Assignment #1 Discussions

C++ Object Oriented Programming
Pei-yih Ting
95/03 NTOU CS

- 作業批改不是幫你 debug, 但是我會盡量給你意見, 可以作為以後寫程式的參考
- 寫書面說明時你一定會發現要說明一個程式的重點不是那麼簡單的, 你需要組織你的想法: 說明你所用的資料結構與演算法是很基礎的作法, 你需要把你自己寫的程式抽象化為一些圖片, 一些簡要的處理概念, 不要被支微末節迷惑, 如此你以後才能夠和別人討論程式中的設計, 大家才能夠合作建構大型的軟體
- 作業請列印出來, 我才能夠很快地幫你看你的作業,你繳交的東西很多, 我就花很多的時間看, 盡量給你意見, 你繳交的東西很少, 我沒有辦法給你什麼意見, 將來作業變得大一點時, 會讓大家選擇性的列印

- 如果你的作業是抄襲的，你會完全不曉得究竟需要練習什麼，後續的作業，實習，課程中的很多觀念對你就沒有用途，不要浪費這樣的資源
- 成績和你寫的程式功能不見得有絕對的關係，但是會盡量考量你付出的學習時間和你學到的東西，所以你的報告裡要顯示這樣的資訊

Design Problems

- Separation of Programming Tools and System Objects
  - Files: contains name and data
  - List of Files: contains the linkage information for the linked list
  - Folders: contains name, files, and subfolders
  - List of Folders: contains the linkage information for the linked list

```
Typedef struct _File
{
   char *name;
   char *content;
   struct _File *next;
} File, *FilePtr;
```

vs.

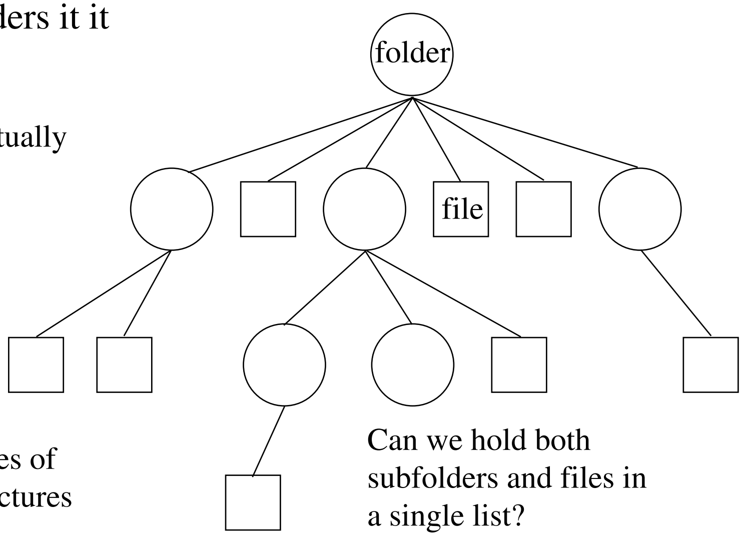
```
typedef struct _File
{
   char *name;
   char *content;
} File, *FilePtr;

typedef struct _FileNode
{
   struct _File *file;
   struct _FileNode *next;
} FileNode, *FileNodePtr;
```

Design Problems

✧ How about folders: if a folder can hold both files and subfolders it it

Conceptually



Two types of data structures

Can we hold both subfolders and files in a single list?

Design Problems

✧ Separate Lists:

```
typedef
struct _Folder
{
   char *name;
   struct _FolderNode *subfolderListHead;
   struct _FileNode *fileListHead;
} Folder, *FolderPtr;

typedef
struct _File
{
   char *name;
   char *content;
} File, *FilePtr;

typedef
struct _FolderNode
{
   struct _Folder *folder;
   struct _FolderNode *next;
} FolderNode, *FolderNodePtr;

typedef
struct _FileNode
{
   struct _File *file;
   struct _FileNode *next;
} FileNode, *FileNodePtr;
```

Problem: handle both lists separately, difficult to sort both lists together

Design Problems

✧ Single Heterogeneous List

```
typedef
struct _Folder
{
   char *name;
   void *folderAndFileListHead;
} Folder, *FolderPtr;
```

Problems:

1. Cast the node type explicitly
2. Still need to determine which node is a folder, which node is a file at the processing stage

Format Problems

✧ Do not include unnecessary declarations with #include
  - ✶ It might contain some unwanted declarations, cause compilation problems.

✧ Identifier naming are crucial, use a rule consistently
  - ✶ Function names are mostly verb phrases representing operations
    query(), printAllFiles(), print_All_Files(), print_all_files()
  - ✶ Variable names are usually nouns or noun phrases
    files, numberOfFiles, folder, folders, currentFolder, ptrToCurrentFolder, current_folder, ptr_to_current_folder
  - ✶ Type names by typedef are usually capitalized nouns or tagged nouns
    File, Folder, file_t, folder_t

Temporary Variables

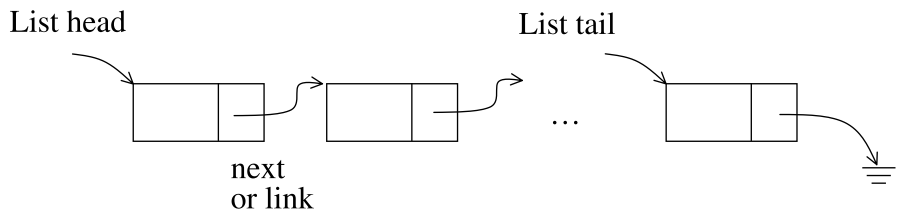
- Sometime we see temporary variable names temp, i, j, …
  If you know the role the data to be kept in these variables in the algorithm, you should give it a descriptive name like numberOfIterations, ptrToListTail, …

- A structure does not contain temporary variables

```
typedef struct _Folder
{
    char *name;
    struct _FolderNode *subfolderListHead;
    struct _FileNode *fileListHead;
    struct _FileNode *ptr;
} Folder, *FolderPtr;
```

Implementation Problem

- Map a conceptual model to your program
  ex. a Linked List



all the used variables must be named after their functions

Input Buffer Problem

- Sometimes we don't know the exact length of the input data at the moment when we write the program, we can use a huge buffer to overcome this problem, but the flexibility and efficiency issues force us to use dynamic memory allocation.
- Consider the following code snippet

```
char buf1[100], *buf2;
 ...
scanf("%s", buf1);
...
buf2 = (char *) malloc(strlen(buf1)+1);
strcpy(buf2, buf1);
...
free(buf2);
```

- There are still problems that do not make much sense:
  "the above code assumes that input is less than 100 characters"
  - Why not use static declaration buf2[100] if the assumption is true?
  - What if user input is longer than 100 characters? fgets(char *, int, FILE *)

Memory Allocation Problems

- Allocate a byte of memory, free a byte of memory.
  Good habit keeps you a respected software engineer.
  - malloc() … free()
- You can check this automatically with memory_leak.h and memory_leak.cpp
- Should a segment of dynamically allocated memory be used through pointer? I am afraid of pointer.><
  not necessarily!! you can use it as though it is array

```
int *data;
data = (int *) malloc(100*sizeof(int));
…
for (i=0; i<100; i++)
   data[i] = 0;
…
free(data);
```

Error Handling

- Memory allocation failure
  - ptr = (int *) malloc(100*sizeof(int));
    if (ptr == 0) { … error handling … }   or   assert(ptr!=0);
  - **Usually, providing "error handling codes" means that your program is designed to function even if the error occurs. Ex. if there is no 10000 bytes of memory available, your program can try using 1000 bytes of memory with a different algorithm**.
  - **Using assert() statement means that your program assumes that the operating system or operator should satisfy your basic demand, if not, your program just quit.**
- File open failure
  - fp = fopen("filename.dat");
    if (fp == 0) { … error handling …}   or   assert(fp!=0);
- Free a pointer
  - if (ptr!=0) free(ptr);   assert(ptr!=0);
    free(ptr);

心得

- 程式設計時的感想、收穫、與疑問
- 程式製作與偵錯時遇見的問題與歸納的結果
- 對於題目的建議