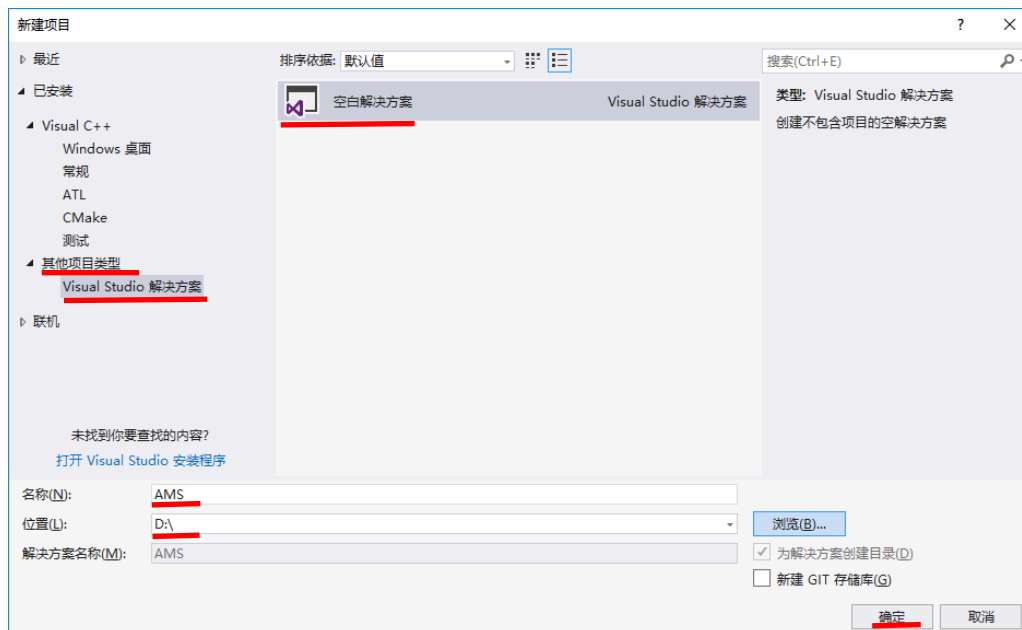


## 一. 创建解决方案

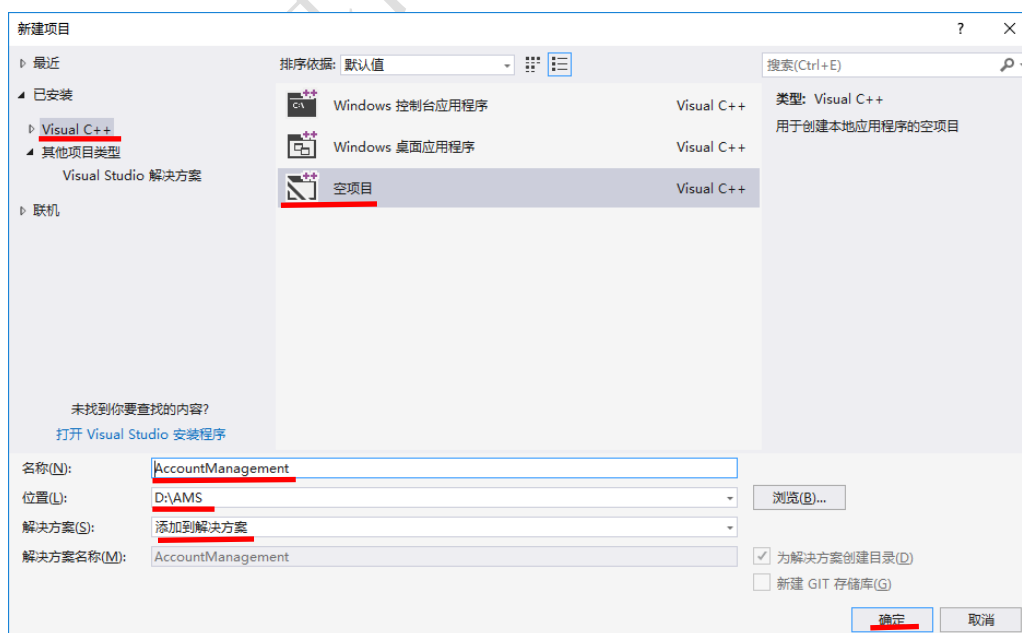
1. 打开 VS2017;
2. 从菜单中选“文件→新建→项目”，出现对话框；
3. 对话框中选“其他项目类型→Visual Studio 解决方案→空白解决方案”，输入解决方案名称为“AMS”,设置路径（根据自己实际情况设置，这里是 D 盘）；



4. “确定”后，会打开 VS 下刚刚创建的 AMS 解决方案，同时在 D 盘会自动生成 AMS 解决方案文件夹，文件夹中自动生成 AMS.sln 文件。

## 二. 创建工程

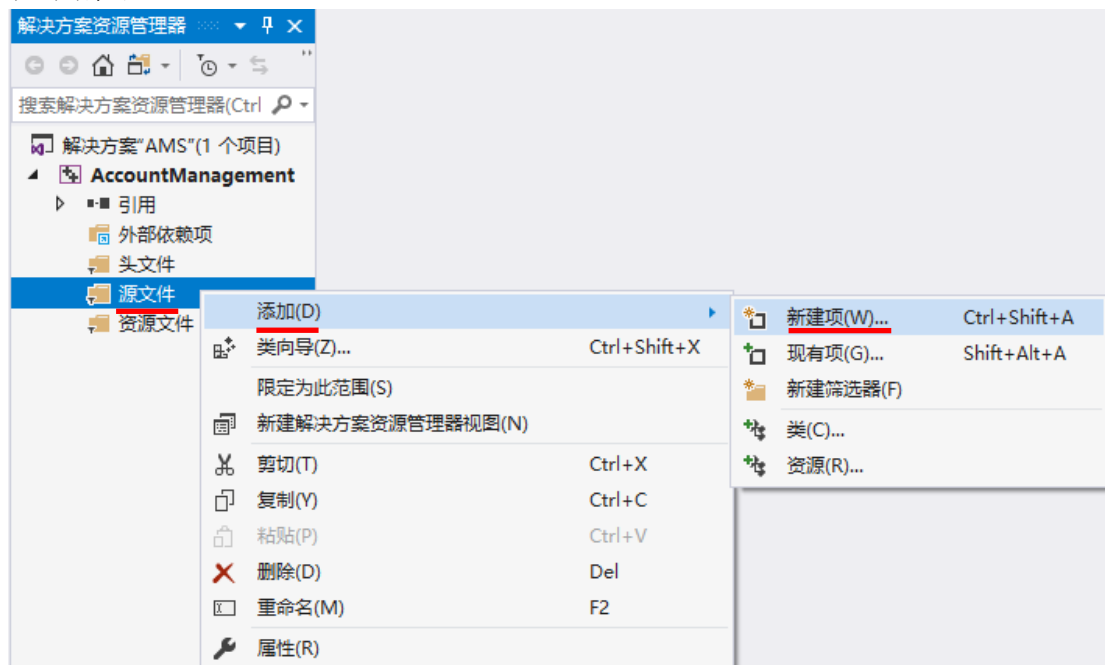
1. 在上面创建的解决方案中，从菜单中选“文件→新建→项目”，出现对话框；
2. 对话框中选“Visual C++→空项目”，输入工程名称为“AccountManagement”,设置路径（前面生成的解决方案路径 D:\AMS），选择“添加到解决方案”；



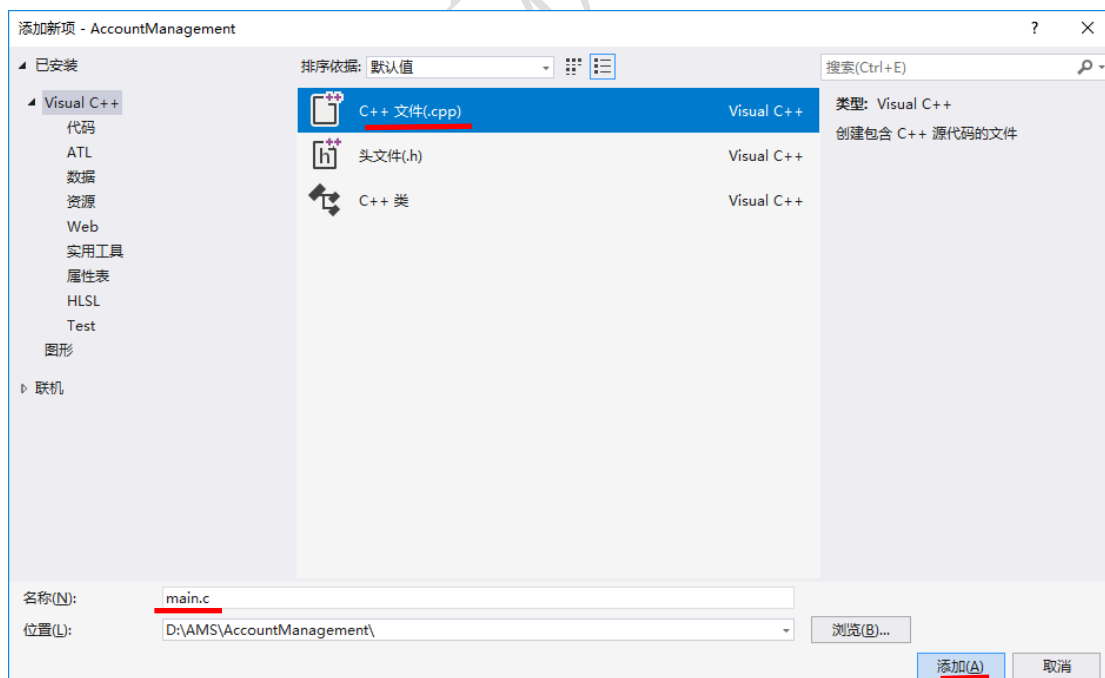
3. “确定”后，会打开刚刚创建的 AccountManagement 工程，同时 AMS 文件夹下会生成 AccountManagement 工程文件夹，以及相关文件。

### 三. 添加主文件

1. 在“解决方案资源管理器”下的“源文件”目录上，单击右键，选择“添加→新建项”，出现对话框：



2. 对话框中选“C++ 文件 (.cpp)”，输入名称为 main.c；



3. “添加”后，自动打开 main.c 的编辑窗口，同时添加到“解决方案资源管理器”下的“源文件”目录下。

（以后需要添加新的\*\*\*.c 源文件时，都按照上述过程来进行）

## 四. 添加编辑主文件代码

在主文件编辑窗口输入以下代码:

```
main.c  X
AccountManagement (全局范围)

1  #define _CRT_SECURE_NO_WARNINGS
2  #include<stdio.h>
3  #include<stdlib.h>
4
5  // [函数名]    main
6  // [功能]      程序入口函数
7  // [参数]      void
8  // [返回值]    0: 程序正常退出; 非零: 程序异常
9  int main()
10 {
11     // 输入的菜单项编号
12     int nSelection;
13     printf("\n★★★欢迎进入计费管理系统★★★\n\n");
14     do{
15         printf("-----计费系统菜单-----\n");
16         printf("1. 添加卡\n");
17         printf("2. 查询卡\n");
18         printf("3. 上机\n");
19         printf("4. 下机\n");
20         printf("5. 充值\n");
21         printf("6. 退费\n");
22         printf("7. 查询统计\n");
23         printf("8. 注销卡\n");
24         printf("0. 退出\n");
25         printf("请选择菜单项编号 (0~8): \n");
26
27         nSelection = -1;
28         // 输入菜单项编号
29         scanf("%d",&nSelection);
30         getchar();
31         // 输出选择的子菜单
32         switch(nSelection){
33             case 1:
34                 printf("-----添加卡-----\n");
35                 break;
36             case 2:
37                 printf("-----查询卡-----\n");
38                 break;
39             case 3:
40                 printf("-----上机-----\n");
41                 break;
42             case 4:
43                 printf("-----下机-----\n");
44                 break;
45             case 5:
46                 printf("-----充值-----\n");
47                 break;
48             case 6:
49                 printf("-----退费-----\n");
50                 break;
51             case 7:
52                 printf("-----查询统计-----\n");
53                 break;
```

```

54      case 8:
55          printf("-----注销卡-----\n");
56          break;
57      case 0:
58          printf("谢谢你使用本系统! \n");
59          break;
60      default:
61          printf("输入菜单选项错误! \n请重新输入! \n");
62      }
63      printf("\n");
64  }while(nSelection !=0);
65  system("pause");
66
67  return 0;
68  }

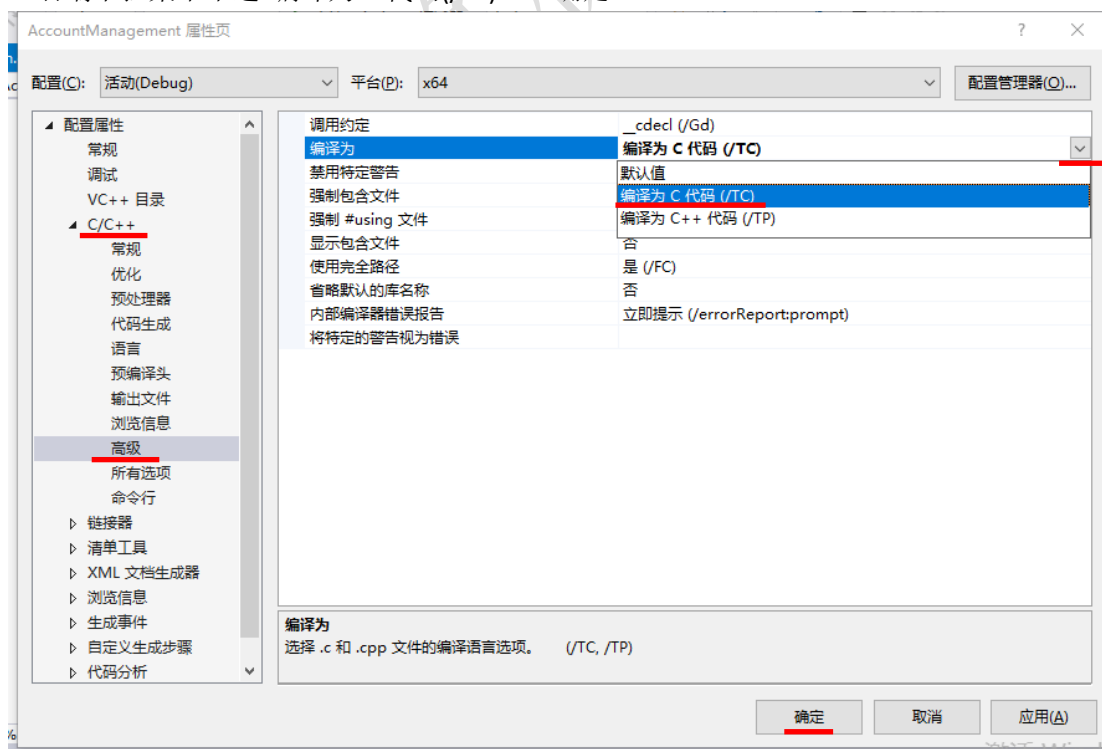
```

- 思考: 1. `#include<stdio.h>` 为什么要包含这个头文件?
2. `#include<stdlib.h>` 和 `system("pause");` 的作用?
3. 输入错误数据会出现什么情况 (中文, 字母, 字母开头的数字, 数字开头后带字母, 字母分隔的数字...)? `getchar` 函数起什么作用 (去掉该语句后测试看看)?
4. `#define _CRT_SECURE_NO_WARNINGS` (后面再说明)

## 五. 编译并连接程序

### 1. 设置编译方式

在 VS 中开发 C 程序, 一般设置为按照 C 语言进行编译, 从菜单中选“项目 → AccountManagement 属性(p)...”, 打开对话框, 对话框中选“C/C++” → “高级” → “编译为” 右端下拉菜单中选“编译为 C 代码(/TC)” → “确定”



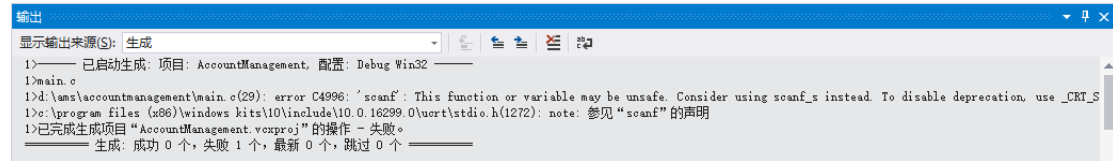
2. 选择“生成 → 生成解决方案”, 在 AMS 文件夹中出现 Debug 文件夹, 其中生成

AccountManagement.exe 文件

3. 以后程序代码修改后都需要重新编译链接

**注意!!!**

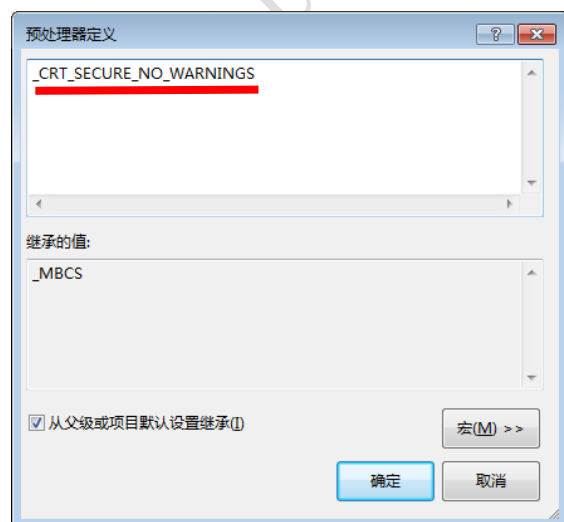
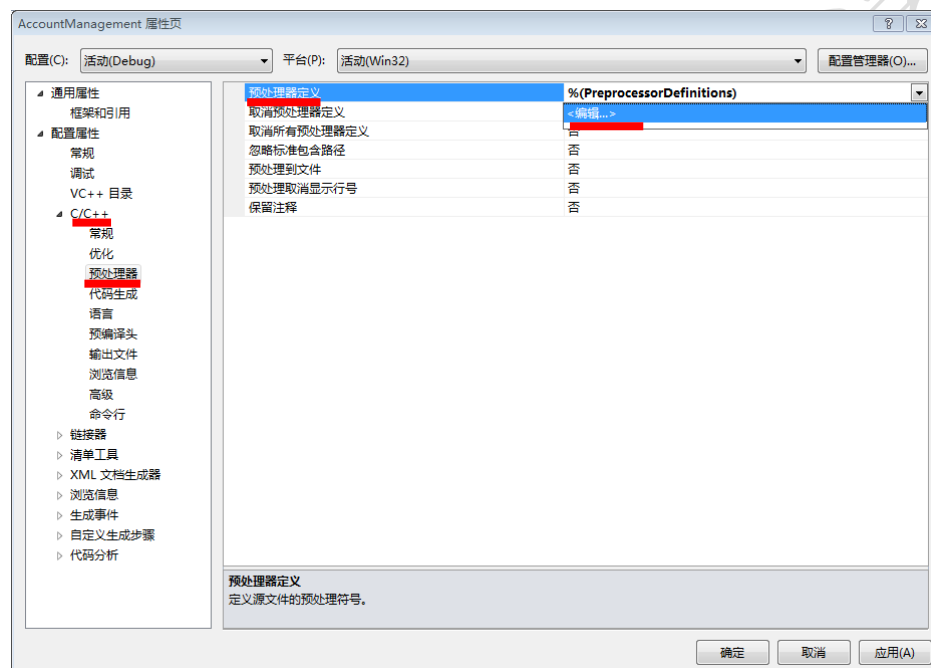
VS 中调用 `scanf` 等 C 函数时输出窗口会提示 `_CRT_SECURE_NO_WARNINGS` 错误，原因是这些函数不安全，可能会造成内存泄露等。



```
输出
显示输出来源(S): 生成
1>----- 已启动生成: 项目: AccountManagement, 配置: Debug Win32 -----
1>main.c
1>d:\msvc\accountmanagement\main.c(29): error C4996: 'scanf': This function or variable may be unsafe. Consider using scanf_s instead. To disable deprecation, use _CRT_S
1>c:\program files (x86)\windows kits\10\include\10.0.16299.0\ucrt\stdio.h(1272): note: 参见“scanf”的声明
1>已完成生成项目“AccountManagement.vcxproj”的操作 - 失败。
----- 生成: 成功 0 个, 失败 1 个, 最新 0 个, 跳过 0 个 -----
```

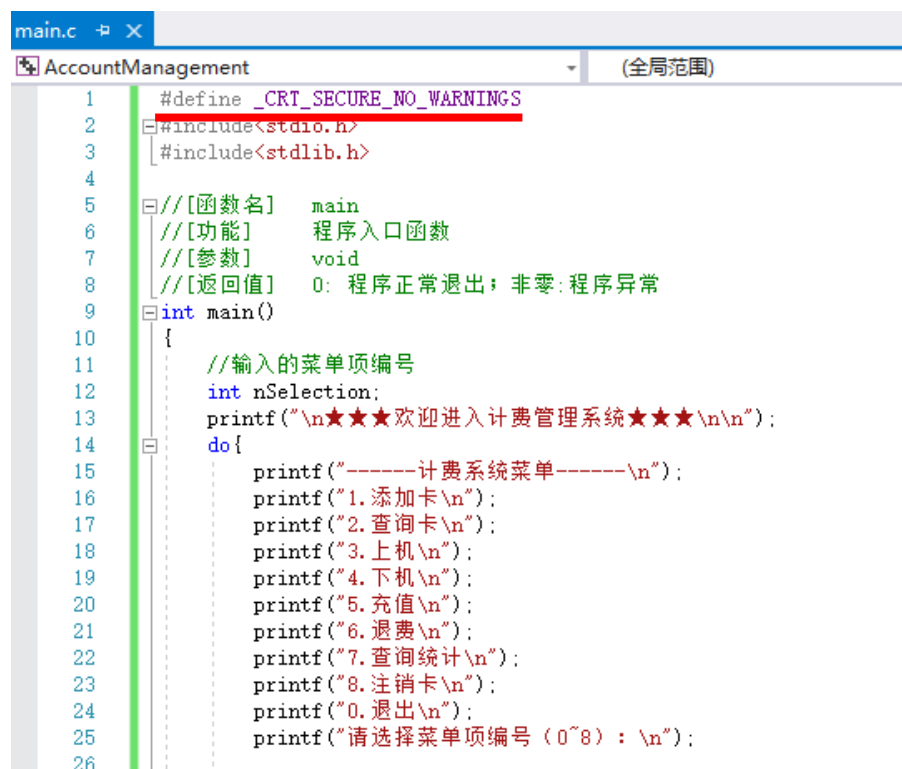
关闭警告的方法一:

- 1) VS 中选“项目->属性”打开对话框
- 2) 选“配置属性->C/C++ ->预处理器->预处理器定义->编辑” 打开对话框中添加 `_CRT_SECURE_NO_WARNINGS` 这个预定义，确定后即可



关闭警告的方法二:

手工在文件最上面，就是第一行，添加宏定义 `#define _CRT_SECURE_NO_WARNINGS`



```

1  #define _CRT_SECURE_NO_WARNINGS
2  #include<stdio.h>
3  #include<stdlib.h>
4
5  //[[函数名]   main
6  //[[功能]     程序入口函数
7  //[[参数]     void
8  //[[返回值]   0: 程序正常退出；非零:程序异常
9  int main()
10 {
11     //输入的菜单项编号
12     int nSelection;
13     printf("\n★★★欢迎进入计费管理系统★★★\n\n");
14     do{
15         printf("-----计费系统菜单-----\n");
16         printf("1. 添加卡\n");
17         printf("2. 查询卡\n");
18         printf("3. 上机\n");
19         printf("4. 下机\n");
20         printf("5. 充值\n");
21         printf("6. 退费\n");
22         printf("7. 查询统计\n");
23         printf("8. 注销卡\n");
24         printf("0. 退出\n");
25         printf("请选择菜单项编号 (0~8) : \n");
26

```

## 六. 运行程序 和测试

1. 选择“调试→开始执行”，或快捷键 `Ctrl+F5`
2. 选择“调试→启动调试”，会同时打开几个调试窗口
3. 系统的测试应该包含两个方面，即：
  - 功能正确测试：正常输入，检测是否能得到预期的正确结果。
  - 错误测试：错误输入，包括数据内容、数据格式异常等，检测系统是否能给出正确的提示，没有非正常退出。

## 七. 封装系统菜单输出函数

1. 在主函数 `main` 后面添加一个新的 `outputMenu()` 函数，将步骤四中虚线中代码封装到 `outputMenu()` 函数中；
  2. 将主函数 `main` 中对应位置代码删除后，直接调用 `outputMenu()` 函数；
  3. 主函数前面添加 `outputMenu()` 函数声明；
  4. 其他部分不变。
- 调整后代码如下：

```

main.c  X
AccountManagement (全局范围)
1  #define _CRT_SECURE_NO_WARNINGS
2  #include<stdio.h>
3  #include<stdlib.h>
4
5  //函数声明
6  void outputMenu();
7
8  //[[函数名]   main
9  //[[功能]     程序入口函数
10 //[[参数]     void
11 //[[返回值]   0: 程序正常退出; 非零: 程序异常
12 int main()
13 {
14     //输入的菜单项编号
15     int nSelection;
16     printf("\n★★★欢迎进入计费管理系统★★★\n\n");
17     do{
18         //输出系统菜单
19         outputMenu();
20
21         nSelection = -1;
22         //输入菜单项编号
23         scanf("%d",&nSelection);
24         getchar();
25         //输出选择的子菜单
26         switch(nSelection){
27
28             }while(nSelection !=0);
29             system("pause");
30
31             return 0;
32         }
33
34         //[[函数名]   outputMenu
35         //[[功能]     输出系统菜单
36         //[[参数]     void
37         //[[返回值]   void
38 void outputMenu()
39 {
40     //输出系统菜单
41     printf("-----计费系统菜单-----\n");
42     printf("1. 添加卡\n");
43     printf("2. 查询卡\n");
44     printf("3. 上机\n");
45     printf("4. 下机\n");
46     printf("5. 充值\n");
47     printf("6. 退费\n");
48     printf("7. 查询统计\n");
49     printf("8. 注销卡\n");
50     printf("0. 退出\n");
51     printf("请选择菜单项编号 (0~8): \n");
52 }

```

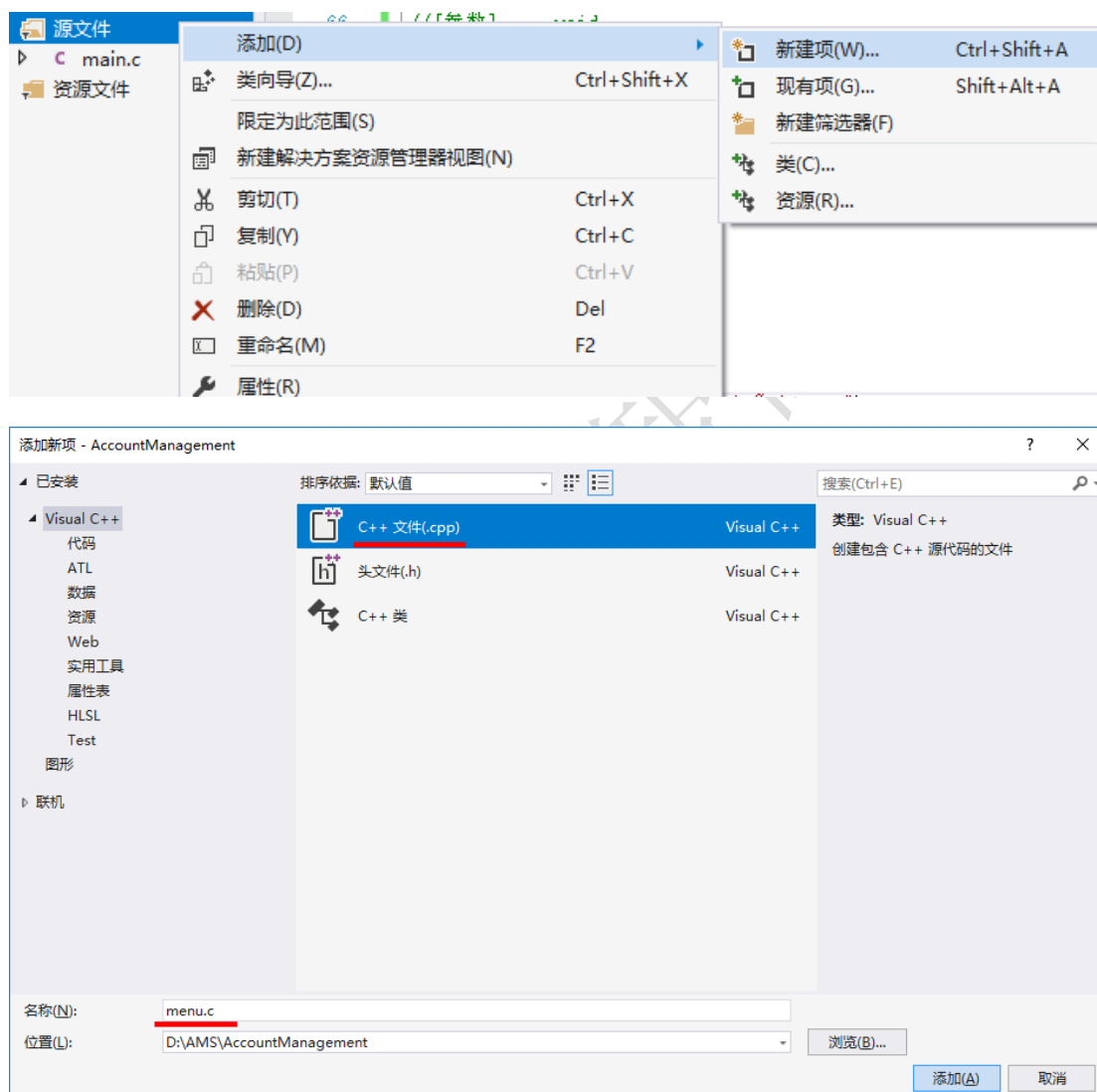
思考：

- 1) 封装部分程序代码的好处？
- 2) 主函数前面为什么要有函数声明？什么时候需要函数声明？什么时候不需要？

5. 重新编译连接，重新调试运行

八. 优化程序结构

1. 按照步骤三的方法，添加 menu.c 文件，将与用户界面输入\输出相关的函数放在这里（目前只有 outputMenu 函数），即将前面 main.c 文件中 outputMenu 函数移过来

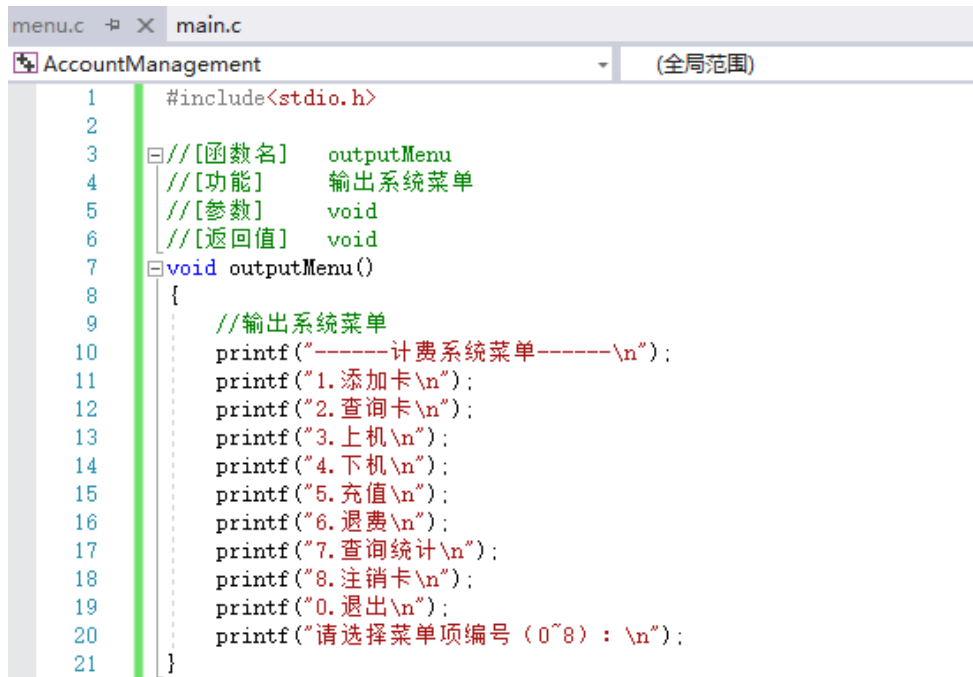


注意：

本教程是用 C 语言编写的，所有源文件是以.c 作为文件后缀名，如果在项目中源文件既有.c 后缀的 C 源文件，又有.cpp 后缀的 C++源文件，编译后链接可能会出现链接错误，函数找不到等错误，所以建议**项目中所有源文件后缀使用同一后缀**（使用 C 语言编写的就用.c 后缀，使用 C++语言编写的就用.cpp 后缀）

menu.c 中代码如下：



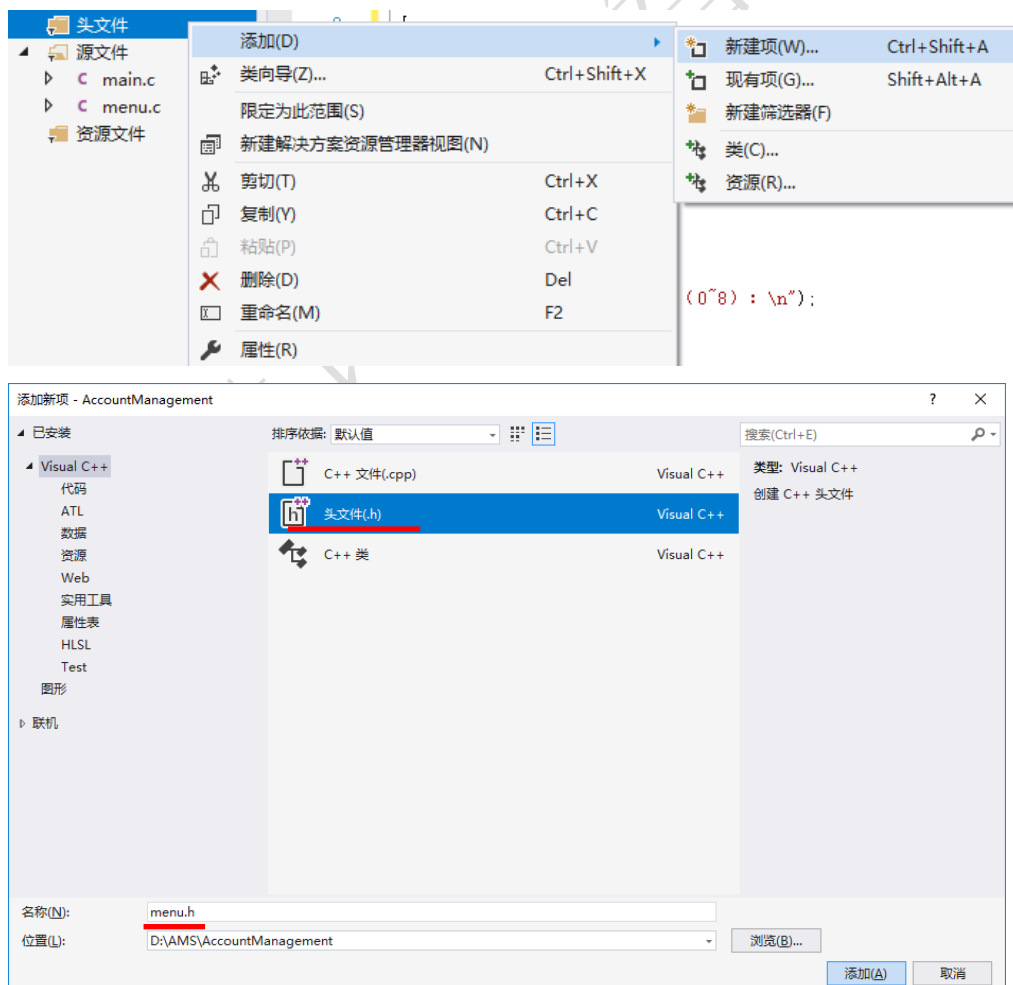


```

1  #include<stdio.h>
2
3  //[[函数名]   outputMenu
4  //[[功能]     输出系统菜单
5  //[[参数]     void
6  //[[返回值]   void
7  void outputMenu()
8  {
9      //输出系统菜单
10     printf("-----计费系统菜单-----\n");
11     printf("1. 添加卡\n");
12     printf("2. 查询卡\n");
13     printf("3. 上机\n");
14     printf("4. 下机\n");
15     printf("5. 充值\n");
16     printf("6. 退费\n");
17     printf("7. 查询统计\n");
18     printf("8. 注销卡\n");
19     printf("0. 退出\n");
20     printf("请选择菜单项编号 (0~8) : \n");
21 }

```

2. 添加 menu.h 文件，将 menu.c 中对应函数的声明放着这里（目前只有 outputMenu 函数）；按照上述类似方法，在“解决方案资源管理器”下的“头文件”目录上，单击右键，选择“添加→新建项”，出现对话框；对话框中选“头文件（.h）”，输入名称为 menu.h；



**注意:**

- 1) 头文件的名称 **menu.h** 和源文件的名称 **menu.c** 最好保持一致(名称一样, 大小写一样), 方便后面使用时易于找到和包含对应的函数申明
- 2) 项目中添加头文件和源文件都要按照上述方法添加, 并且不能随意拖动到其他位置 (拖动后如果配置文件没有相应修改, 可能无法找到需要的函数等)

menu.h 中代码如下:

```

1 //函数声明
2 void outputMenu();
3

```

3.主文件 main.c 中移走了前面的 outputMenu 函数声明, 移走了后面的 outputMenu 函数, 添加包含 menu.h 头文件。代码如下:

```

1 #define _CRT_SECURE_NO_WARNINGS
2 #include<stdio.h>
3 #include<stdlib.h>
4
5 #include"menu.h"
6
7 //函数声明
8 //void outputMenu(); //移到menu.h头文件中, 前面包含menu.h头文件, 此处删除
9
10 //【函数名】 main
11 //【功能】 程序入口函数
12 //【参数】 void
13 //【返回值】 0: 程序正常退出; 非零: 程序异常
14 int main()
15 {
16     //输入的菜单项编号
17     int nSelection;
18     printf("\n★★★欢迎进入计费管理系统★★★\n\n");
19     do{
20         //输出系统菜单
21         outputMenu();
22     } while(1);
23 }

```

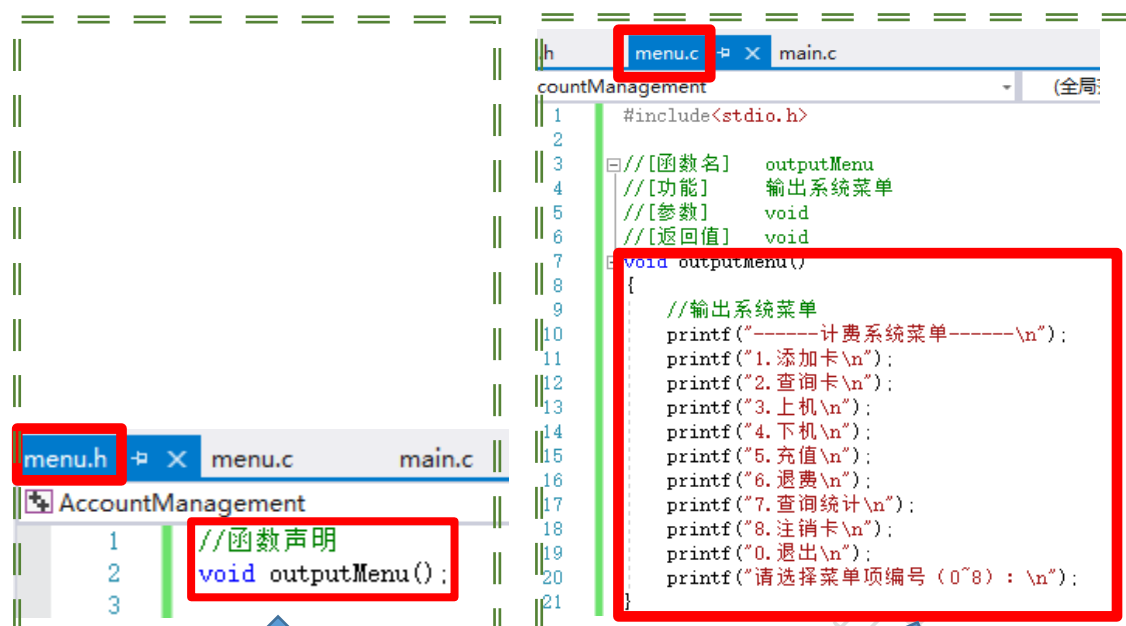
4 重新编译连接, 重新调试运行

思考: 1) .c 源文件和.h 头文件的作用? 什么时候要包含.h 头文件?

2) .c 源文件和.h 头文件是不是一对一的, 有一个源文件就有对应的一个头文件?

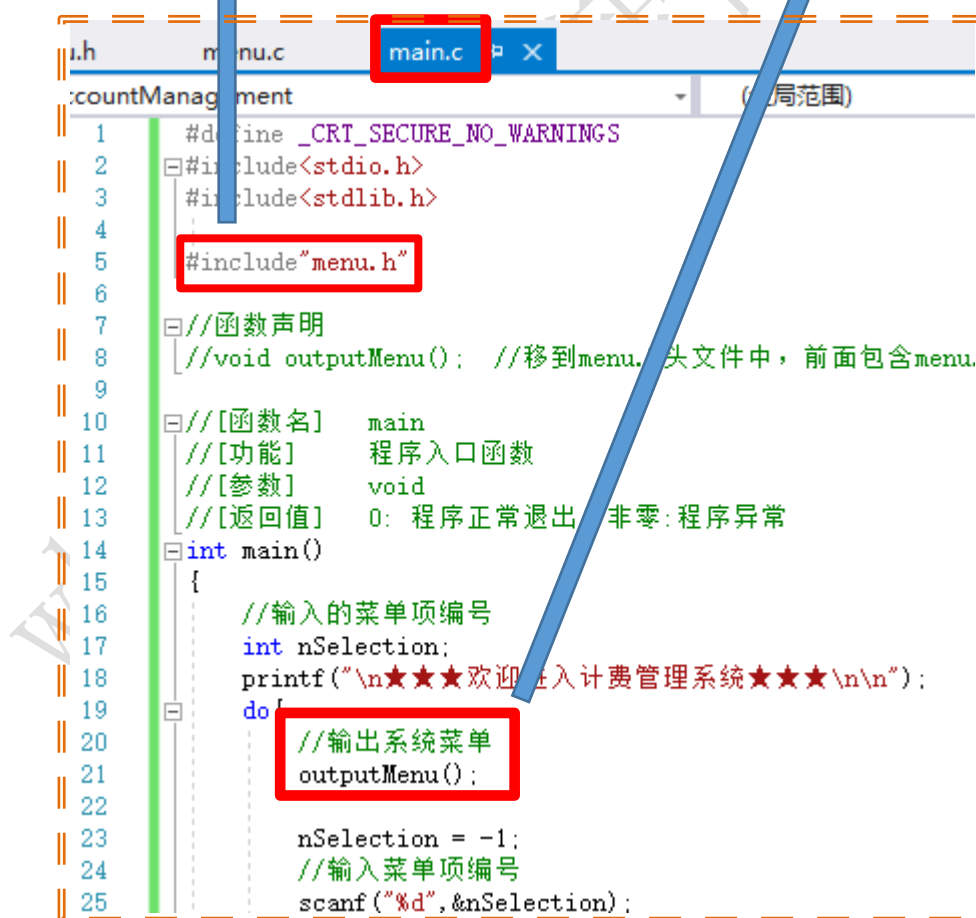
优化后整个项目包括三个文件 (一个 h 头文件, 两个 C 源文件)





编译时 include  
位置上替换为头  
文件中所有内容

运行时调用执行



## 九. 总结

本次任务建立的层次结构和调用关系

