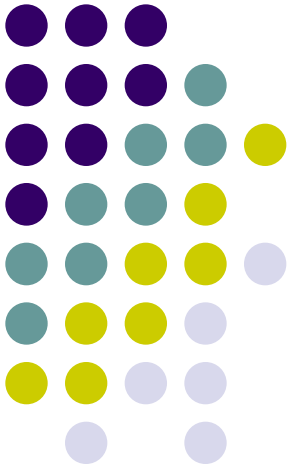
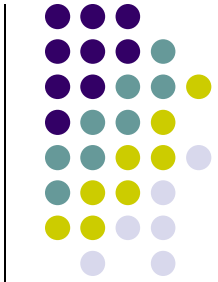
Data Structures
2011-2012 Spring

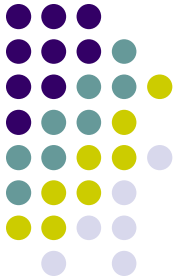
Practice Session 1



Contents



- Steps of compilation and linking a program
- File maintenance in Phonebook example
- Listing records in alphabetical order (by using Index array) in Phonebook example



Steps of Compilation and Linking

Compilation and Linking

- The executable file contains machine code for both programmer's code files and used libraries. Therefore, creation of executable file is a two-step process.

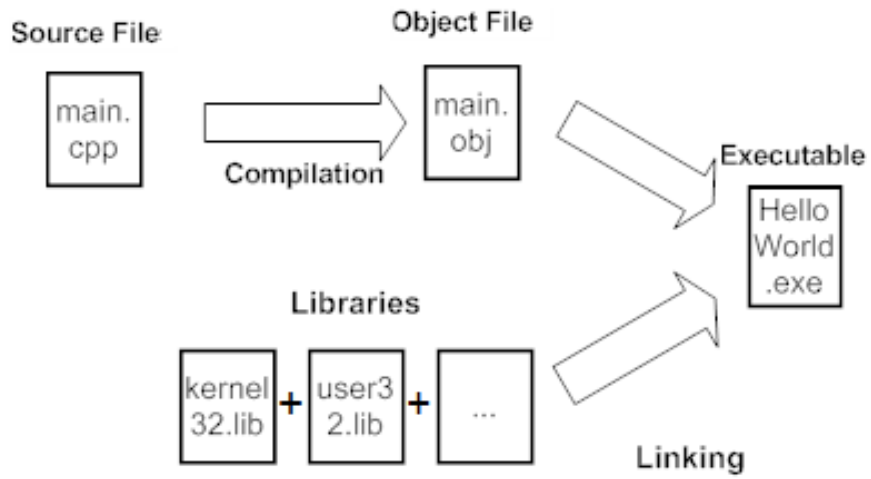
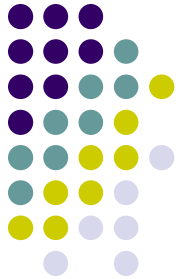


Figure 1: Compilation and linking steps in a project with a single code file.

Compilation and Linking



- **Compilation** refers to the processing of source code files (.c, .cc, or .cpp) and the creation of an 'object' file. This step doesn't create anything the user can actually run. Instead, the compiler merely produces the machine language instructions that correspond to the source code file that was compiled without linking libraries used in the code.
- **Linking:** Connections between object files and used libraries are established in this stage to produce a directly runnable program.

Compilation and Linking

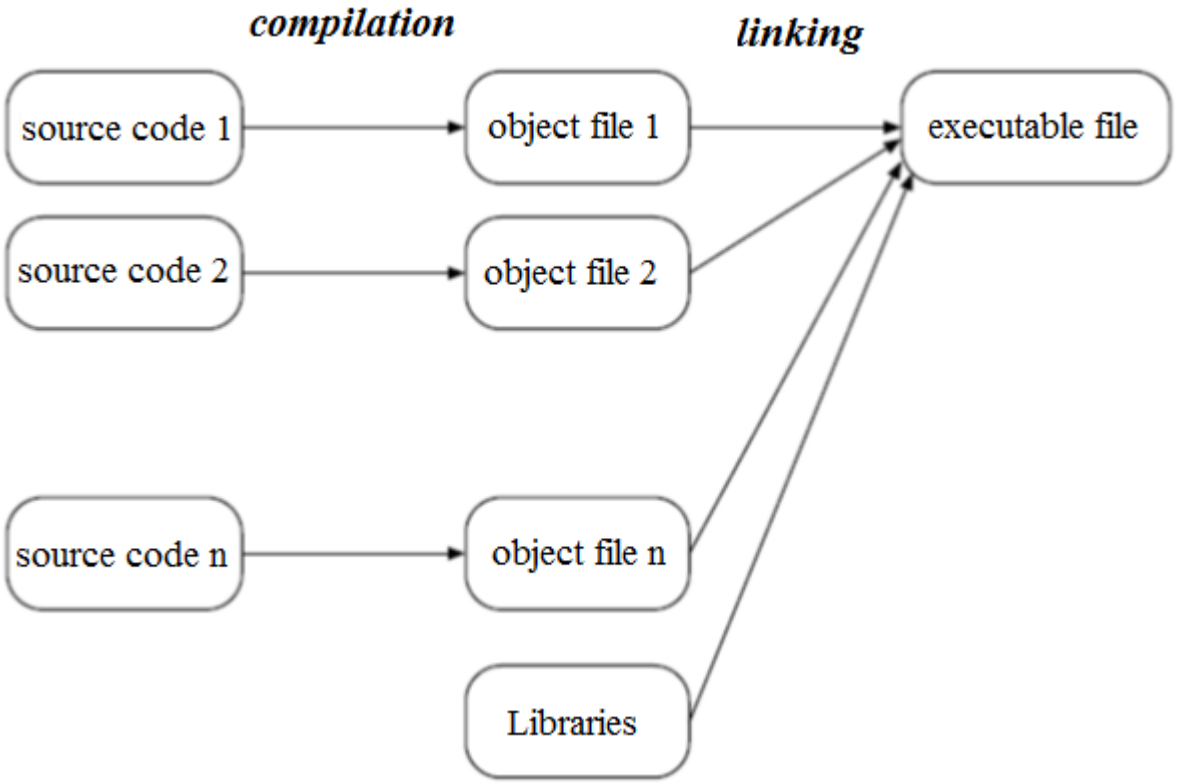
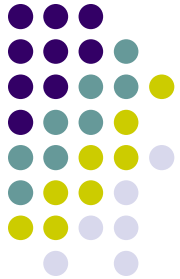


Figure 2: Compilation and linking steps in a project with multiple sourcecode files.

gcc and g++



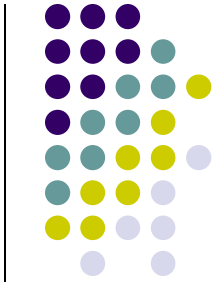
- In UNIX family operating systems GNU is used to compile and link a C(or C++) program:

  gcc *sourcecode.c* –o *executable_file*

- If source code is written in C++:

  g++ *sourcecode.cpp* –o *executable_file*

Compilation

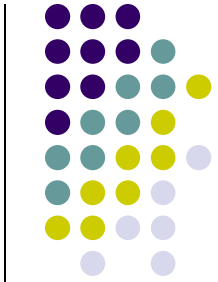


- Compiler searches for header files under «/usr/include» folder and library files under «/usr/lib» folder.

- By using –c flag, the sourcecode file can be compiled into an object file without the linking process.

g++ *main.cpp* –c –o *main.o*

Compilation



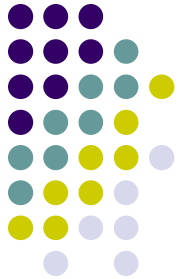
- **I flag:** To include another folder for searching header files:

  g++ *main.cpp* –I/usr/X11R/include –o *main*

- **L Flag:** To include another folder for searching library files:

  g++ *main.cpp* -L/usr/X11R/lib –o *main*

Compilation and Linking



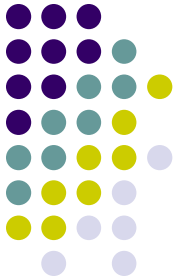
- Compilation and linking is done with multiple commands for projects with multiple source code files.
- **For example:** A project consisting of file1.cpp and file2.cpp can be compiled as follows:

  g++ *file1.cpp* –c –o *file1.o*

  g++ *file2.cpp* –c –o *file2.o*

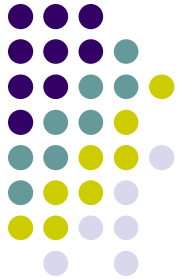
- And linking is done by,

  g++ *file1.o file1.o –o project*



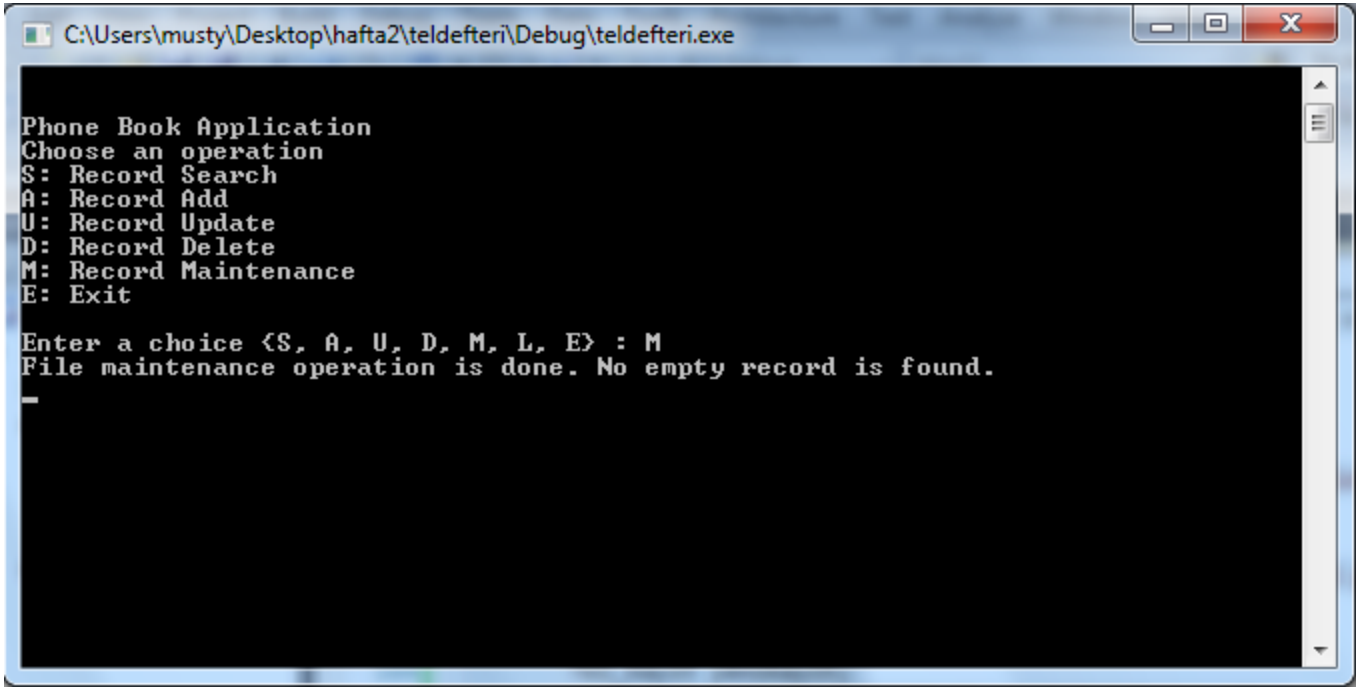
File Maintenace After Delete Operation

Record Deletion



- In the example shown in class, in order to delete a record, an empty record is copied into the file on the place of the record to be removed. However, in this case, undesired empty spaces occur in the file.
- A file maintenance operation will be done in order to remove this undesired empty spaces from the file.

Example Screenshot

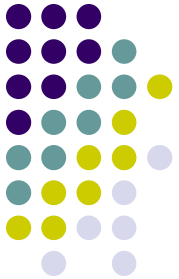


File Maintenance

```
void file_maintenance(){
    int emptyrecord = phonebook.filemaintenance();
    if (emptyrecord == 0)
        cout << "File maintenance operation is done. No empty record is found."<< endl;

    else
        cout << "File maintenance operation is done." << emptyrecord
             << " empty records are deleted from file."<<endl;

    getchar();
};
```

file_maintenance function given above is added to the main program.

File Maintenance

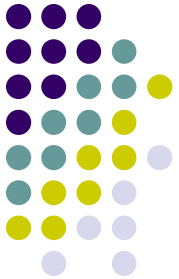


```
#ifndef FILEOPERATIONS_H
#define FILEOPERATIONS_H
#include "record.h"
#include <stdio.h>

typedef struct file{
    char *filename;
    FILE *phonebook;
    void create();
    void close();
    void add(Phone_Record *);
    int search(char []);
    void remove(int recordnum);
    void update(int recordnum, Phone_Record *);
    int filemaintenance();
} File;
#endif
```

filemaintenance function is added into fileoperations.h’

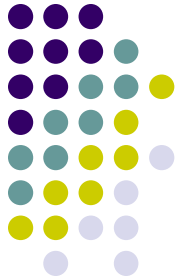
File Maintenance

```
int File::filemaintenance(){
    char *tempfilename = "tempphonebook.txt";
    FILE *tempphonebook;
    Phone_Record r;
    int counter = 0;
    tempphonebook = fopen(tempfilename, "w+");
    if(!tempphonebook){
            cerr << "Temporary file cannot be opened" << endl;
            exit(1);
    }
    fseek(phonebook, 0, SEEK_SET);
    while(!feof(phonebook)){
        fread( &r, sizeof (Phone_Record), 1, phonebook);
        if(feof(phonebook)) break;
        if((strcmp(r.name,"")==0))
            counter ++;
        else
            fwrite(&r, sizeof (Phone_Record), 1, tempphonebook);

    }
    if(counter > 0){
        fclose(phonebook);
        fclose(tempphonebook);
        char command[500]="copy ";   //For copying in Linux, use cp command instead.
        strcat(command,tempfilename);
        strcat(command," ");
        strcat(command,filename);
        system(command);
        create();
    }
    return counter;
}
```



Listing Records in Alphabetical Order

Sorted Listing



- For sorted listing of records in the file, an **index** array is used to store the alphabetical order information.
- **index** array is modified when a record is added/deleted/updated.
- Disadvantage of this method is due to the cost of array operations and limit of array size.

Sorted Listing

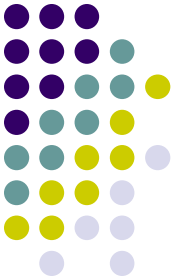
```
#ifndef FILEOPERATIONS_H
#define FILEOPERATIONS_H
#include "record.h"
#include <stdio.h>
#define MAXRECORDS 100

struct File{
    char *filename;
    FILE *phonebook;
    int index[MAXRECORDS];
    void create();
    void close();
    void add(Phone_Record *);
    int search(char []);
    void remove(int recordnum);
    void update(int recordnum, Phone_Record *);
    int filemaintenance();
    void createarray();
    int recordcount;
};
#endif
```

createarray() function and recordcount variable is added to ‘’fileoperations.h’’.

Sorted Listing

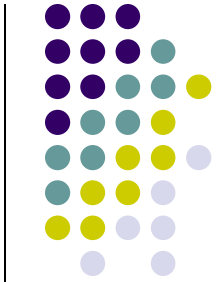
```
void File::createarray(){
    Phone_Record r1,r2;
    int tmp1,tmp2;
    int counter;
    fseek(phonebook, 0, SEEK_SET);
    while(!feof(phonebook)){
        fseek(phonebook, recordcount*sizeof(Phone_Record), SEEK_SET);
        fread( &r1, sizeof(Phone_Record), 1, phonebook);
        if(feof(phonebook)) break;
        recordcount++;
        counter = 0;
        for(int i=0;i<recordcount-1;i++){
            fseek(phonebook, index[i]*sizeof(Phone_Record), SEEK_SET);
            fread( &r2, sizeof(Phone_Record), 1, phonebook);
            if((strcmp(r1.name, r2.name)>0))
                counter++;
            else
                break;
        }
        tmp1 = index[counter];
        index[counter] = recordcount-1;
        counter++;
        while(counter<=recordcount){
            tmp2 = index[counter];
            index[counter] = tmp1;
            tmp1 = tmp2;
            counter++;
        }
    }
}
```

Sorted Listing



```
void File::create(){
    filename="phonebook.txt";
    phonebook = fopen(filename, "r+");
    if(!phonebook){
        phonebook = fopen(filename, "w");
        fclose(phonebook);
        phonebook = fopen(filename, "r+");
        if(!phonebook){
            cerr << "File cannot be opened" << endl;
            exit(1);
        }
    }
    recordcount=0;
    index[0]=0;
    createarray();
}
```

Modified Add Function

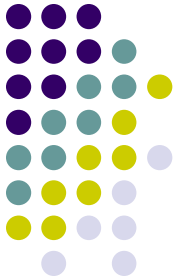


```
void File::add(Phone_Record *nrptr){
    Phone_Record r;
    int tmp1,tmp2;
    int counter = 0;
    for(int i=0;i<recordcount;i++){
        fseek(phonebook, index[i]*sizeof(Phone_Record), SEEK_SET);
        fread( &r, sizeof(Phone_Record), 1, phonebook);
          if((strcmp(r.name, nrptr->name)<0))
                counter++;
          else
              break;
    }
    recordcount++;
    tmp1 = index[counter];
    index[counter]=recordcount-1;
    counter++;
    while(counter<=recordcount){
        tmp2 = index[counter];
        index[counter] = tmp1;
        tmp1 = tmp2;
        counter++;
    }
    fseek(phonebook, 0, SEEK_END);
    fread(nrptr, sizeof(Phone_Record), 1, phonebook);
}
```

Modified Search Function

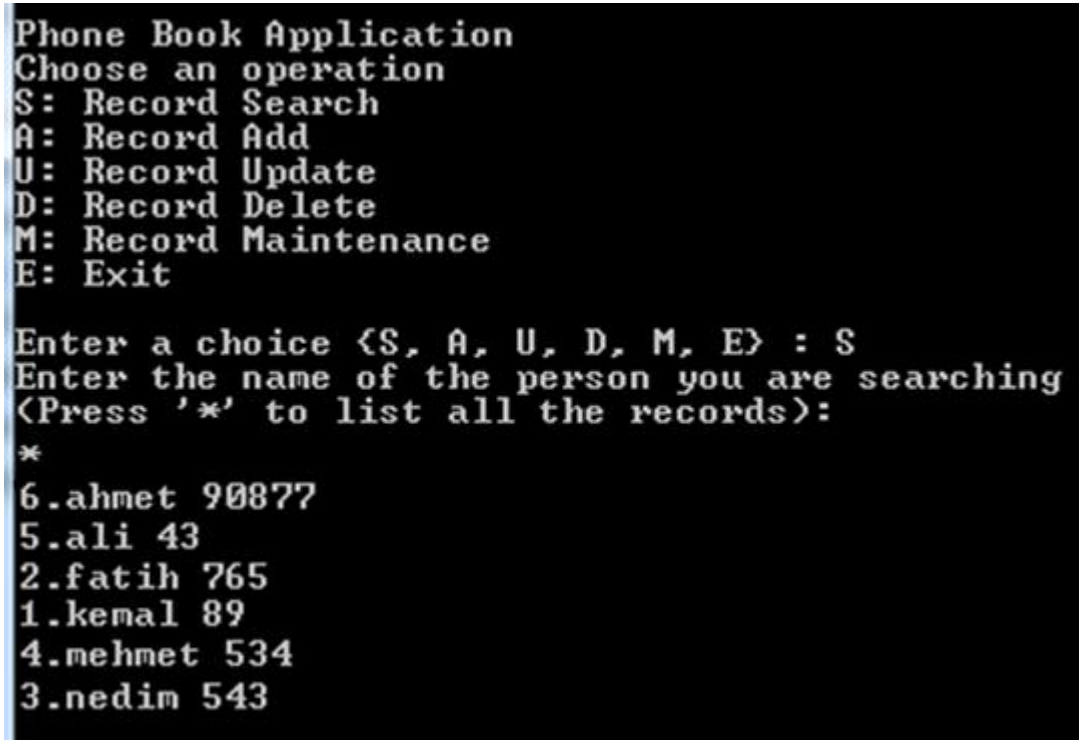
```
int File::search(char tosearch[]){
    Phone_Record r;
    int counter = 0;
    bool all = false;
    int found = 0;
    if(strcmp(tosearch,"*")==0)
        all = true;
    fseek(phonebook, 0, SEEK_SET);
    for(int i=0;i<recordcount;i++){
        fseek(phonebook, index[i]*sizeof(Phone_Record), SEEK_SET);
        fread( &r, sizeof(Phone_Record), 1, phonebook);
        if(all && (strcmp(r.name,"")!=0)){
            counter++;
            cout << index[i]+1 << "." << r.name << " " << r.phonenum << endl;
            found++;
        }
        if(!all && strnicmp(r.name, tosearch, strlen(tosearch))==0){
            cout << index[i]+1 << "." << r.name << " " << r.phonenum << endl;
            found++;
        }
    }
    return found;
}
```

Modified Update Function



```
void File::update(int recordnum, Phone_Record *nrptr){
    if(fseek(phonebook, sizeof(Phone_Record)*(recordnum-1), SEEK_SET)==0)
        fwrite(nrptr, sizeof(Phone_Record), 1, phonebook);
    recordcount = 0;
    index[0] = 0;
    createarray();
}
```

Sorted Listing



Sorted Listing

```
Phone Book Application
Choose an operation
S: Record Search
A: Record Add
U: Record Update
D: Record Delete
M: Record Maintenance
E: Exit

Enter a choice (S, A, U, D, M, E) : S
Enter the name of the person you are searching
(Press '*' to list all the records):
a
6.ahmet 90877
5.ali 43
```

Sorted Listing

- After deleting the record with number 5.

```
Phone Book Application
Choose an operation
S: Record Search
A: Record Add
U: Record Update
D: Record Delete
M: Record Maintenance
E: Exit

Enter a choice (S, A, U, D, M, E) : S
Enter the name of the person you are searching
(Press '*' to list all the records):
*
6.ahmet 90877
2.fatih 765
1.kemal 89
4.mehmet 534
3.nedim 543
```

Sorted Listing

- After updating the record with number 1 as ayse.

```
Phone Book Application
Choose an operation
S: Record Search
A: Record Add
U: Record Update
D: Record Delete
M: Record Maintenance
E: Exit

Enter a choice <S, A, U, D, M, E> : S
Enter the name of the person you are searching
(Press '*' to list all the records):
*
6.ayse 90877
2.fatih 765
1.kemal 89
4.mehmet 534
3.nedim 543
```

References

- For compilation and linking section, the following source is used:

  H. Turgut Uyar, “Programlamaya Giriş Ders Notları”, Şubat 2004.