Case Study: High Adventure Travel Agency—Part 2

This case study is a modification of the High Adventure Travel Agency program from Case Study 2. Recall that the program calculates and itemizes the charges for each of four vacation packages. Our next task is to enhance the program so it keeps a file of the vacation packages sold. In this case study, data structures will be designed to hold the data about each package. In Case Study 6, the modification will be completed with the addition of file I/O capabilities.

Creating the Data Structures

The original program in Case Study 2 uses functions to process each vacation package. The functions keep their data in local variables. For instance, here are the variable definitions of the `climbing` function, which calculates the costs of vacation package 1:

```
int    beginners,     // Those needing instruction
       advanced,      // Those not needing instruction
       needEquip;     // Those renting camping equipment
double baseCharges,   // Base charges
       charges,       // Total charges
       instruction,   // Cost of instruction
       equipment,     // Cost of equipment rental
       discount = 0,  // Discount
       deposit;       // Required deposit
```

Here are the variable definitions of the `scuba` function, which calculates, the costs of vacation package 2:

```
int    beginners,     // Those needing instruction
       advanced;      // Those not needing instruction
double baseCharges,   // Base charges
       charges,       // Total charges
       instruction,   // Cost of instruction
       discount = 0,  // Discount
       deposit;       // Required deposit
```

Here are the variable definitions of the `skyDive` function, which calculates the costs of vacation package 3:

```
int    party,        // Number in party
       lodge1,       // Number at 1st lodging choice
       lodge2;       // Number at 2nd lodging choice
double baseCharges,  // Base charges
       charges,      // Total charges
       discount = 0, // Discount
       lodging,      // Cost of lodging
       deposit;      // Required deposit
```

And finally, here are the variable definitions of the `spelunk` function, which calculates the costs of vacation package 4:

```
int    party,        // Number in party
       needEquip;    // Those renting camping equipment
double baseCharges,  // Base charges
       charges,      // Total charges
       equipment,    // Cost of equipment rental
       discount = 0, // Discount
       deposit;      // Required deposit
```

The new version of the program will not hold the vacation package data in local variables, but in a structure that will be passed to the functions. This structure will encapsulate all of the variables needed for any of the vacation packages into a single object. This will make the file I/O features, which will be implemented in Case Study 6, easier to design.

The first step in creating the new data structure is to replace the local variable definitions with the following structure declarations:

```
struct Package1               // Climbing Package
{
    int      num;             // Number in party
    int      beginners;       // Those needing instruction
    int      advanced;        // Those not needing instruction
    int      needEquip;       // Those renting camping equipment
    double   baseCharges;     // Base charges
    double   charges;         // Total charges
    double   instruction;     // Cost of instruction
    double   equipment;       // Cost of equipment rental
    double   discount;        // Discount
    double   deposit;         // Required deposit
};

struct Package2               // Scuba Package
{
    int      num;             // Number in party
    int      beginners;       // Those needing instruction
    int      advanced;        // Those not needing instruction
    double   baseCharges;     // Base charges
    double   charges;         // Total charges
    double   instruction;     // Cost of instruction
    double   discount;        // Discount
    double   deposit;         // Required deposit
};
```

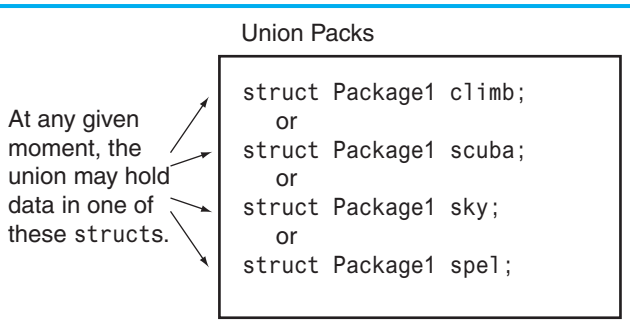
```
struct Package3              // Sky Diving Package
{
    int     num;             // Number in party
    int     lodge1;          // Number at 1st lodging choice
    int     lodge2;          // Number at 2nd lodging choice
    double  baseCharges;     // Base charges
    double  charges;         // Total charges
    double  discount;        // Discount
    double  lodging;         // Cost of lodging
    double  deposit;         // Required deposit
};

struct Package4              // Spelunking Package
{
    int     num;             // Number in party
    int     needEquip;       // Those renting camping equipment
    double  baseCharges;     // Base charges
    double  charges;         // Total charges
    double  equipment;       // Cost of equipment rental
    double  discount;        // Discount
    double  deposit;         // Required deposit
};
```

When the modifications to the program are complete, a record will be stored in a file each time a vacation package is sold. Because each record will record the data on a single package, the structures can be combined into a union (see Appendix K for information about unions). Here is the declaration:


```
union Pack
{
    struct Package1 climb;
    struct Package2 scuba;
    struct Package3 sky;
    struct Package4 spel;
};
```

Figure 1 illustrates that the union can hold in memory the data for any one of the structures at any given time.

Figure 1 The Packs union

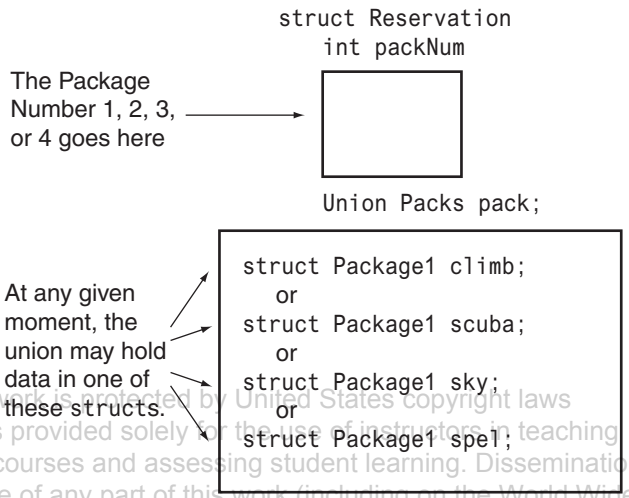


The last step is to create the following structure, which contains a variable identifying which package it holds data for:

```
struct Reservation
{
    int packNum;
    union Pack packs;
};
```

Figure 2 illustrates that the `Reservation` structure holds an integer and a union (which may represent one of its member structures at any time).

Figure 2 The `Reservation` structure



Now the program has a data type, `Reservation`, that can represent any of the four vacation packages. Function `main`, which follows, defines the `Reservation` variable `group`, which is passed by reference to the `climbing`, `scuba`, `skyDive`, or `spelunk` function, depending upon the package chosen by the user.

```
int main ()
{
    int selection;
    Reservation group;

    cout << fixed << showpoint << setprecision(2);
    do
    {
        selection = menu();
        switch(selection)
        {
            case 1 :  climbing(group);
                      break;
            case 2 :  scuba(group);
                      break;
            case 3 :  skyDive(group);
                      break;
            case 4 :  spelunk(group);
                      break;
            case 5 :  cout << "Exiting program.\n\n";
        }
```

```
    if (selection < 5)
        displayInfo(group);
    } while (selection != 5);
    return 0;
}
```

Here is the modified code for the `climbing` function:

```
void climbing(Reservation &group)
{
    group.packNum = 1;
    cout << "\nDevil's Courthouse Adventure Weekend\n";
    cout << "-----------------------------------\n";
    cout << "How many will be going who need an instructor? ";
    cin >> group.packs.climb.beginners;
    cout << "How many advanced climbers will be going? ";
    cin >> group.packs.climb.advanced;
    group.packs.climb.num = group.packs.climb.beginners +
                    group.packs.climb.advanced;
    cout << "How many will rent camping equipment? ";
    cin >> group.packs.climb.needEquip;
    // Calculate base charges.
    group.packs.climb.baseCharges = group.packs.climb.num *
                          CLIMB_RATE;
    group.packs.climb.charges = group.packs.climb.baseCharges;
    // Calculate 10% discount for 5 or more.
    if (group.packs.climb.num > 4)
    {
      group.packs.climb.discount = group.packs.climb.charges
                                   * .1;
      group.packs.climb.charges -= group.packs.climb.discount;
    }
    else
      group.packs.climb.discount = 0;
    // Add cost of instruction.
    group.packs.climb.instruction = group.packs.climb.beginners
                                    * CLIMB_INSTRUCT;
    group.packs.climb.charges += group.packs.climb.instruction;
    // Add cost of camping equipment rental
    group.packs.climb.equipment = group.packs.climb.needEquip *
                          DAILY_CAMP_RENTAL * 4;
    group.packs.climb.charges += group.packs.climb.equipment;
    // Calculate required deposit.
    group.packs.climb.deposit = group.packs.climb.charges / 2.0;
}
```


The function starts by storing the vacation package number in the `group.packNum` member:

```
group.packNum = 1;
```

This member indicates which vacation package has been purchased and which of the union's structures hold the data for the package.

Notice the hierarchy of the data structure indicated by the dot notation of each member name. For instance, the number of beginners in the party is stored in `group.packs.climb.beginners`. This name indicates that `beginners` is a member of `climb`, which is a member of `packs`, which is a member of `group`.

Each of the functions, `climbing`, `scuba`, `skyDive`, and `spelunk` have been modified to accept the `group` structure as an argument and work with the appropriate member variables. The `scuba` function works with the `group.packs.scuba` member, the `skyDive` function works with `group.packs.sky`, and the `spelunk` function works with `group.packs.spel`. Each function stores the correct number in `group.packNum` to indicate which member of the `packs` union is being used.

Notice outside the `switch` construct, if the user has not selected 5 to exit the program, the `displayInfo` function is called with `group` as an argument.

```
void displayInfo(const Reservation &group)
{
    switch (group.packNum)
    {
         case 1: displayPack1(group);
                 break;
         case 2: displayPack2(group);
                 break;
         case 3: displayPack3(group);
                 break;
         case 4: displayPack4(group);
                 break;
         default: cout << "ERROR: Invalid package number.\n";
    }
}
```


NOTE: Notice `group` is passed into a constant reference parameter. Because the structure is passed by reference, the program doesn't have to make a copy of it. This decreases the overhead of the function call, thus improving the program's performance. Because the function has no reason to modify the structure's contents, it's passed as a constant.

The `displayInfo` function looks at the contents of `group.packNum` to determine which package is represented by the structure, and calls one of four other functions to display its contents. If `group.packNum` holds the value 1, the structure is passed to the function `displayPack1`:

```
void displayPack1(const Reservation &group)
{
    cout << "Number in party: "
         << group.packs.climb.num << endl;
    cout << "Base charges: $"
         << group.packs.climb.baseCharges << endl;
    cout << "Instruction cost: $"
         << group.packs.climb.instruction << endl;
    cout << "Equipment rental: $"
         << group.packs.climb.equipment << endl;
    cout << "Discount: $"
         << group.packs.climb.discount << endl;
    cout << "Total charges: $"
         << group.packs.climb.charges << endl;
    cout << "Required deposit: $"
         << group.packs.climb.deposit << endl << endl;
}
```

If `group.packNum` holds the value 2, the function `displayPack2` is called:

```
void displayPack2(const Reservation &group)
{
    cout << "Number in party: "
         << group.packs.scuba.num << endl;
    cout << "Base charges: $"
         << group.packs.scuba.baseCharges << endl;
    cout << "Instruction cost: $"
         << group.packs.scuba.instruction << endl;
    cout << "Discount: $"
         << group.packs.scuba.discount << endl;
    cout << "Total charges: $"
         << group.packs.scuba.charges << endl;
    cout << "Required deposit: $"
         << group.packs.scuba.deposit << endl << endl;
}
```

If `group.packNum` contains the number 3, the function `displayPack3` is called:

```
void displayPack3(const Reservation &group)
{
    cout << "Number in party: "
         << group.packs.sky.num << endl;
    cout << "Base charges: $"
         << group.packs.sky.baseCharges << endl;
    cout << "Lodging: $"
         << group.packs.sky.lodging << endl;
    cout << "Discount: $"
         << group.packs.sky.discount << endl;
    cout << "Total charges: $"
         << group.packs.sky.charges << endl;
    cout << "Required deposit: $"
         << group.packs.sky.deposit << endl << endl;
}
```


Finally, `displayPack4` is called if `group.packNum` holds the value 4:

```
void displayPack4(const Reservation &group)
{
    cout << "Number in party: "
         << group.packs.spel.num << endl;
    cout << "Base charges: $"
         << group.packs.spel.baseCharges << endl;
    cout << "Equipment rental: $"
         << group.packs.spel.equipment << endl;
    cout << "Discount: $"
         << group.packs.spel.discount << endl;
    cout << "Total charges: $"
         << group.packs.spel.charges << endl;
    cout << "Required deposit: $"
         << group.packs.spel.deposit << endl << endl;
}
```

For the entire program, see Program CS5-1 in the Student Sample Source Code files.