

Appointment Manager (AMR)

Generated by Doxygen 1.8.8

Wed Apr 8 2015 21:34:51

Contents

1	Main Page	1
2	Data Structure Index	3
2.1	Data Structures	3
3	File Index	5
3.1	File List	5
4	Data Structure Documentation	7
4.1	Appointment Struct Reference	7
4.1.1	Detailed Description	7
4.2	AppointmentList Struct Reference	7
4.2.1	Detailed Description	8
4.3	AppointmentType Struct Reference	8
4.3.1	Detailed Description	8
4.4	Summary Struct Reference	8
4.4.1	Detailed Description	8
4.5	User Struct Reference	9
4.5.1	Detailed Description	9
5	File Documentation	11
5.1	appointment_list.c File Reference	11
5.1.1	Detailed Description	12
5.1.2	LICENSE	12
5.1.3	Function Documentation	12
5.1.3.1	AddAppointment	12
5.1.3.2	AddAppointmentFromList	13
5.1.3.3	AddAppointmentOrdered	13
5.1.3.4	AddAppointmentOrderedFromList	13
5.1.3.5	CompareAppointment	13
5.1.3.6	CompareAppointmentPriority	14
5.1.3.7	ConflictInList	14
5.1.3.8	PrintAppointment	14

5.1.3.9	PrintAppointmentList	14
5.1.4	Variable Documentation	14
5.1.4.1	AppointmentTypeStr	14
5.2	appointment_list.c	15
5.3	appointment_list.h File Reference	18
5.3.1	Detailed Description	19
5.3.2	LICENSE	20
5.3.3	Function Documentation	20
5.3.3.1	AddAppointment	20
5.3.3.2	AddAppointmentFromList	20
5.3.3.3	AddAppointmentOrdered	20
5.3.3.4	AddAppointmentOrderedFromList	20
5.3.3.5	CompareAppointment	21
5.3.3.6	CompareAppointmentPriority	21
5.3.3.7	ConflictInList	21
5.3.3.8	PrintAppointment	21
5.3.3.9	PrintAppointmentList	22
5.4	appointment_list.h	22
5.5	main.c File Reference	23
5.5.1	Detailed Description	23
5.5.2	LICENSE	24
5.5.3	Variable Documentation	24
5.5.3.1	NumOfUser	24
5.5.3.2	user	24
5.6	main.c	24
5.7	scheduler.c File Reference	28
5.7.1	Detailed Description	28
5.7.2	LICENSE	28
5.8	scheduler.c	29
5.9	scheduler.h File Reference	34
5.9.1	Detailed Description	34
5.9.2	LICENSE	34
5.10	scheduler.h	35
5.11	user.c File Reference	35
5.11.1	Detailed Description	35
5.11.2	LICENSE	36
5.11.3	Function Documentation	36
5.11.3.1	GetUserID	36
5.12	user.c	36
5.13	user.h File Reference	37

5.13.1 Detailed Description	38
5.13.2 LICENSE	38
5.13.3 Macro Definition Documentation	38
5.13.3.1 MAX_USERNAME	38
5.13.3.2 USER_NUMBER	38
5.13.4 Function Documentation	38
5.13.4.1 GetUserID	38
5.13.5 Variable Documentation	39
5.13.5.1 NumOfUser	39
5.13.5.2 user	39
5.14 user.h	39
Index	40

Chapter 1

Main Page

An appointment management software that have the calendar and scheduling function.

Build

To compile the program. “ make “ The executable program is located in bin/.

To clean up the object files. “ make clean “

To clean up the object files and the executable file. “ make remove “

To join the source files into one AMR.c file “ make onefile “

Documentation

Project documentation: doc/AMRReport.pdf

API documentation: doc/API.pdf

File structure

-bin/ executable file -src/ source and header files -obj/ object files during make -doc/ documentations
-test/in .. testing input -test/out . testing output

How to test

Use input redirection ./bin/amr alice bob charlie < test/in/fcfs.in ./bin/amr alice bob charlie < test/in/prio.in ./bin/amr
alice bob charlie < test/in/opti.in

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Appointment	
Store a appointment record	7
AppointmentList	
A double-linked list for appointment record	7
AppointmentType	
Store all the appointment type	8
Summary	8
User	
Store the basic information of the user and the appointments	9

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

appointment_list.c	
Handling the appointments and appointment list	11
appointment_list.h	
Handling the appointments and appointment list	18
main.c	
Appointment (p. 7) Manager (AMR) main program. Also the input handling	23
scheduler.c	
Secheduling algorithms	28
scheduler.h	
Secheduling algorithms	34
user.c	
Handling each users	35
user.h	
Handling each users	37

Chapter 4

Data Structure Documentation

4.1 Appointment Struct Reference

Store a appointment record.

```
#include <appointment_list.h>
```

Data Fields

- enum **AppointmentType** **type**
- int **id**
- int **caller_id**
- int **callee_id** [10]
- time_t **start**
- time_t **end**
- int **is_accepted**
- int **rescheduled**
- char **reason** [50]
- struct **Appointment** * **prev**
- struct **Appointment** * **next**

4.1.1 Detailed Description

Store a appointment record.

Definition at line **48** of file **appointment_list.h**.

The documentation for this struct was generated from the following file:

- **appointment_list.h**

4.2 AppointmentList Struct Reference

A double-linked list for appointment record.

```
#include <appointment_list.h>
```

Data Fields

- int **count**
- struct **Appointment** * **head**
- struct **Appointment** * **tail**

4.2.1 Detailed Description

A double-linked list for appointment record.

Definition at line **67** of file **appointment_list.h**.

The documentation for this struct was generated from the following file:

- **appointment_list.h**

4.3 AppointmentType Struct Reference

Store all the appointment type.

```
#include <appointment_list.h>
```

4.3.1 Detailed Description

Store all the appointment type.

The documentation for this struct was generated from the following file:

- **appointment_list.h**

4.4 Summary Struct Reference

Data Fields

- int **total_accepted**
- int **total_rejected**
- int **accepted** [USER_NUMBER]
- int **rejected** [USER_NUMBER]
- int **empty_timeslot** [USER_NUMBER]
- time_t **start**
- time_t **end**

4.4.1 Detailed Description

Definition at line **28** of file **scheduler.h**.

The documentation for this struct was generated from the following file:

- **scheduler.h**

4.5 User Struct Reference

Store the basic information of the user and the appointments.

```
#include <user.h>
```

Data Fields

- char **username** [MAX_USERNAME]
- struct **AppointmentList** * **accepted**
- struct **AppointmentList** * **rejected**

4.5.1 Detailed Description

Store the basic information of the user and the appointments.

Definition at line **40** of file **user.h**.

The documentation for this struct was generated from the following file:

- **user.h**

Chapter 5

File Documentation

5.1 appointment_list.c File Reference

Handling the appointments and appointment list.

```
#include "appointment_list.h"
#include "user.h"
```

Functions

- struct **AppointmentList** * **CreateAppointmentList** ()
Create a appointment list and init the value.
- struct **Appointment** * **CreateAppointment** ()
Create a appointment and init the value.
- void **AddAppointment** (struct **AppointmentList** *list, const struct **Appointment** *newItem)
Insert a copy of the appointment into the end of the appointment list.
- void **AddAppointmentOrdered** (struct **AppointmentList** *list, const struct **Appointment** *newItem)
Insert a copy of the appointment into the sorted appointment list. Using the start time then end time as the sorting condition.
- void **AddAppointmentFromList** (struct **AppointmentList** *dst_list, const struct **AppointmentList** *src_list)
Insert a copy of the appointment into the end of the appointment list.
- void **AddAppointmentOrderedFromList** (struct **AppointmentList** *dst_list, const struct **AppointmentList** *src_list)
Insert a copy of the appointment into the sorted appointment list. Using the start time then end time as the sorting condition.
- int **CompareAppointment** (const struct **Appointment** *a, const struct **Appointment** *b)
Compare the start time and then end time of the appointment. Used to keep the ordered appointment list.
- int **CompareAppointmentPriority** (const struct **Appointment** *a, const struct **Appointment** *b)
Compare the appointment by it's priority.
- void **RemoveItemFromList** (struct **AppointmentList** *list, const struct **Appointment** *item)
Remove an item from the list. Items should be unique inside the list. Delete if the two item have the same id.
- void **RemoveListFromList** (struct **AppointmentList** *ori_list, const struct **AppointmentList** *del_list)
Remove a list of items from the list. Items should be unique inside the list. Delete if the two item have the same id.
- void **PrintAppointment** (const struct **Appointment** *item)
Print out the appointment.
- void **PrintAppointmentList** (const struct **AppointmentList** *list)
Print out the appointment list.

- int **IsConflict** (const struct **Appointment** *a, const struct **Appointment** *b)
Check whether if two appointments have time conflict.
- int **IsConflictInList** (const struct **AppointmentList** *list, const struct **Appointment** *item)
Check whether if the appointment item have conflict with the list.
- struct **AppointmentList** * **ConflictInList** (const struct **AppointmentList** *list, const struct **Appointment** *item)
Check whether the the new appointment is conflict with the existing appointments that are already in the list.
- struct **Appointment** * **GetAppointmentById** (const struct **AppointmentList** *list, int id)
Return the appointment that match the id in the list.
- void **SetReasonForList** (struct **AppointmentList** *list, const char *reason)
Set the reject reason.

Variables

- const char * **AppointmentTypeStr** []

5.1.1 Detailed Description

Handling the appointments and appointment list.

Author

oneonestar oneonestar@gmail.com

Version

1.0

Copyright

2015

5.1.2 LICENSE

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Definition in file **appointment_list.c**.

5.1.3 Function Documentation

5.1.3.1 void AddAppointment (struct AppointmentList * list, const struct Appointment * newItem)

Insert a copy of the appointment into the end of the appointment list.

Parameters

out	<i>list</i>	The destination appointment list.
in	<i>newItem</i>	The item that needs to add into the list.

Definition at line 65 of file **appointment_list.c**.

5.1.3.2 void AddAppointmentFromList (struct AppointmentList * *dst_list*, const struct AppointmentList * *src_list*)

Insert a copy of the appointment into the end of the appointment list.

Parameters

out	<i>list</i>	The destination appointment list.
in	<i>newItem</i>	The item that needs to add into the list.

Definition at line 125 of file **appointment_list.c**.

5.1.3.3 void AddAppointmentOrdered (struct AppointmentList * *list*, const struct Appointment * *newItem*)

Insert a copy of the appointment into the sorted appointment list. Using the start time then end time as the sorting condition.

Parameters

out	<i>list</i>	The destination appointment list.
in	<i>newItem</i>	The item that needs to add into the list.

Definition at line 84 of file **appointment_list.c**.

5.1.3.4 void AddAppointmentOrderedFromList (struct AppointmentList * *dst_list*, const struct AppointmentList * *src_list*)

Insert a copy of the appointment into the sorted appointment list. Using the start time then end time as the sorting condition.

Parameters

out	<i>list</i>	The destination appointment list.
in	<i>newItem</i>	The item that needs to add into the list.

Definition at line 135 of file **appointment_list.c**.

5.1.3.5 int CompareAppointment (const struct Appointment * *a*, const struct Appointment * *b*)

Compare the start time and then end time of the appointment. Used to keep the ordered appointment list.

Parameters

in	<i>a</i>	Appointment (p. 7) to be compared.
in	<i>b</i>	Appointment (p. 7) to be compared.

Return values

< 0	a is before b
0	a is equal to b

>0	a is after b
------	--------------

Definition at line **145** of file **appointment_list.c**.

5.1.3.6 int CompareAppointmentPriority (const struct Appointment * a, const struct Appointment * b)

Compare the appointment by it's priority.

Parameters

in	a	Appointment (p. 7) to be compared.
in	b	Appointment (p. 7) to be compared.

Return values

<0	a is before b
0	a is equal to b
>0	a is after b

Definition at line **155** of file **appointment_list.c**.

5.1.3.7 struct AppointmentList* ConflictInList (const struct AppointmentList * list, const struct Appointment * item)

Check whether the the new appointment is conflict with the existing appointments that are already in the list.

Parameters

in	list	The destination appointment list.
in	newItem	The item that needs to add into the list.

Definition at line **254** of file **appointment_list.c**.

5.1.3.8 void PrintAppointment (const struct Appointment * item)

Print out the appointment.

Parameters

item	Appointment (p. 7) to be printed.
------	--

Definition at line **192** of file **appointment_list.c**.

5.1.3.9 void PrintAppointmentList (const struct AppointmentList * list)

Print out the appointment list.

Parameters

list	Appointment (p. 7) list to be printed.
------	---

Definition at line **224** of file **appointment_list.c**.

5.1.4 Variable Documentation

5.1.4.1 const char* AppointmentTypeStr[]

Initial value:

```
= {[STUDY] = "Study", [ASSIGNMENT] = "Assignment",
  [PROJECT] = "Project", [GATHERING] = "Gathering"}
```

For printing

Definition at line 26 of file `appointment_list.c`.

5.2 appointment_list.c

```
00001
00022 #include "appointment_list.h"
00023 #include "user.h"
00024
00026 const char *AppointmentTypeStr[] = {[STUDY] = "Study", [ASSIGNMENT] = "Assignment",
00027   [PROJECT] = "Project", [GATHERING] = "Gathering"};
00028
00029 /*****
00030  * Implementation
00031  *****/
00032 struct AppointmentList* CreateAppointmentList()
00033 {
00034     struct AppointmentList *list = (struct AppointmentList*)malloc(sizeof(struct
AppointmentList));
00035     if(!list)
00036     {
00037         fprintf(stderr, "Failed to allocate memory.\n");
00038         exit(EXIT_FAILURE);
00039     }
00040     list->count = 0;
00041     list->head = NULL;
00042     list->tail = NULL;
00043     return list;
00044 }
00045
00046 struct Appointment* CreateAppointment()
00047 {
00048     struct Appointment *item = (struct Appointment*)malloc(sizeof(struct
Appointment));
00049     if(!item)
00050     {
00051         fprintf(stderr, "Failed to allocate memory.\n");
00052         exit(EXIT_FAILURE);
00053     }
00054     for(int i=0; i<USER_NUMBER; i++)
00055         item->callee_id[i] = -1;
00056     item->is_accepted = 0;
00057     item->id = -1;
00058     item->rescheduled = 0;
00059     item->prev = NULL;
00060     item->next = NULL;
00061     strcpy(item->reason, "");
00062     return item;
00063 }
00064
00065 void AddAppointment(struct AppointmentList *list, const struct Appointment *newItem)
00066 {
00067     struct Appointment *item = CreateAppointment();
00068     *item = *newItem;
00069     item->next = item->prev = 0;
00070     if(!list->head) //if the list is empty
00071     {
00072         list->head = item;
00073         list->tail = item;
00074     }
00075     else
00076     {
00077         list->tail->next = item;
00078     }
00079     item->prev = list->tail;
00080     list->tail = item;
00081     list->count++;
00082 }
00083
00084 void AddAppointmentOrdered(struct AppointmentList *list, const struct
Appointment *newItem)
00085 {
00086     struct Appointment *item = CreateAppointment();
00087     *item = *newItem;
00088     item->next = item->prev = 0;
00089     struct Appointment *ptr = list->head;
00090     //if the list is empty
00091     if(!ptr)
```

```

00092     {
00093         list->head = item;
00094         list->tail = item;
00095     }
00096     else if(CompareAppointment(item, ptr)<0)    //if insert at the head
00097     {
00098         if(!list->head)
00099             list->head->prev = item;
00100         item->next = list->head;
00101         list->head = item;
00102     }
00103     else    //insert at middle or at the tail
00104     {
00105         while(ptr->next)    //find the insertion position
00106         {
00107             if(difftime(item->start, ptr->next->start)<0)
00108                 break;
00109             ptr = ptr->next;
00110         }
00111         if(!ptr)    //insert at the tail
00112         {
00113             ptr = list->tail;
00114             list->tail = item;
00115         }
00116         item->prev = ptr;    //insert after ptr
00117         item->next = ptr->next;
00118         if(item->next)
00119             item->next->prev = item;
00120         ptr->next = item;
00121     }
00122     list->count++;
00123 }
00124
00125 void AddAppointmentFromList(struct AppointmentList *dst_list, const struct
AppointmentList *src_list)
00126 {
00127     struct Appointment *newItem = src_list->head;
00128     while(newItem)
00129     {
00130         AddAppointment(dst_list, newItem);
00131         newItem = newItem->next;
00132     }
00133 }
00134
00135 void AddAppointmentOrderedFromList(struct AppointmentList *dst_list, const struct
AppointmentList *src_list)
00136 {
00137     struct Appointment *newItem = src_list->head;
00138     while(newItem)
00139     {
00140         AddAppointmentOrdered(dst_list, newItem);
00141         newItem = newItem->next;
00142     }
00143 }
00144
00145 int CompareAppointment(const struct Appointment *a, const struct Appointment *b)
00146 {
00147     if(difftime(a->start, b->start)<0)
00148         return -1;    //a before b
00149     else if(difftime(a->start, b->start)==0)
00150         return difftime(a->end, b->end);
00151     else
00152         return 1;
00153 }
00154
00155 int CompareAppointmentPriority(const struct Appointment *a, const struct
Appointment *b)
00156 {
00157     return a->type - b->type;
00158 }
00159
00160 void RemoveItemFromList(struct AppointmentList *list, const struct Appointment *item)
00161 {
00162     struct Appointment *delItem = list->head;
00163     while(delItem)
00164     {
00165         if(delItem->id == item->id)
00166         {
00167             if(!delItem->prev)    //if prev is null, first item in list
00168                 list->head = delItem->next;
00169             else
00170                 delItem->prev->next = delItem->next;
00171             if(!delItem->next)    //if next is null, last item in list
00172                 list->tail = delItem->prev;
00173             else
00174                 delItem->next->prev = delItem->prev;
00175             list->count--;

```

```

00176         return;
00177     }
00178     delItem = delItem->next;
00179 }
00180 }
00181
00182 void RemoveListFromList(struct AppointmentList *ori_list, const struct
AppointmentList *del_list)
00183 {
00184     struct Appointment *delItem = del_list->head;
00185     while(delItem)
00186     {
00187         RemoveItemFromList(ori_list, delItem);
00188         delItem = delItem->next;
00189     }
00190 }
00191
00192 void PrintAppointment(const struct Appointment *item)
00193 {
00194     struct tm tm_start, tm_end;
00195     memcpy(&tm_start, localtime (&item->start), sizeof(struct tm));
00196     memcpy(&tm_end, localtime (&item->end), sizeof(struct tm));
00197     printf("%2d ", item->id);
00198     printf("%4d-%02d-%02d %02d:%02d %02d:%02d %-12s ", tm_start.tm_year+1900, tm_start.tm_mon+1,
tm_start.tm_mday, tm_start.tm_hour,
00199 tm_start.tm_min, tm_end.tm_hour, tm_end.tm_min, AppointmentTypeStr[item->type]);
00200     if(item->rescheduled)
00201         printf("%-9c ", 'Y');
00202     else
00203         printf("%-9c ", 'N');
00204     if(strcmp(item->reason, ""))
00205     {
00206         printf(" %s", item->reason);
00207     }
00208     else
00209     {
00210         if(item->callee_id[0] == -1)
00211             printf("-");
00212         else
00213             printf("%s ", user[item->caller_id].username);
00214         for(int i=0; i<USER_NUMBER; i++)
00215         {
00216             if(item->callee_id[i]==-1)
00217                 break;
00218             printf("%s ", user[item->callee_id[i]].username);
00219         }
00220     }
00221     printf("\n");
00222 }
00223
00224 void PrintAppointmentList(const struct AppointmentList *list)
00225 {
00226     struct Appointment *ptr = list->head;
00227     while(ptr!=NULL)
00228     {
00229         PrintAppointment(ptr);
00230         ptr = ptr->next;
00231     }
00232 }
00233
00234 int IsConflict(const struct Appointment *a, const struct Appointment *b)
00235 {
00236     return !(difftime(a->end, b->start)<=0 || //a before b
00237 difftime(a->start, b->end)>=0); //a after b
00238 }
00239
00240 int IsConflictInList(const struct AppointmentList *list, const struct
Appointment *item)
00241 {
00242     if(!list || !item)
00243         return 0;
00244     struct Appointment *ptr = list->head;
00245     while(ptr)
00246     {
00247         if(IsConflict(ptr, item))
00248             return 1;
00249         ptr = ptr->next;
00250     }
00251     return 0;
00252 }
00253
00254 struct AppointmentList* ConflictInList(const struct AppointmentList *list, const struct
Appointment *item)
00255 {
00256     if(!list || !item)
00257         return NULL;
00258     struct AppointmentList *conflict_list = CreateAppointmentList();

```

```

00259     struct Appointment *ptr = list->head;
00260     while(ptr)
00261     {
00262         if(IsConflict(ptr, item))
00263             AddAppointment(conflict_list, ptr);
00264         ptr = ptr->next;
00265     }
00266     return conflict_list;
00267 }
00268
00269 struct Appointment* GetAppointmentById(const struct AppointmentList *list, int id)
00270 {
00271     if(!list)
00272         return NULL;
00273     struct Appointment *ptr = list->head;
00274     while(ptr)
00275     {
00276         if(ptr->id == id)
00277             return ptr;
00278         ptr = ptr->next;
00279     }
00280     return NULL;
00281 }
00282
00283
00284 void SetReasonForList(struct AppointmentList *list, const char *reason)
00285 {
00286     if(!list)
00287         return;
00288     struct Appointment *ptr = list->head;
00289     while(ptr)
00290     {
00291         strcpy(ptr->reason, reason);
00292         ptr = ptr->next;
00293     }
00294 }

```

5.3 appointment_list.h File Reference

Handling the appointments and appointment list.

```

#include <ctype.h>
#include <math.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "user.h"

```

Data Structures

- struct **Appointment**
Store a appointment record.
- struct **AppointmentList**
A double-linked list for appointment record.

Enumerations

- enum **AppointmentType** { **ASSIGNMENT** = 0, **PROJECT**, **STUDY**, **GATHERING** }

Functions

- struct **Appointment** * **CreateAppointment** ()
Create a appointment and init the value.

- struct **AppointmentList** * **CreateAppointmentList** ()
Create a appointment list and init the value.
- void **AddAppointment** (struct **AppointmentList** *list, const struct **Appointment** *newItem)
Insert a copy of the appointment into the end of the appointment list.
- void **AddAppointmentOrdered** (struct **AppointmentList** *list, const struct **Appointment** *newItem)
Insert a copy of the appointment into the sorted appointment list. Using the start time then end time as the sorting condition.
- void **AddAppointmentFromList** (struct **AppointmentList** *dst_list, const struct **AppointmentList** *src_list)
Insert a copy of the appointment into the end of the appointment list.
- void **AddAppointmentOrderedFromList** (struct **AppointmentList** *dst_list, const struct **AppointmentList** *src_list)
Insert a copy of the appointment into the sorted appointment list. Using the start time then end time as the sorting condition.
- struct **AppointmentList** * **ConflictInList** (const struct **AppointmentList** *list, const struct **Appointment** *item)
Check whether the the new appointment is conflict with the existing appointments that are already in the list.
- int **IsConflict** (const struct **Appointment** *a, const struct **Appointment** *b)
Check whether if two appointments have time conflict.
- int **IsConflictInList** (const struct **AppointmentList** *list, const struct **Appointment** *item)
Check whether if the appointment item have conflict with the list.
- void **RemoveItemFromList** (struct **AppointmentList** *list, const struct **Appointment** *item)
Remove an item from the list. Items should be unique inside the list. Delete if the two item have the same id.
- void **RemoveListFromList** (struct **AppointmentList** *ori_list, const struct **AppointmentList** *del_list)
Remove a list of items from the list. Items should be unique inside the list. Delete if the two item have the same id.
- int **CompareAppointment** (const struct **Appointment** *a, const struct **Appointment** *b)
Compare the start time and then end time of the appointment. Used to keep the ordered appointment list.
- int **CompareAppointmentPriority** (const struct **Appointment** *a, const struct **Appointment** *b)
Compare the appointment by it's priority.
- void **PrintAppointment** (const struct **Appointment** *item)
Print out the appointment.
- void **PrintAppointmentList** (const struct **AppointmentList** *list)
Print out the appointment list.
- struct **Appointment** * **GetAppointmentByld** (const struct **AppointmentList** *list, int id)
Return the appointment that match the id in the list.
- void **SetReasonForList** (struct **AppointmentList** *list, const char *reason)
Set the reject reason.

5.3.1 Detailed Description

Handling the appointments and appointment list.

Author

oneonestar oneonestar@gmail.com

Version

1.0

Copyright

2015

5.3.2 LICENSE

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Definition in file **appointment_list.h**.

5.3.3 Function Documentation

5.3.3.1 void AddAppointment (struct AppointmentList * *list*, const struct Appointment * *newItem*)

Insert a copy of the appointment into the end of the appointment list.

Parameters

out	<i>list</i>	The destination appointment list.
in	<i>newItem</i>	The item that needs to add into the list.

Definition at line **65** of file **appointment_list.c**.

5.3.3.2 void AddAppointmentFromList (struct AppointmentList * *dst_list*, const struct AppointmentList * *src_list*)

Insert a copy of the appointment into the end of the appointment list.

Parameters

out	<i>list</i>	The destination appointment list.
in	<i>newItem</i>	The item that needs to add into the list.

Definition at line **125** of file **appointment_list.c**.

5.3.3.3 void AddAppointmentOrdered (struct AppointmentList * *list*, const struct Appointment * *newItem*)

Insert a copy of the appointment into the sorted appointment list. Using the start time then end time as the sorting condition.

Parameters

out	<i>list</i>	The destination appointment list.
in	<i>newItem</i>	The item that needs to add into the list.

Definition at line **84** of file **appointment_list.c**.

5.3.3.4 void AddAppointmentOrderedFromList (struct AppointmentList * *dst_list*, const struct AppointmentList * *src_list*)

Insert a copy of the appointment into the sorted appointment list. Using the start time then end time as the sorting condition.

Parameters

out	<i>list</i>	The destination appointment list.
-----	-------------	-----------------------------------

<i>in</i>	<i>newItem</i>	The item that needs to add into the list.
-----------	----------------	---

Definition at line **135** of file **appointment_list.c**.

5.3.3.5 int CompareAppointment (const struct Appointment * a, const struct Appointment * b)

Compare the start time and then end time of the appointment. Used to keep the ordered appointment list.

Parameters

<i>in</i>	<i>a</i>	Appointment (p. 7) to be compared.
<i>in</i>	<i>b</i>	Appointment (p. 7) to be compared.

Return values

< 0	a is before b
0	a is equal to b
> 0	a is after b

Definition at line **145** of file **appointment_list.c**.

5.3.3.6 int CompareAppointmentPriority (const struct Appointment * a, const struct Appointment * b)

Compare the appointment by it's priority.

Parameters

<i>in</i>	<i>a</i>	Appointment (p. 7) to be compared.
<i>in</i>	<i>b</i>	Appointment (p. 7) to be compared.

Return values

< 0	a is before b
0	a is equal to b
> 0	a is after b

Definition at line **155** of file **appointment_list.c**.

5.3.3.7 struct AppointmentList* ConflictInList (const struct AppointmentList * list, const struct Appointment * item)

Check whether the the new appointment is conflict with the existing appointments that are already in the list.

Parameters

<i>in</i>	<i>list</i>	The destination appointment list.
<i>in</i>	<i>newItem</i>	The item that needs to add into the list.

Definition at line **254** of file **appointment_list.c**.

5.3.3.8 void PrintAppointment (const struct Appointment * item)

Print out the appointment.

Parameters

<i>item</i>	Appointment (p. 7) to be printed.
-------------	--

Definition at line 192 of file **appointment_list.c**.

5.3.3.9 void PrintAppointmentList (const struct AppointmentList * *list*)

Print out the appointment list.

Parameters

<i>list</i>	Appointment (p. 7) list to be printed.
-------------	---

Definition at line 224 of file **appointment_list.c**.

5.4 appointment_list.h

```

00001
00022 #ifndef APPOINTMENT_LIST
00023 #define APPOINTMENT_LIST
00024
00025 #include <ctype.h>
00026 #include <math.h>
00027 #include <signal.h>
00028 #include <stdio.h>
00029 #include <stdlib.h>
00030 #include <string.h>
00031 #include <time.h>
00032
00033 #include "user.h"
00034
00039 enum AppointmentType
00040 {
00041     ASSIGNMENT = 0, PROJECT, STUDY, GATHERING
00042 };
00043
00048 struct Appointment
00049 {
00050     enum AppointmentType type;
00051     int id;
00052     int caller_id;
00053     int callee_id[10];
00054     time_t start;
00055     time_t end;
00056     int is_accepted;
00057     int rescheduled;
00058     char reason[50];
00059     struct Appointment *prev;
00060     struct Appointment *next;
00061 };
00062
00067 struct AppointmentList
00068 {
00069     int count;
00070     struct Appointment *head;
00071     struct Appointment *tail;
00072 };
00073
00074
00078 struct Appointment* CreateAppointment();
00079
00083 struct AppointmentList* CreateAppointmentList();
00084
00085
00091 void AddAppointment(struct AppointmentList *list, const struct Appointment *newItem);
00092
00099 void AddAppointmentOrdered(struct AppointmentList *list, const struct
Appointment *newItem);
00100
00106 void AddAppointmentFromList(struct AppointmentList *dst_list, const struct
AppointmentList *src_list);
00107
00114 void AddAppointmentOrderedFromList(struct AppointmentList *dst_list, const struct
AppointmentList *src_list);
00115
00121 struct AppointmentList* ConflictInList(const struct AppointmentList *list, const struct
Appointment *item);
00122
00126 int IsConflict(const struct Appointment *a, const struct Appointment *b);

```

```

00127
00131 int IsConflictInList(const struct AppointmentList *list, const struct
    Appointment *item);
00132
00137 void RemoveItemFromList(struct AppointmentList *list, const struct Appointment *item);
00138
00143 void RemoveListFromList(struct AppointmentList *ori_list, const struct
    AppointmentList *del_list);
00144
00154 int CompareAppointment(const struct Appointment *a, const struct Appointment *b);
00155
00164 int CompareAppointmentPriority(const struct Appointment *a, const struct
    Appointment *b);
00165
00170 void PrintAppointment(const struct Appointment *item);
00171
00176 void PrintAppointmentList(const struct AppointmentList *list);
00177
00181 struct Appointment* GetAppointmentById(const struct AppointmentList *list, int id);
00182
00186 void SetReasonForList(struct AppointmentList *list, const char *reason);
00187
00188 #endif

```

5.5 main.c File Reference

Appointment (p. 7) Manager (AMR) main program. Also the input handling.

```

#include <ctype.h>
#include <math.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <sys/wait.h>
#include <errno.h>
#include "appointment_list.h"
#include "scheduler.h"
#include "user.h"

```

Functions

- void **HandleInput** (const char *line)
- void **HandleSchedule** (const char *algorithm)
- void **inputLoop** (FILE *stream)
- int **main** (int argc, char *argv[])

Variables

- int **NumOfUser**
- struct **User** **user** [**USER_NUMBER**]
- struct **AppointmentList** * **inputList**

5.5.1 Detailed Description

Appointment (p. 7) Manager (AMR) main program. Also the input handling.

Author

oneonestar oneonestar@gmail.com

Version

1.0

Copyright

2015

5.5.2 LICENSE

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Definition in file **main.c**.

5.5.3 Variable Documentation**5.5.3.1 int NumOfUser**

Global variable storing current number of user.

Definition at line **55** of file **user.h**.

5.5.3.2 struct User user[USER_NUMBER]

Global variable storing the user data.

Definition at line **50** of file **user.h**.

5.6 main.c

```

00001
00023 #include <ctype.h>
00024 #include <math.h>
00025 #include <signal.h>
00026 #include <stdio.h>
00027 #include <stdlib.h>
00028 #include <string.h>
00029 #include <time.h>
00030 #include <unistd.h>
00031 #include <sys/wait.h>
00032 #include <errno.h>
00033
00034 #include "appointment_list.h"
00035 #include "scheduler.h"
00036 #include "user.h"
00037
00038 extern int NumOfUser;
00039 extern struct User user[USER_NUMBER];
00040
00041 struct AppointmentList *inputList;
00042
00043 void HandleInput(const char *line);
00044 void HandleSchedule(const char *algorithm);
00045 void inputLoop(FILE *stream);
00046

```

```

00047 void HandleSchedule(const char *algorithm)
00048 {
00049     if(inputList->count == 0)
00050     {
00051         printf("Empty timetable.\n");
00052         return;
00053     }
00054
00055 #ifndef NO_FORK
00056     struct Summary *summary = NULL;
00057     //TODO: remove items from list
00058     for (int i=0; i<NumOfUser; i++)
00059     {
00060         user[i].accepted = CreateAppointmentList();
00061         user[i].rejected = CreateAppointmentList();
00062     }
00063     if(!strcmp(algorithm, "-fcfs"))
00064         summary = Scheduling_FCFS(inputList);
00065     else if(!strcmp(algorithm, "-prio"))
00066         summary = Scheduling_PRIO(inputList);
00067     else if(!strcmp(algorithm, "-opti"))
00068         summary = Scheduling_OPTI(inputList);
00069     else
00070     {
00071         printf("Unknown scheduler.\n");
00072         return;
00073     }
00074     PrintAllUser();
00075     PrintSummary(summary);
00076 #else
00077     struct Summary *summary;
00078     int fd[2];
00079     if (pipe(fd) < 0) {
00080         printf("Pipe creation error\n");
00081         exit(EXIT_FAILURE);
00082     }
00083
00084     int ret=fork();
00085     if (ret < 0)
00086     {
00087         printf("error in fork!");
00088         exit(EXIT_FAILURE);
00089     }
00090     else if (ret == 0) { //Child
00091         //TODO: remove items from list
00092         for (int i=0; i<NumOfUser; i++)
00093         {
00094             user[i].accepted = CreateAppointmentList();
00095             user[i].rejected = CreateAppointmentList();
00096         }
00097         if(!strcmp(algorithm, "-fcfs"))
00098             summary = Scheduling_FCFS(inputList);
00099         else if(!strcmp(algorithm, "-prio"))
00100             summary = Scheduling_PRIO(inputList);
00101         else if(!strcmp(algorithm, "-opti"))
00102             summary = Scheduling_OPTI(inputList);
00103         else
00104         {
00105             printf("Unknown scheduler.\n");
00106             return;
00107         }
00108         PrintAllUser();
00109         if(write(fd[1], summary, sizeof(struct Summary)) < 0)
00110             printf("Oh dear, something went wrong with write()! %s\n", strerror(errno));
00111         _exit(EXIT_SUCCESS);
00112     }
00113
00114     summary = (struct Summary *)malloc(sizeof(struct Summary));
00115     if(read(fd[0], summary, sizeof(struct Summary)) < 0)
00116         printf("Oh dear, something went wrong with read()! %s\n", strerror(errno));
00117     wait(NULL);
00118     PrintSummary(summary);
00119 #endif
00120 }
00121
00122 void HandleInput(const char *line)
00123 {
00124     char command[25];
00125     char caller[MAX_USERNAME];
00126     int year, month, day;
00127     int hour, minutes;
00128     float duration;
00129     int callee_count = 0;
00130     char *pch;
00131
00132     struct Appointment *item = CreateAppointment();
00133     char myLine[255];

```

```

00134     strcpy(myLine, line);
00135
00136     //parse the command
00137     pch = strtok(myLine, " \n");
00138     if(!pch)
00139         goto UNKNOWN; //eg. strtok("\n", " \n") will return null from strtok, goto unknown command
00140     strcpy(command, pch);
00141     if(!strcmp(command, "addStudy"))
00142         item->type = STUDY;
00143     else if(!strcmp(command, "addAssignment"))
00144         item->type = ASSIGNMENT;
00145     else if(!strcmp(command, "addProject"))
00146         item->type = PROJECT;
00147     else if(!strcmp(command, "addGathering"))
00148         item->type = GATHERING;
00149     else if(!strcmp(command, "addBatch"))
00150     {
00151         pch = strtok(NULL, " \n");
00152         char filename[255];
00153         strcpy(filename, pch);
00154         FILE *f = fopen(filename+1, "r"); //offset +1 to remove the '-'
00155         if(!f)
00156         {
00157             fprintf(stderr, "Failed to open file %s.\n", filename);
00158             return;
00159             // exit(EXIT_FAILURE);
00160         }
00161         inputLoop(f);
00162         return;
00163     }
00164     else if(!strcmp(command, "printSchd"))
00165     {
00166         pch = strtok(NULL, " \n");
00167         char algorithmStr[30];
00168         strcpy(algorithmStr, pch);
00169         HandleSchedule(algorithmStr);
00170         return;
00171     }
00172     else if(!strcmp(command, "endProgram"))
00173     {
00174         printf("Received end program command.\n");
00175         exit(EXIT_SUCCESS);
00176     }
00177     else
00178     {
00179         UNKNOWN:
00180         printf("Unknown command: %s\n", line);
00181         return;
00182     }
00183
00184     pch = strtok(NULL, " \n");
00185     strcpy(caller, pch+1);
00186
00187     pch = strtok(NULL, " \n");
00188     sscanf(pch, "%d-%d-%d", &year, &month, &day);
00189
00190     pch = strtok(NULL, " \n");
00191     sscanf(pch, "%d:%d", &hour, &minutes);
00192
00193     pch = strtok(NULL, " \n"); duration = atof(pch);
00194
00195     while(1)
00196     {
00197         pch = strtok(NULL, " \n");
00198         if(!pch)
00199             break;
00200         int id = GetUserID(pch);
00201         if(id==--1)
00202         {
00203             printf("->[Rejected: Unknown callee %s]\n", pch);
00204             return;
00205         }
00206         item->callee_id[callee_count++] = id;
00207     }
00208
00209     item->caller_id = GetUserID(caller);
00210     if(item->caller_id==--1)
00211     {
00212         printf("->[Rejected: Unknown caller %s]\n", caller);
00213         return;
00214     }
00215     //time
00216     struct tm timeinfo, timeinfo_tmp;
00217     memset(&timeinfo, 0, sizeof(timeinfo));
00218     timeinfo.tm_isdst = -1;
00219     timeinfo.tm_year = year - 1900;
00220     timeinfo.tm_mon = month - 1;

```



```

00221     timeinfo.tm_mday = day;
00222     //start time
00223     //convert ot half hour base
00224     timeinfo.tm_hour = hour;
00225     if(minutes>=0 && minutes <= 30)
00226         timeinfo.tm_min = 0;
00227     else
00228         timeinfo.tm_min = 30;
00229
00230     timeinfo_tmp = timeinfo;    //because mktime could modify the value
00231     item->start = mktime(&timeinfo_tmp);
00232     //convert duration to end time
00233     double _;
00234     double fractional = modf(duration, &_amp;);
00235     minutes += fractional*60;
00236     hour = hour+(int)duration;
00237     if(minutes>=60)
00238         hour++;
00239     minutes %= 60;
00240
00241     timeinfo.tm_hour = hour;
00242     if(minutes>0 && minutes <= 30)
00243         timeinfo.tm_min = 0;
00244     else
00245         timeinfo.tm_min = 30;
00246     item->end = mktime(&timeinfo);
00247
00248     item->id = inputList->count;
00249     AddAppointment(inputList, item);
00250     printf("-> [Pending]\n");
00251 }
00252
00253
00254 void inputLoop(FILE *stream)
00255 {
00256     const int MAX_CHAR = 255;
00257     char line[MAX_CHAR];
00258     char *return_val;
00259     while(1)
00260     {
00261         printf("Please enter appointment:\n");
00262         return_val = fgets(line, MAX_CHAR, stream);
00263         if(!return_val)
00264         {
00265             if(feof(stream))
00266             {
00267                 printf("Received EOF.\n");
00268                 return;
00269             }
00270             else
00271             {
00272                 fprintf(stderr, "IO error, existing program.\n");
00273                 return;
00274                 // exit(EXIT_FAILURE);
00275             }
00276         }
00277         HandleInput(line);
00278     }
00279 }
00280
00281 int main(int argc, char* argv[])
00282 {
00283     if (argc < 4 || argc > 11)
00284     {
00285         fprintf(stderr, "Error: The number of users should between 3 and 10.\n");
00286         return EXIT_FAILURE;
00287     }
00288     NumOfUser = argc - 1;
00289     //Initialize each user in struct user[];
00290     for (int i=0; i<NumOfUser; i++)
00291     {
00292         if(GetUserID(argv[i+1]) != -1)
00293         {
00294             printf("Duplicate names of users!\n");
00295             exit(EXIT_FAILURE);
00296         }
00297         strcpy(user[i].username, argv[i+1]);
00298         user[i].username[0] = toupper(user[i].username[0]);
00299         user[i].accepted = CreateAppointmentList();
00300         user[i].rejected = CreateAppointmentList();
00301     }
00302     inputList = CreateAppointmentList();
00303
00304     printf("~~WELCOME TO AMR~~\n");
00305     inputLoop(stdin);
00306     return EXIT_SUCCESS;
00307 }

```

5.7 scheduler.c File Reference

Scheduling algorithms.

```
#include "appointment_list.h"
#include "user.h"
#include <unistd.h>
#include "scheduler.h"
```

Functions

- struct **Summary** * **Schedual_FCFS** (struct **AppointmentList** *inputList)
First come first served. The order is following the input order. The result will be putted into the each user's appointment lists (accept / reject).
- struct **Summary** * **Schedual_PRIO** (struct **AppointmentList** *inputList)
Priority. The order is following the pre-defined priority. The result will be putted into the each user's appointment lists (accept / reject).
- struct **Summary** * **Schedual_OPTI** (struct **AppointmentList** *inputList)
Optimized. Bonus part, reschedule those rejected appointments. The result will be putted into the each user's appointment lists (accept / reject).
- void **PrintSummary** (struct **Summary** *summary)
Print out the summary about the scheduling.

5.7.1 Detailed Description

Scheduling algorithms.

Author

oneonestar oneonestar@gmail.com

Version

1.0

Copyright

2015

5.7.2 LICENSE

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Definition in file **scheduler.c**.

5.8 scheduler.c

```

00001
00021 #include "appointment_list.h"
00022 #include "user.h"
00023
00024 #include <unistd.h>
00025
00026 #include "scheduler.h"
00027
00028 static int AllAvailable(const struct Appointment *item)
00029 {
00030     if(!item)
00031         return 0;
00032     struct AppointmentList *temp_list;
00033     //check caller timetable
00034     temp_list = ConflictInList(user[item->caller_id].accepted, item);
00035     if(temp_list->count)
00036         return item->caller_id;
00037     //check callees timetable
00038     for(int i=0; i<USER_NUMBER; i++)
00039     {
00040         if(item->callee_id[i]==-1)
00041             break;
00042         temp_list = ConflictInList(user[item->callee_id[i]].accepted, item);
00043         if(temp_list->count)
00044             return item->callee_id[i];
00045     }
00046     return -1;
00047 }
00048
00049 static int AllAvailablePriority(const struct Appointment *item)
00050 {
00051     struct AppointmentList *temp_list;
00052     struct Appointment *ptr;
00053     //check caller timetable
00054     temp_list = ConflictInList(user[item->caller_id].accepted, item);
00055     ptr = temp_list->head;
00056     while(ptr)
00057     {
00058         if(CompareAppointmentPriority(item, ptr)>=0)    //if item has equal or lower priority
00059             return item->caller_id; //not available
00060         ptr = ptr->next;
00061     }
00062     //check callees timetable
00063     for(int i=0; i<USER_NUMBER; i++)
00064     {
00065         if(item->callee_id[i]==-1)
00066             break;
00067         //TODO: may need refactoring
00068         temp_list = ConflictInList(user[item->callee_id[i]].accepted, item);
00069         ptr = temp_list->head;
00070         while(ptr)
00071         {
00072             if(CompareAppointmentPriority(item, ptr)>=0)    //if item has equal or lower priority
00073                 return item->callee_id[i]; //not available
00074             ptr = ptr->next;
00075         }
00076     }
00077     return -1; //available
00078 }
00079
00080 static void AddToAllAccept(struct Appointment *item)
00081 {
00082     strcpy(item->reason, "");
00083     AddAppointmentOrdered(user[item->caller_id].accepted, item);
00084     for(int i=0; i<USER_NUMBER; i++)
00085     {
00086         if(item->callee_id[i]==-1)
00087             break;
00088         AddAppointmentOrdered(user[item->callee_id[i]].accepted, item);
00089     }
00090 }
00091
00092 static void AddToAllAcceptForced(struct Appointment *item)
00093 {
00094     struct AppointmentList *temp_list;
00095     strcpy(item->reason, "");
00096     //caller
00097     //delete old appointments from accepted list
00098     temp_list = ConflictInList(user[item->caller_id].accepted, item);
00099     RemoveListFromList(user[item->caller_id].accepted, temp_list);
00100     //add to accept and reject lsit
00101     AddAppointmentOrdered(user[item->caller_id].accepted, item);
00102     SetReasonForList(temp_list, "Higher priority item being added.");
00103     AddAppointmentOrderedFromList(user[item->caller_id].rejected, temp_list);

```

```

00104     for(int i=0; i<USER_NUMBER; i++)
00105     {
00106         if(item->callee_id[i]==-1)
00107             break;
00108         //delete old appointments from accepted list
00109         temp_list = ConflictInList(user[item->callee_id[i]].accepted, item);
00110         RemoveListFromList(user[item->callee_id[i]].accepted, temp_list);
00111         //add to accept and reject list
00112         SetReasonForList(temp_list, "Higher priority item being added.");
00113         AddAppointmentOrdered(user[item->callee_id[i]].accepted, item);
00114         AddAppointmentOrderedFromList(user[item->callee_id[i]].rejected, temp_list);
00115     }
00116 }
00117
00118 static void AddToAllReject(const struct Appointment *item)
00119 {
00120     AddAppointmentOrdered(user[item->caller_id].rejected, item);
00121     for(int i=0; i<USER_NUMBER; i++)
00122     {
00123         if(item->callee_id[i]==-1)
00124             break;
00125         AddAppointmentOrdered(user[item->callee_id[i]].rejected, item);
00126     }
00127 }
00128
00130 static void SetAppointmentAccepted(struct AppointmentList *inputList)
00131 {
00132     for(int i=0; i<NumOfUser; i++)
00133     {
00134         struct Appointment *ptr = user[i].accepted->head;
00135         while(ptr)
00136         {
00137             GetAppointmentById(inputList, ptr->id)->is_accepted = 1;
00138             ptr = ptr->next;
00139         }
00140     }
00141 }
00142
00143 static struct AppointmentList* GetEmptyTimeSlotInDay(struct AppointmentList *userList, time_t date)
00144 {
00145     struct AppointmentList *empty_list = CreateAppointmentList();
00146     //18:00-22:00
00147     struct tm timeinfo, timeinfo_tmp;
00148     timeinfo = *localtime (&date);
00149
00150     //foreach 18:00-22:00 half hour timeslot
00151     timeinfo.tm_hour = 18;
00152     while(timeinfo.tm_hour<22)
00153     {
00154         struct Appointment *item = CreateAppointment();
00155         //first half hour
00156         timeinfo_tmp = timeinfo; //because mktime could modify the value
00157         item->start = mktime(&timeinfo_tmp);
00158         timeinfo.tm_min = 30;
00159         timeinfo_tmp = timeinfo; //because mktime could modify the value
00160         item->end = mktime(&timeinfo_tmp);
00161         if(!IsConflictInList(userList, item))
00162             AddAppointmentOrdered(empty_list, item);
00163
00164         //second half hour
00165         timeinfo_tmp = timeinfo; //because mktime could modify the value
00166         item->start = mktime(&timeinfo_tmp);
00167         timeinfo.tm_hour++;
00168         timeinfo.tm_min = 0;
00169         timeinfo_tmp = timeinfo; //because mktime could modify the value
00170         item->end = mktime(&timeinfo_tmp);
00171         if(!IsConflictInList(userList, item))
00172             AddAppointmentOrdered(empty_list, item);
00173     }
00174     return empty_list;
00175 }
00176
00177 static struct AppointmentList* GetEmptyTimeSlotInRange(struct AppointmentList *userList, time_t start_date,
00178 time_t end_date)
00179 {
00180     struct AppointmentList *empty_list = CreateAppointmentList();
00181     struct tm timeinfo, timeinfo_tmp;
00182
00183     //set the time start<end so that we can use difftime() to compare the date.
00184     timeinfo = *localtime (&start_date);
00185     timeinfo.tm_hour = 1;
00186     timeinfo_tmp = timeinfo;
00187     start_date = mktime(&timeinfo_tmp);
00188
00189     timeinfo = *localtime (&end_date);
00190     timeinfo.tm_hour = 2;

```

```

00191     timeinfo_tmp = timeinfo;
00192     end_date = mktime(&timeinfo_tmp);
00193
00194     timeinfo = *localtime (&start_date);
00195     while(difftime(start_date, end_date)<0)
00196     {
00197         AddAppointmentFromList(empty_list, GetEmptyTimeSlotInDay(userList, start_date));
00198
00199         timeinfo.tm_mday++;
00200         timeinfo_tmp = timeinfo;
00201         start_date = mktime(&timeinfo_tmp);
00202     }
00203     return empty_list;
00204 }
00205
00206 static time_t GetEarliestStartTime(struct AppointmentList *list)
00207 {
00208     struct Appointment *item = list->head;
00209     time_t earliest = item->start;
00210     while(item)
00211     {
00212         if(difftime(item->start, earliest)<0)
00213             earliest = item->start;
00214         item = item->next;
00215     }
00216     return earliest;
00217 }
00218
00219 static time_t GetLatestEndTime(struct AppointmentList *list)
00220 {
00221     struct Appointment *item = list->head;
00222     time_t latest = item->end;
00223     while(item)
00224     {
00225         if(difftime(latest, item->end)<0)
00226             latest = item->end;
00227         item = item->next;
00228     }
00229     return latest;
00230 }
00231
00232 static struct AppointmentList* GetContinueTimeslotFromList(const struct
AppointmentList *list, time_t duration)
00233 {
00234     struct Appointment *ptr;
00235     struct Appointment *item = list->head;
00236     struct AppointmentList *ret_list = CreateAppointmentList();
00237     int timeslot = duration / 60 / 30 - 1; //how many half hour
00238     while(item)
00239     {
00240         ptr = item;
00241         for(int i=0; i<timeslot; i++)
00242             ptr = ptr->next;
00243         if(!ptr)
00244             return ret_list;
00245         if(difftime(ptr->end, item->start)==duration)
00246             AddAppointmentOrdered(ret_list, item);
00247         item = item->next;
00248     }
00249     return ret_list;
00250 }
00251
00252 struct Summary* Schedual_FCFS(struct AppointmentList *inputList)
00253 {
00254     struct Appointment *ptr = inputList->head;
00255     while(ptr)
00256     {
00257         int ret = AllAvailable(ptr);
00258         if(ret<0)
00259         {
00260             AddToAllAccept(ptr);
00261         }
00262         else
00263         {
00264             char reason[50];
00265             strcpy(reason, user[ret].username);
00266             strcat(reason, " is unavailable.");
00267             strcpy(ptr->reason, reason);
00268             AddToAllReject(ptr);
00269         }
00270         ptr = ptr->next;
00271     }
00272
00273     //Summary
00274     struct Summary *summary = (struct Summary *)malloc(sizeof(struct Summary));
00275     summary->start = GetEarliestStartTime(inputList);
00276     summary->end = GetLatestEndTime(inputList);

```

```

00277
00278     SetAppointmentAccepted(inputList);
00279     ptr = inputList->head;
00280     while(ptr)
00281     {
00282         if(ptr->is_accepted)
00283             summary->total_accepted++;
00284         else
00285             summary->total_rejected++;
00286         ptr = ptr->next;
00287     }
00288     for(int i=0; i<NumOfUser; i++)
00289     {
00290         summary->accepted[i] = user[i].accepted->count;
00291         summary->rejected[i] = user[i].rejected->count;
00292         summary->empty_timeslot[i] = GetEmptyTimeSlotInRange(user[i].accepted, summary->start, summary->end
00293     )->count;
00294
00295         // PrintAppointmentList(GetContinueTimeslotFromList(GetEmptyTimeSlotInRange(user[i].accepted,
00296         summary->start, summary->end), 2*60*30));
00297         // PrintAppointmentList(GetEmptyTimeSlotInRange(user[i].accepted, summary->start, summary->end));
00298     }
00299     return summary;
00300 }
00301
00302 struct Summary* Schedual_PRIO(struct AppointmentList *inputList)
00303 {
00304     struct Appointment *ptr = inputList->head;
00305     while(ptr)
00306     {
00307         int ret = AllAvailablePriority(ptr);
00308         if(ret<0)
00309         {
00310             AddToAllAcceptForced(ptr);
00311         }
00312         else
00313         {
00314             char reason[50];
00315             strcpy(reason, user[ret].username);
00316             strcat(reason, " is unavailable.");
00317             strcpy(ptr->reason, reason);
00318             AddToAllReject(ptr);
00319         }
00320         ptr = ptr->next;
00321     }
00322
00323     //Summary
00324     struct Summary *summary = (struct Summary *)malloc(sizeof(struct Summary));
00325     summary->start = GetEarliestStartTime(inputList);
00326     summary->end = GetLatestEndTime(inputList);
00327
00328     SetAppointmentAccepted(inputList);
00329     ptr = inputList->head;
00330     while(ptr)
00331     {
00332         if(ptr->is_accepted)
00333             summary->total_accepted++;
00334         else
00335             summary->total_rejected++;
00336         ptr = ptr->next;
00337     }
00338     for(int i=0; i<NumOfUser; i++)
00339     {
00340         summary->accepted[i] = user[i].accepted->count;
00341         summary->rejected[i] = user[i].rejected->count;
00342         summary->empty_timeslot[i] = GetEmptyTimeSlotInRange(user[i].accepted, summary->start, summary->end
00343     )->count;
00344
00345         // PrintAppointmentList(GetEmptyTimeSlotInRange(user[i].accepted, summary->start, summary->end));
00346     }
00347     return summary;
00348 }
00349
00350 struct Summary* Schedual_OPTI(struct AppointmentList *inputList)
00351 {
00352     struct Summary *summary = Schedual_PRIO(inputList);
00353     time_t day = 60*60*24;
00354
00355     //For each user, try to reschedule their rejected jobs
00356     for(int i=0; i<NumOfUser; i++)
00357     {
00358         struct Appointment *item = user[i].rejected->head;
00359         NEXT_ITEM:
00360         while(item)
00361         {
00362             time_t ori_start = item->start;
00363             time_t ori_end = item->end;
00364             time_t duration = difftime(item->end, item->start);

```

```

00361         struct AppointmentList *list = GetEmptyTimeSlotInRange(user[i].accepted, item->start, item->
start+3*day);
00362         struct AppointmentList *c_list = GetContinueTimeslotFromList(list, duration);
00363
00364         struct Appointment *timeslot = c_list->head;
00365         while(timeslot)
00366         {
00367             item->start = timeslot->start;
00368             item->end = item->start + duration;
00369
00370             int ret = AllAvailable(item);
00371             if(ret<0)
00372             {
00373                 item->rescheduled = 1;
00374                 AddToAllAccept(item);
00375                 struct Appointment *temp = item;
00376                 item = item->next;
00377                 for(int j=0; j<NumOfUser; j++)
00378                     RemoveItemFromList(user[j].rejected, temp);
00379                 goto NEXT_ITEM;
00380             }
00381             strcpy(item->reason, "No available timeslot for the reschedule.");
00382             item->start = ori_start;
00383             item->end = ori_end;
00384             timeslot = timeslot->next;
00385         }
00386         item = item->next;
00387     }
00388 }
00389
00390 //Re-calculate the summary
00391 memset(summary, 0, sizeof(struct Summary));
00392 summary->start = GetEarliestStartTime(inputList);
00393 summary->end = GetLatestEndTime(inputList);
00394 SetAppointmentAccepted(inputList);
00395 struct Appointment *ptr = inputList->head;
00396 while(ptr)
00397 {
00398     if(ptr->is_accepted)
00399         summary->total_accepted++;
00400     else
00401         summary->total_rejected++;
00402     ptr = ptr->next;
00403 }
00404 for(int i=0; i<NumOfUser; i++)
00405 {
00406     summary->accepted[i] = user[i].accepted->count;
00407     summary->rejected[i] = user[i].rejected->count;
00408     summary->empty_timeslot[i] = GetEmptyTimeSlotInRange(user[i].accepted, summary->start, summary->end
)->count;
00409     // PrintAppointmentList(GetEmptyTimeSlotInRange(user[i].accepted, summary->start, summary->end));
00410 }
00411 return summary;
00412 }
00413
00414 static int rdn(int y, int m, int d) {
00415     if (m < 3)
00416         y--, m += 12;
00417     return 365*y + y/4 - y/100 + y/400 + (153*m - 457)/5 + d - 306;
00418 }
00419
00420 void PrintSummary(struct Summary *summary)
00421 {
00422     float total = summary->total_accepted+summary->total_rejected;
00423     struct tm timeinfo, timeinfo2;
00424     int days;
00425
00426     printf("Performance:\n");
00427     timeinfo = *localtime(&summary->start);
00428     printf("Date start: %4d-%02d-%02d\n", timeinfo.tm_year+1900, timeinfo.tm_mon+1, timeinfo.tm_mday);
00429     timeinfo2 = *localtime(&summary->end);
00430     printf("Date end: %4d-%02d-%02d\n\n", timeinfo2.tm_year+1900, timeinfo2.tm_mon+1, timeinfo2.tm_mday);
00431
00432     printf("Total Number of Appointment Assigned: %d (%.1f%%)\n", summary->total_accepted, summary->
total_accepted/total*100);
00433     printf("Total Number of Appointment Rejected: %d (%.1f%%)\n", summary->total_rejected, summary->
total_rejected/total*100);
00434
00435     days = rdn(timeinfo2.tm_year, timeinfo2.tm_mon+1, timeinfo2.tm_mday) - rdn(timeinfo.tm_year, timeinfo.
tm_mon+1, timeinfo.tm_mday) + 1;
00436     printf("Utilization of Time Slot: (%d days)\n", days+1);
00437     for(int i=0; i<NumOfUser; i++)
00438     {
00439         printf("    %-10s - %.1f%%\n", user[i].username, (days*2*4-(float)summary->empty_timeslot[i])/(days
*2*4)*100); //each day have 2*8 timeslots
00440     }
00441 }

```

```
00442 }
```

5.9 scheduler.h File Reference

Scheduling algorithms.

```
#include "user.h"
#include <time.h>
```

Data Structures

- struct **Summary**

Functions

- struct **Summary** * **Schedual_FCFS** (struct **AppointmentList** *inputList)
First come first served. The order is following the input order. The result will be putted into the each user's appointment lists (accept / reject).
- struct **Summary** * **Schedual_PRIO** (struct **AppointmentList** *inputList)
Priority. The order is following the pre-defined priority. The result will be putted into the each user's appointment lists (accept / reject).
- struct **Summary** * **Schedual_OPTI** (struct **AppointmentList** *inputList)
Optimized. Bonus part, reschedule those rejected appointments. The result will be putted into the each user's appointment lists (accept / reject).
- void **PrintSummary** (struct **Summary** *summary)
Print out the summary about the scheduling.

5.9.1 Detailed Description

Scheduling algorithms.

Author

oneonestar oneonestar@gmail.com

Version

1.0

Copyright

2015

5.9.2 LICENSE

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Definition in file **scheduler.h**.

5.10 scheduler.h

```

00001
00022 #ifndef SCHEDULER
00023 #define SCHEDULER
00024
00025 #include "user.h"
00026 #include <time.h>
00027
00028 struct Summary
00029 {
00030     int total_accepted;
00031     int total_rejected;
00032     int accepted[USER_NUMBER];
00033     int rejected[USER_NUMBER];
00034     int empty_timeslot[USER_NUMBER];
00035     time_t start;
00036     time_t end;
00037 };
00042 struct Summary* Scheduling_FCFS(struct AppointmentList *inputList);
00043
00048 struct Summary* Scheduling_PRIO(struct AppointmentList *inputList);
00049
00054 struct Summary* Scheduling_OPTI(struct AppointmentList *inputList);
00055
00059 void PrintSummary(struct Summary *summary);
00060
00061 #endif

```

5.11 user.c File Reference

Handling each users.

```

#include <string.h>
#include <ctype.h>
#include <stdio.h>
#include "user.h"
#include "appointment_list.h"

```

Functions

- int **strcmp** (char const *a, char const *b)
- int **GetUserID** (const char *username)
Return the user id that have the same username.
- void **PrintAccepted** (const struct **User** *user)
Print the accepted list for user.
- void **PrintRejected** (const struct **User** *user)
Print the rejected list for user.
- void **PrintAllUser** ()
Print the accepted & reject list for all users.

5.11.1 Detailed Description

Handling each users.

Author

oneonestar oneonestar@gmail.com

Version

1.0

Copyright

2015

5.11.2 LICENSE

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Definition in file **user.c**.

5.11.3 Function Documentation**5.11.3.1 int GetUserID (const char * username)**

Return the user id that have the same username.

Parameters

<i>username</i>	The username that are used to compare.
-----------------	--

Returns

The user id that have the same username. Return -1 if user is not found.

Definition at line **38** of file **user.c**.

5.12 user.c

```

00001
00022 #include <string.h>
00023 #include <ctype.h>
00024 #include <stdio.h>
00025
00026 #include "user.h"
00027 #include "appointment_list.h"
00028
00029 int strcmp(char const *a, char const *b)
00030 {
00031     for (; a++, b++) {
00032         int d = tolower(*a) - tolower(*b);
00033         if (d != 0 || !*a)
00034             return d;
00035     }
00036 }
00037
00038 int GetUserID(const char *username)
00039 {
00040     for(int i=0; i<NumOfUser; i++)
00041         if (!strcmp(user[i].username, username))
00042             return i;
00043     return -1;
00044 }
00045
00046
00047
00048 void PrintAccepted(const struct User *user)
00049 {

```

```

00050     printf("%s, you have %d appointments.\n", user->username, user->accepted->count);
00051     PrintAppointmentList (user->accepted);
00052 }
00053
00054
00055
00056 void PrintRejected(const struct User *user)
00057 {
00058     printf("%s, you have %d appointments rejected.\n", user->username, user->rejected->count);
00059     PrintAppointmentList (user->rejected);
00060 }
00061
00062
00063 void PrintAllUser()
00064 {
00065     printf("***Appointment Schedule - ACCEPTED ***\n\n");
00066     for(int i=0; i<NumOfUser; i++)
00067     {
00068         PrintAccepted(&user[i]);
00069     }
00070     printf("    -End-\n");
00071     printf("=====\\n");
00072     printf("***Appointment Schedule - REJECTED ***\n\n");
00073     for(int i=0; i<NumOfUser; i++)
00074     { 00075     PrintRejected(&user[i]);
00076     }
00077     printf("    -End-\n");
00078     printf("=====\\n");
00079 }

```

5.13 user.h File Reference

Handling each users.

Data Structures

- struct **User**
Store the basic information of the user and the appointments.

Macros

- #define **USER_NUMBER** 10
- #define **MAX_USERNAME** 31

Functions

- int **GetUserID** (const char *username)
Return the user id that have the same username.
- void **PrintAccepted** (const struct **User** *user)
Print the accepted list for user.
- void **PrintRejected** (const struct **User** *user)
Print the rejected list for user.
- void **PrintAllUser** ()
Print the accepted & reject list for all users.

Variables

- struct **User** user [**USER_NUMBER**]
- int **NumOfUser**

5.13.1 Detailed Description

Handling each users.

Author

oneonestar oneonestar@gmail.com

Version

1.0

Copyright

2015

5.13.2 LICENSE

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Definition in file **user.h**.

5.13.3 Macro Definition Documentation

5.13.3.1 #define MAX_USERNAME 31

Maximum length of the username, 30 for the name and 1 for the null.

Definition at line **34** of file **user.h**.

5.13.3.2 #define USER_NUMBER 10

The number of users is from 3 to 10, so maximum is 10.

Definition at line **28** of file **user.h**.

5.13.4 Function Documentation

5.13.4.1 int GetUserID (const char * username)

Return the user id that have the same username.

Parameters

<i>username</i>	The username that are used to compare.
-----------------	--

Returns

The user id that have the same username. Return -1 if user is not found.

Definition at line **38** of file **user.c**.

5.13.5 Variable Documentation

5.13.5.1 int NumOfUser

Global variable storing current number of user.

Definition at line 55 of file **user.h**.

5.13.5.2 struct User user[USER_NUMBER]

Global variable storing the user data.

Definition at line 50 of file **user.h**.

5.14 user.h

```
00001
00022 #ifndef USER
00023 #define USER
00024
00028 #define USER_NUMBER 10
00029
00034 #define MAX_USERNAME 31
00035
00040 struct User
00041 {
00042     char username[MAX_USERNAME];
00043     struct AppointmentList *accepted;
00044     struct AppointmentList *rejected;
00045 };
00046
00050 struct User user[USER_NUMBER];
00051
00055 int NumOfUser;
00056
00062 int GetUserID(const char *username);
00063
00067 void PrintAccepted(const struct User *user);
00068
00072 void PrintRejected(const struct User *user);
00073
00077 void PrintAllUser();
00078
00079 #endif
```

Index

AddAppointment
 appointment_list.c, 12
 appointment_list.h, 20
AddAppointmentFromList
 appointment_list.c, 13
 appointment_list.h, 20
AddAppointmentOrdered
 appointment_list.c, 13
 appointment_list.h, 20
AddAppointmentOrderedFromList
 appointment_list.c, 13
 appointment_list.h, 20
Appointment, 7
appointment_list.c, 11, 15
 AddAppointment, 12
 AddAppointmentFromList, 13
 AddAppointmentOrdered, 13
 AddAppointmentOrderedFromList, 13
 AppointmentTypeStr, 14
 CompareAppointment, 13
 CompareAppointmentPriority, 14
 ConflictInList, 14
 PrintAppointment, 14
 PrintAppointmentList, 14
appointment_list.h, 18, 22
 AddAppointment, 20
 AddAppointmentFromList, 20
 AddAppointmentOrdered, 20
 AddAppointmentOrderedFromList, 20
 CompareAppointment, 21
 CompareAppointmentPriority, 21
 ConflictInList, 21
 PrintAppointment, 21
 PrintAppointmentList, 22
AppointmentList, 7
AppointmentType, 8
AppointmentTypeStr
 appointment_list.c, 14

CompareAppointment
 appointment_list.c, 13
 appointment_list.h, 21
CompareAppointmentPriority
 appointment_list.c, 14
 appointment_list.h, 21
ConflictInList
 appointment_list.c, 14
 appointment_list.h, 21

GetUserID
 user.c, 36
 user.h, 38

MAX_USERNAME
 user.h, 38
main.c, 23, 24
 NumOfUser, 24
 user, 24

NumOfUser
 main.c, 24
 user.h, 39

PrintAppointment
 appointment_list.c, 14
 appointment_list.h, 21
PrintAppointmentList
 appointment_list.c, 14
 appointment_list.h, 22

scheduler.c, 28, 29
scheduler.h, 34, 35
Summary, 8

USER_NUMBER
 user.h, 38
User, 9
user
 main.c, 24
 user.h, 39
user.c, 35, 36
 GetUserID, 36
user.h, 37, 39
 GetUserID, 38
 MAX_USERNAME, 38
 NumOfUser, 39
 USER_NUMBER, 38
 user, 39