Last week we considered the problem of creating a program that stores mail messages for multiple users. 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 Recieved;
    int NumRecievedMessages; // Number of messages in Recieved
    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;
}
```
We want to write a function to initialize the array to dummy values. However, this is a fairly complex data structure, so attempting to do so in a single function would be complicated and messy. The solution is to divide and conquer – decompose the problem, write a function to solve the basic initialization problem, and use that function to solve a slightly more complicated one. We can continue this process until we’re able to write the main initialization function (which was our original problem). The first several questions ask you to write a function definition to accomplish some initialization task. For each subsequent function, you must use the function that you previously defined as part of your definition.

1) Write a function definition for a function that takes a single EmailMessage and initializes it.

    const Address NoAddress = "nobody@nowhere.com";
    // Initialize a struct at declaration
    const Date NoDate = {0, 0, 0};

    void InitMessage(EMailMessage& Message)
    {
        Message.Sender.FirstName = "No First Name";
        Message.Sender.LastName = "No LastName";
        Message.ReturnAddress = NoAddress;
        Message.MessageDate = NoDate;
        Message.Subject = "No Subject";
        Message.MessageText = "No Message Text";
    }

2) Write a function that takes a single folder and initializes it.

    void InitFolder(Folder CurrentFolder)
    {
        int i = 0;
        for (i = 0; i < MAX_MESSAGES; i++)
            InitMessage(CurrentFolder[i]);
    }
3) Write a function that initializes the mailbox of a single user.

```c
void InitUserMailbox(UserMailbox &Mailbox)
{
    Mailbox.UserName.FirstName = "No First Name";
    Mailbox.UserName.LastName = "No Last Name";
    Mailbox.UserAddress = NoAddress;

    InitFolder(Mailbox.Inbox);
    Mailbox.NumInboxMessages = 0;
    InitFolder(Mailbox.Recieved);
    Mailbox.NumRecievedMessages = 0;
    InitFolder(Mailbox.Sent);
    Mailbox.NumSentMessages = 0;
    InitFolder(Mailbox.Deleted);
    Mailbox.NumDeletedMessages = 0;
}
```

4) Finally, the original problem – write a function to initialize the array of mailboxes.

```c
void InitAllMailboxes(UserMailbox Mailboxes[])
{
    int i;

    for (i = 0; i < MAX_USERS; i++)
        InitUserMailbox(Mailboxes[i]);
}
```
5) Assuming the binary search function described in the notes, what values should go in the blanks?

```cpp
const int MISSING = -1;

bool FoundSubject(const UserMailbox& Mailbox)
{
    int Location;
    string TargetSubject;

    cout << "Enter subject line: " << endl;
    cin >> TargetSubject;

    Location = BinarySearch(Mailbox.Inbox,
                            TargetSubject, __________, __________);

    if (Location != MISSING)
        return true;
    return false;
}

Solution: BinarySearch(Mailbox.Inbox, TargetSubject,
                        __________, __________);
```

6) In the notes, the code for a linear search has a loop that looks like:

```cpp
while ((Scan < Size) && (List[Scan] != Target))
    Scan++;
```

If the target is not found, the last test of the Boolean expression will have the variable `Scan` equal to `Size`. However, `List[Size]` (from the second part of the expression) is outside of the array bounds. Will this cause a problem? Why or why not?

**Solution:** For the last test of the Boolean expression, `Scan < Size` will fail. Short-circuit evaluation will prevent the second half of the AND expression from ever being evaluated.