Let’s revisit our mailbox program. Each user had a mailbox that contained, among other data, four folders (a folder being an array of structs representing an e-mail message), Inbox, Received, Sent, and Deleted. All the user mailboxes were stored as an array. The code to declare the array of user mailboxes looked like:

```c
const int MAX_USERS = 10;
const int MAX_MESSAGES = 100;

typedef string Address;

struct Name {
    string FirstName;
    string LastName;
};

struct Date {
    int Month;
    int Day;
    int Year;
};

struct EMailMessage {
    Name Sender;
    Address ReturnAddress;
    Date MessageDate;
    string Subject;
    string MessageText;
};

typedef EMailMessage Folder [MAX_MESSAGES];

struct UserMailbox {
    Name UserName;
    Address UserAddress;
    Folder Inbox;
    int NumInboxMessages;   // Number of messages in Inbox
    Folder Received;
    int NumReceivedMessages; // Number of messages in Received
    Folder Sent;
    int NumSentMessages;   // Number of messages in Sent
    Folder Deleted;
    int NumDeletedMessages;   // Number of messages in Deleted
};

int main()
{
    UserMailbox Mailboxes[MAX_USERS];
    return 0;
}
```
void PrintSummary(const UserMailbox[], const Address&);
int FindAddress(const UserMailbox[], const Address&, int);
void PrintMailboxSummary(const Folder, int);

void PrintSummary(const UserMailbox Mailboxes[], const Address& Target) {
    int Location;
    Location = FindAddress(Mailboxes, Target, MAX_USERS);
    if (Location == MISSING) {
        cout << Target << ": Invalid address" << endl;
        return;
    }
    PrintMailboxSummary(Mailboxes[Location].Inbox,
                         Mailboxes[Location].NumInboxMessages);
}

int FindAddress(const UserMailbox Mailboxes[], const Address& Target, int Size) {
    int i;
    for (i = 0; i < Size; i++)
        if (Mailboxes[i].UserAddress == Target)
            return i;
    return MISSING;
}

void PrintMailboxSummary(const Folder CurrentFolder, int Size) {
    int i;
    for (i = 0; i < Size; i++) {
        cout << CurrentFolder[i].ReturnAddress << "\t" <<
             CurrentFolder[i].MessageDate.Month << "/" <<
             CurrentFolder[i].MessageDate.Day << "/" <<
             CurrentFolder[i].MessageDate.Year << "\t" <<
             CurrentFolder[i].Subject << endl;
    }
}
2) Rather than separate variables for each folder, we can create an array of folders (and an associated array of integers for the number of messages in each folder). To simplify accessing the arrays, we can create an enumerated type. Write the required enumerated type and rewrite the definition for `UserMailbox` to include the two arrays previously described, rather than the eight individual variables.

Solution:

```c
enum FolderTypes {INBOX, RECEIVED, SENT, DELETED};
cost int NUM_FOLDERS = 4;

struct UserMailbox {
    Name UserName;
    Address UserAddress;
    
    Folder MailFolders[NUM_FOLDERS];
    int NumMessages[NUM_FOLDERS];
};
```

3) A function that prints information on messages in a folder has the prototype:

```c
void PrintMessageInformation(const Folder CurrentFolder, int NumMessages);
```

Using the new format described in problem 2, write a function invocation that takes the Sent folder of a mailbox `CurrentMailbox`.

Solution:

```c
PrintMessageInformation(CurrentMailbox.MailFolders[SENT],
                        CurrentMailbox.NumMessages[SENT]);
```