Appointment Manager (AMR)

Generated by Doxygen 1.8.8

Sat Apr 4 2015 03:30:28

Contents

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

doc/latex/dox.pdf

File structure

-bin/ -src/ .... source and header files

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

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**
- struct **Appointment** * **prev**
- struct **Appointment** * **next**

4.1.1 Detailed Description

Store a appointment record.

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

4.1.2 Field Documentation

4.1.2.1 int callee_id[10]

Definition at line **53** of file **appointment_list.h**.

4.1.2.2 int caller_id

Definition at line **52** of file **appointment_list.h**.

4.1.2.3 time_t end

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

4.1.2.4 int id

Definition at line **51** of file **appointment_list.h**.

4.1.2.5 int is_accepted

Definition at line **56** of file **appointment_list.h**.

4.1.2.6 struct Appointment* next

Definition at line **59** of file **appointment_list.h**.

4.1.2.7 struct Appointment* prev

Definition at line **58** of file **appointment_list.h**.

4.1.2.8 int rescheduled

Definition at line **57** of file **appointment_list.h**.

4.1.2.9 time_t start

Definition at line **54** of file **appointment_list.h**.

4.1.2.10 enum AppointmentType type

Definition at line **50** 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 **66** of file **appointment_list.h**.

4.2.2 Field Documentation

4.2.2.1 int count

Definition at line **68** of file **appointment_list.h**.

4.2.2.2 struct Appointment* head

Definition at line **69** of file **appointment_list.h**.

4.2.2.3 struct Appointment* tail

Definition at line **70** 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

`#include <scheduler.h>`

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**.

4.4.2 Field Documentation

4.4.2.1 int accepted[USER_NUMBER]

Definition at line **32** of file **scheduler.h**.

4.4.2.2 int empty_timeslot[USER_NUMBER]

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

4.4.2.3 time_t end

Definition at line **36** of file **scheduler.h**.

4.4.2.4 int rejected[USER_NUMBER]

Definition at line **33** of file **scheduler.h**.

4.4.2.5 time_t start

Definition at line **35** of file **scheduler.h**.

4.4.2.6 int total_accepted

Definition at line **30** of file **scheduler.h**.

4.4.2.7 int total_rejected

Definition at line **31** 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**.

4.5.2 Field Documentation

4.5.2.1 struct AppointmentList* accepted

Definition at line **43** of file **user.h**.

4.5.2.2 struct AppointmentList* rejected

Definition at line **44** of file **user.h**.

4.5.2.3 char username[MAX_USERNAME]

Definition at line **42** 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.

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`	*list*	The destination appointment list.

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

Definition at line **64** 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	*list*	The destination appointment list.
in	*newItem*	The item that needs to add into the list.

Definition at line **124** 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	*list*	The destination appointment list.
in	*newItem*	The item that needs to add into the list.

Definition at line **83** 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	*list*	The destination appointment list.
in	*newItem*	The item that needs to add into the list.

Definition at line **134** 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	*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 **144** 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 **154** 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 **246** of file **appointment_list.c**.

5.1.3.8 struct Appointment* CreateAppointment ( )

Create a appointment and init the value.

Definition at line **46** of file **appointment_list.c**.

5.1.3.9 struct AppointmentList* CreateAppointmentList ( )

Create a appointment list and init the value.

Definition at line **32** of file **appointment_list.c**.

5.1.3.10 struct Appointment* GetAppointmentById (const struct AppointmentList * *list,* int *id*)

Return the appointment that match the id in the list.

Definition at line **261** of file **appointment_list.c**.

5.1.3.11 int IsConflict (const struct Appointment * *a,* const struct Appointment * *b*)

Check whether if two appointments have time conflict.

Definition at line **226** of file **appointment_list.c**.

5.1.3.12 int IsConflictInList (const struct AppointmentList * *list,* const struct Appointment * *item*)

Check whether if the appointment item have conflict with the list.

Definition at line **232** of file **appointment_list.c**.

5.1.3.13 void PrintAppointment (const struct Appointment * *item*)

Print out the appointment.

Parameters

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

Definition at line **191** of file **appointment_list.c**.

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

Print out the appointment list.

Parameters

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

Definition at line **216** of file **appointment_list.c**.

5.1.3.15 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.

Definition at line **159** of file **appointment_list.c**.

5.1.3.16 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.

Definition at line **181** 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     return item;
00062 }
00063
00064 void AddAppointment(struct AppointmentList *list, const struct Appointment *newItem)
00065 {
00066     struct Appointment *item = CreateAppointment();
00067     *item = *newItem;
00068     item->next = item->prev = 0;
00069     if(!list->head) //if the list is empty
00070     {
00071         list->head = item;
00072         list->tail = item;
00073     }
00074     else
00075     {
00076         list->tail->next = item;
00077     }
00078     item->prev = list->tail;
00079     list->tail = item;
00080     list->count++;
00081 }
00082
00083 void AddAppointmentOrdered(struct AppointmentList *list, const struct
      Appointment *newItem)
00084 {
00085     struct Appointment *item = CreateAppointment();
00086     *item = *newItem;
00087     item->next = item->prev = 0;
00088     struct Appointment *ptr = list->head;
00089     //if the list is empty
00090     if(!ptr)
00091     {
00092         list->head = item;
00093         list->tail = item;
00094     }
00095     else if(CompareAppointment(item, ptr)<0)    //if insert at the head
00096     {
00097         if(!list->head)
00098             list->head->prev = item;
00099         item->next = list->head;
00100         list->head = item;
00101     }
00102     else    //insert at middle or at the tail
00103     {
00104         while(ptr->next)    //find the insertion position
00105         {
00106             if(difftime(item->start, ptr->next->start)<0)
00107                 break;
00108             ptr = ptr->next;
00109         }
00110         if(!ptr)    //insert at the tail
00111         {
00112             ptr = list->tail;
00113             list->tail = item;
00114         }
00115         item->prev = ptr;   //insert after ptr
00116         item->next = ptr->next;
00117         if(item->next)
00118             item->next->prev = item;
00119         ptr->next = item;
00120     }
00121     list->count++;
00122 }
00123
00124 void AddAppointmentFromList(struct AppointmentList *dst_list, const struct
      AppointmentList *src_list)
00125 {
```

```
00126     struct Appointment *newItem = src_list->head;
00127     while(newItem)
00128     {
00129         AddAppointment(dst_list, newItem);
00130         newItem = newItem->next;
00131     }
00132 }
00133
00134 void AddAppointmentOrderedFromList(struct AppointmentList *dst_list, const struct
      AppointmentList *src_list)
00135 {
00136     struct Appointment *newItem = src_list->head;
00137     while(newItem)
00138     {
00139         AddAppointmentOrdered(dst_list, newItem);
00140         newItem = newItem->next;
00141     }
00142 }
00143
00144 int CompareAppointment(const struct Appointment *a, const struct Appointment *b)
00145 {
00146     if(difftime(a->start, b->start)<0)
00147         return -1;  //a before b
00148     else if(difftime(a->start, b->start)==0)
00149         return difftime(a->end, b->end);
00150     else
00151         return 1;
00152 }
00153
00154 int CompareAppointmentPriority(const struct Appointment *a, const struct
      Appointment *b)
00155 {
00156     return a->type - b->type;
00157 }
00158
00159 void RemoveItemFromList(struct AppointmentList *list, const struct Appointment *item)
00160 {
00161     struct Appointment *delItem = list->head;
00162     while(delItem)
00163     {
00164         if(delItem->id == item->id)
00165         {
00166             if(!delItem->prev)  //if prev is null, first item in list
00167                 list->head = delItem->next;
00168             else
00169                 delItem->prev->next = delItem->next;
00170             if(!delItem->next)  //if next is null, last item in list
00171                 list->tail = delItem->prev;
00172             else
00173                 delItem->next->prev = delItem->prev;
00174             list->count--;
00175             return;
00176         }
00177         delItem = delItem->next;
00178     }
00179 }
00180
00181 void RemoveListFromList(struct AppointmentList *ori_list, const struct
      AppointmentList *del_list)
00182 {
00183     struct Appointment *delItem = del_list->head;
00184     while(delItem)
00185     {
00186         RemoveItemFromList(ori_list, delItem);
00187         delItem = delItem->next;
00188     }
00189 }
00190
00191 void PrintAppointment(const struct Appointment *item)
00192 {
00193     struct tm tm_start, tm_end;
00194     memcpy(&tm_start, localtime (&item->start), sizeof(struct tm));
00195     memcpy(&tm_end, localtime (&item->end), sizeof(struct tm));
00196     printf("%d ", item->id);
00197     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,
00198      tm_start.tm_min, tm_end.tm_hour, tm_end.tm_min, AppointmentTypeStr[item->
      type]);
00199     if(item->rescheduled)
00200         printf("%-9c ",  'Y');
00201     else
00202         printf("%-9c ", 'N');
00203     if(item->callee_id[0] == -1)
00204         printf("-");
00205     else
00206         printf("%s ", user[item->caller_id].username);
00207     for(int i=0; i<USER_NUMBER; i++)
```

```
00208     {
00209         if(item->callee_id[i]==-1)
00210             break;
00211         printf("%s ", user[item->callee_id[i]].username);
00212     }
00213     printf("\n");
00214 }
00215
00216 void PrintAppointmentList(const struct AppointmentList *list)
00217 {
00218     struct Appointment *ptr = list->head;
00219     while(ptr!=NULL)
00220     {
00221         PrintAppointment(ptr);
00222         ptr = ptr->next;
00223     }
00224 }
00225
00226 int IsConflict(const struct Appointment *a, const struct Appointment *b)
00227 {
00228     return !(difftime(a->end, b->start)<=0 ||   //a before b
00229         difftime(a->start, b->end)>=0);     //a after b
00230 }
00231
00232 int IsConflictInList(const struct AppointmentList *list, const struct
      Appointment *item)
00233 {
00234     if(!list || !item)
00235         return 0;
00236     struct Appointment *ptr = list->head;
00237     while(ptr)
00238     {
00239         if(IsConflict(ptr, item))
00240             return 1;
00241         ptr = ptr->next;
00242     }
00243     return 0;
00244 }
00245
00246 struct AppointmentList* ConflictInList(const struct AppointmentList *list, const struct
      Appointment *item)
00247 {
00248     if(!list || !item)
00249         return NULL;
00250     struct AppointmentList *conflict_list = CreateAppointmentList();
00251     struct Appointment *ptr = list->head;
00252     while(ptr)
00253     {
00254         if(IsConflict(ptr, item))
00255             AddAppointment(conflict_list, ptr);
00256         ptr = ptr->next;
00257     }
00258     return conflict_list;
00259 }
00260
00261 struct Appointment* GetAppointmentById(const struct AppointmentList *list, int id)
00262 {
00263     if(!list)
00264         return NULL;
00265     struct Appointment *ptr = list->head;
00266     while(ptr)
00267     {
00268         if(ptr->id == id)
00269             return ptr;
00270         ptr = ptr->next;
00271     }
00272     return NULL;
00273 }
```

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** * **GetAppointmentById** (const struct **AppointmentList** *list, int id)

  Return the appointment that match the id in the list.

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 Enumeration Type Documentation

5.3.3.1 enum AppointmentType

Enumerator

ASSIGNMENT

PROJECT

STUDY

GATHERING

Definition at line **39** of file **appointment_list.h**.

5.3.4 Function Documentation

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

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

Parameters

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

Definition at line **64** of file **appointment_list.c**.

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

Definition at line **124** of file **appointment_list.c**.

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

Definition at line **83** of file **appointment_list.c**.

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

Definition at line **134** of file **appointment_list.c**.

5.3.4.5 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	*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 **144** of file **appointment_list.c**.

5.3.4.6 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 **154** of file **appointment_list.c**.

5.3.4.7 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 **246** of file **appointment_list.c**.

5.3.4.8 struct Appointment* CreateAppointment ( )

Create a appointment and init the value.

Definition at line **46** of file **appointment_list.c**.

5.3.4.9 struct AppointmentList* CreateAppointmentList ( )

Create a appointment list and init the value.

Definition at line **32** of file **appointment_list.c**.

5.3.4.10 struct Appointment* GetAppointmentById (const struct AppointmentList * *list,* int *id*)

Return the appointment that match the id in the list.

Definition at line **261** of file **appointment_list.c**.

5.3.4.11 int IsConflict (const struct Appointment * *a,* const struct Appointment * *b*)

Check whether if two appointments have time conflict.

Definition at line **226** of file **appointment_list.c**.

5.3.4.12 int IsConflictInList (const struct AppointmentList * *list*, const struct Appointment * *item*)

Check whether if the appointment item have conflict with the list.

Definition at line **232** of file **appointment_list.c**.

5.3.4.13 void PrintAppointment (const struct Appointment * *item*)

Print out the appointment.

Parameters

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

Definition at line **191** of file **appointment_list.c**.

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

Print out the appointment list.

Parameters

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

Definition at line **216** of file **appointment_list.c**.

5.3.4.15 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.

Definition at line **159** of file **appointment_list.c**.

5.3.4.16 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.

Definition at line **181** 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     struct Appointment *prev;
00059     struct Appointment *next;
00060 };
00061
00066 struct AppointmentList
00067 {
00068     int count;
00069     struct Appointment *head;
00070     struct Appointment *tail;
00071 };
00072
00073
00077 struct Appointment* CreateAppointment();
00078
00082 struct AppointmentList* CreateAppointmentList();
00083
00084
00090 void AddAppointment(struct AppointmentList *list, const struct Appointment *newItem);
00091
00098 void AddAppointmentOrdered(struct AppointmentList *list, const struct
      Appointment *newItem);
00099
00105 void AddAppointmentFromList(struct AppointmentList *dst_list, const struct
      AppointmentList *src_list);
00106
00113 void AddAppointmentOrderedFromList(struct AppointmentList *dst_list, const struct
      AppointmentList *src_list);
00114
00120 struct AppointmentList* ConflictInList(const struct AppointmentList *list, const struct
      Appointment *item);
00121
00125 int IsConflict(const struct Appointment *a, const struct Appointment *b);
00126
00130 int IsConflictInList(const struct AppointmentList *list, const struct
      Appointment *item);
00131
00136 void RemoveItemFromList(struct AppointmentList *list, const struct Appointment *item);
00137
00142 void RemoveListFromList(struct AppointmentList *ori_list, const struct
      AppointmentList *del_list);
00143
00153 int CompareAppointment(const struct Appointment *a, const struct Appointment *b);
00154
00163 int CompareAppointmentPriority(const struct Appointment *a, const struct
      Appointment *b);
00164
00169 void PrintAppointment(const struct Appointment *item);
00170
00175 void PrintAppointmentList(const struct AppointmentList *list);
00176
00180 struct Appointment* GetAppointmentById(const struct AppointmentList *list, int id);
00181
00182 #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 Function Documentation

5.5.3.1 void HandleInput (const char ∗ *line*)

Definition at line **122** of file **main.c**.

5.5.3.2 void HandleSchedule (const char ∗ *algorithm*)

Definition at line **47** of file **main.c**.

5.5.3.3 void inputLoop (FILE ∗ *stream*)

Definition at line **244** of file **main.c**.

5.5.3.4 int main (int *argc*, char * *argv[]*)

Definition at line **271** of file **main.c**.

5.5.4 Variable Documentation

5.5.4.1 struct AppointmentList* inputList

Definition at line **41** of file **main.c**.

5.5.4.2 int NumOfUser

Global variable storing current number of user.

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

5.5.4.3 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 #ifdef 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 = Schedual_FCFS(inputList);
00065     else if(!strcmp(algorithm, "-prio"))
00066         summary = Schedual_PRIO(inputList);
00067     else if(!strcmp(algorithm, "-opti"))
00068         summary = Schedual_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 = Schedual_FCFS(inputList);
00099         else if(!strcmp(algorithm, "-prio"))
00100             summary = Schedual_PRIO(inputList);
00101         else if(!strcmp(algorithm, "-opti"))
00102             summary = Schedual_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", 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         item->callee_id[callee_count++] = id;
00202     }
00203
00204     item->caller_id = GetUserID(caller);
00205     //time
00206     struct tm timeinfo, timeinfo_tmp;
00207     memset(&timeinfo, 0, sizeof(timeinfo));
00208     timeinfo.tm_isdst = -1;
00209     timeinfo.tm_year = year - 1900;
00210     timeinfo.tm_mon = month - 1;
00211     timeinfo.tm_mday = day;
00212     //start time
00213     //convert ot half hour base
00214     timeinfo.tm_hour = hour;
00215     if(minutes>=0 && minutes <= 30)
00216         timeinfo.tm_min = 0;
00217     else
00218         timeinfo.tm_min = 30;
00219
00220     timeinfo_tmp = timeinfo;    //because mktime could modify the value
00221     item->start = mktime(&timeinfo_tmp);
00222     //convert duration to end time
00223     double _;
00224     double fractional = modf(duration, &_);
00225     minutes += fractional*60;
00226     hour = hour+(int)duration;
00227     if(minutes>=60)
00228         hour++;
00229     minutes %= 60;
00230
00231     timeinfo.tm_hour = hour;
00232     if(minutes>0 && minutes <= 30)
00233         timeinfo.tm_min = 0;
00234     else
00235         timeinfo.tm_min = 30;
00236     item->end = mktime(&timeinfo);
00237
00238     item->id = inputList->count;
00239     AddAppointment(inputList, item);
00240     printf("-> [Pending]\n");
00241 }
00242
```

```
00243
00244 void inputLoop(FILE *stream)
00245 {
00246     const int MAX_CHAR = 255;
00247     char line[MAX_CHAR];
00248     char *return_val;
00249     while(1)
00250     {
00251         printf("Please enter appointment:\n");
00252         return_val = fgets(line, MAX_CHAR, stream);
00253         if(!return_val)
00254         {
00255             if(feof(stream))
00256             {
00257                 printf("Received EOF.\n");
00258                 return;
00259             }
00260             else
00261             {
00262                 fprintf(stderr, "IO error, existing program.\n");
00263                 return;
00264                 // exit(EXIT_FAILURE);
00265             }
00266         }
00267         HandleInput(line);
00268     }
00269 }
00270
00271 int main(int argc, char* argv[])
00272 {
00273     if (argc < 4 || argc > 11)
00274     {
00275         fprintf(stderr, "Error: The number of users should between 3 and 10.\n");
00276         return EXIT_FAILURE;
00277     }
00278     NumOfUser = argc - 1;
00279     //Initialize each user in struct user[];
00280     for (int i=0; i<NumOfUser; i++)
00281     {
00282         if(GetUserID(argv[i+1]) != -1)
00283         {
00284             printf("Duplicate names of users!\n");
00285             exit(EXIT_FAILURE);
00286         }
00287         strcpy(user[i].username, argv[i+1]);
00288         user[i].username[0] = toupper(user[i].username[0]);
00289         user[i].accepted = CreateAppointmentList();
00290         user[i].rejected = CreateAppointmentList();
00291     }
00292     inputList = CreateAppointmentList();
00293
00294     printf("~~WELCOME TO AMR~~\n");
00295     inputLoop(stdin);
00296     return EXIT_SUCCESS;
00297 }
```

5.7 README.md File Reference

5.8 README.md

```
00001 Appointment Manager (AMR)
00002 ========================
00003 An appointment management software that have the calendar and scheduling function.
00004
00005 Build
00006 -----
00007 To compile the program.
00008 ```
00009 make
00010 ```
00011 The executable program is located in bin/.
00012
00013 To clean up the object files.
00014 ```
00015 make clean
00016 ```
00017
00018 To clean up the object files and the executable file.
00019 ```
00020 make remove
00021 ```
```

```
00022
00023 To join the source files into one AMR.c file
00024 ```
00025 make onefile
00026 ```
00027
00028 Documentation
00029 -------------
00030 doc/latex/dox.pdf
00031
00032 File structure
00033 --------------
00034 -bin/
00035 -src/ .... source and header files
```

5.9 scheduler.c File Reference

Secheduling algorithms.

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

Functions

- int **IsAllAvailable** (const struct **Appointment** *item)
- int **IsAllAvailablePriority** (const struct **Appointment** *item)
- void **AddToAllAccept** (const struct **Appointment** *item)
- void **AddToAllAcceptForced** (const struct **Appointment** *item)
- void **AddToAllReject** (const struct **Appointment** *item)
- struct **Summary** * **Schedual_FCFS** (struct **AppointmentList** ***inputList**)

  First come first served. The order is folloing 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 folloing 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

Secheduling 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.c**.

5.9.3 Function Documentation

5.9.3.1 void AddToAllAccept (const struct Appointment * *item*)

Definition at line **80** of file **scheduler.c**.

5.9.3.2 void AddToAllAcceptForced (const struct Appointment * *item*)

Definition at line **91** of file **scheduler.c**.

5.9.3.3 void AddToAllReject (const struct Appointment * *item*)

Definition at line **114** of file **scheduler.c**.

5.9.3.4 int IsAllAvailable (const struct Appointment * *item*)

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

5.9.3.5 int IsAllAvailablePriority (const struct Appointment * *item*)

Definition at line **49** of file **scheduler.c**.

5.9.3.6 void PrintSummary (struct Summary * *summary*)

Print out the summary about the scheduling.

Definition at line **404** of file **scheduler.c**.

5.9.3.7 struct Summary* Schedual_FCFS (struct AppointmentList * *inputList*)

First come first served. The order is folloing the input order. The result will be putted into the each user's appointment lists (accept / reject).

Definition at line **248** of file **scheduler.c**.

5.9.3.8 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).

Definition at line **332** of file **scheduler.c**.

5.9.3.9 struct Summary* Schedual_PRIO (struct AppointmentList * *inputList*)

Priority. The order is folloing the pre-defined priority. The result will be putted into the each user's appointment lists (accept / reject).

Definition at line **291** of file **scheduler.c**.

5.10 scheduler.c

```
00001
00021 #include "appointment_list.h"
00022 #include "user.h"
00023
00024 #include <unistd.h>
00025
00026 #include "scheduler.h"
00027
00028 int IsAllAvailable(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 0;
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 0;
00045     }
00046     return 1;
00047 }
00048
00049 int IsAllAvailablePriority(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 0;   //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 0;   //not available
00074             ptr = ptr->next;
00075         }
00076     }
00077     return 1;   //available
00078 }
00079
00080 void AddToAllAccept(const struct Appointment *item)
00081 {
00082     AddAppointmentOrdered(user[item->caller_id].accepted, item);
00083     for(int i=0; i<USER_NUMBER; i++)
00084     {
00085         if(item->callee_id[i]==-1)
00086             break;
00087         AddAppointmentOrdered(user[item->callee_id[i]].accepted, item);
00088     }
00089 }
00090
00091 void AddToAllAcceptForced(const struct Appointment *item)
```

```
00092 {
00093     struct AppointmentList *temp_list;
00094     //caller
00095     //delete old appointments from accepted list
00096     temp_list = ConflictInList(user[item->caller_id].accepted, item);
00097     RemoveListFromList(user[item->caller_id].accepted, temp_list);
00098     //add to accept and reject lsit
00099     AddAppointmentOrdered(user[item->caller_id].accepted, item);
00100     AddAppointmentOrderedFromList(user[item->caller_id].rejected, temp_list);
00101     for(int i=0; i<USER_NUMBER; i++)
00102     {
00103         if(item->callee_id[i]==-1)
00104             break;
00105         //delete old appointments from accepted list
00106         temp_list = ConflictInList(user[item->callee_id[i]].accepted, item);
00107         RemoveListFromList(user[item->callee_id[i]].accepted, temp_list);
00108         //add to accept and reject lsit
00109         AddAppointmentOrdered(user[item->callee_id[i]].accepted, item);
00110         AddAppointmentOrderedFromList(user[item->callee_id[i]].rejected, temp_list);
00111     }
00112 }
00113
00114 void AddToAllReject(const struct Appointment *item)
00115 {
00116     AddAppointmentOrdered(user[item->caller_id].rejected, item);
00117     for(int i=0; i<USER_NUMBER; i++)
00118     {
00119         if(item->callee_id[i]==-1)
00120             break;
00121         AddAppointmentOrdered(user[item->callee_id[i]].rejected, item);
00122     }
00123 }
00124
00126 static void SetAppointmentAccepted(struct AppointmentList *inputList)
00127 {
00128     for(int i=0; i<NumOfUser; i++)
00129     {
00130         struct Appointment *ptr = user[i].accepted->head;
00131         while(ptr)
00132         {
00133             GetAppointmentById(inputList, ptr->id)->is_accepted = 1;
00134             ptr = ptr->next;
00135         }
00136     }
00137 }
00138
00139 static struct AppointmentList* GetEmptyTimeSlotInDay(struct AppointmentList *userList, time_t date)
00140 {
00141     struct AppointmentList *empty_list = CreateAppointmentList();
00142     //18:00-22:00
00143     struct tm timeinfo, timeinfo_tmp;
00144     timeinfo = *localtime (&date);
00145
00146     //foreach 18:00-22:00 half hour timeslot
00147     timeinfo.tm_hour = 18;
00148     while(timeinfo.tm_hour<22)
00149     {
00150         struct Appointment *item = CreateAppointment();
00151         //first half hour
00152         timeinfo_tmp = timeinfo;    //because mktime could modify the value
00153         item->start = mktime(&timeinfo_tmp);
00154         timeinfo.tm_min = 30;
00155         timeinfo_tmp = timeinfo;    //because mktime could modify the value
00156         item->end = mktime(&timeinfo_tmp);
00157         if(!IsConflictInList(userList, item))
00158             AddAppointmentOrdered(empty_list, item);
00159
00160
00161         //second half hour
00162         timeinfo_tmp = timeinfo;    //because mktime could modify the value
00163         item->start = mktime(&timeinfo_tmp);
00164         timeinfo.tm_hour++;
00165         timeinfo.tm_min = 0;
00166         timeinfo_tmp = timeinfo;    //because mktime could modify the value
00167         item->end = mktime(&timeinfo_tmp);
00168         if(!IsConflictInList(userList, item))
00169             AddAppointmentOrdered(empty_list, item);
00170     }
00171     return empty_list;
00172 }
00173
00174 static struct AppointmentList* GetEmptyTimeSlotInRange(struct AppointmentList *userList, time_t start_date,
      time_t end_date)
00175 {
00176     struct AppointmentList *empty_list = CreateAppointmentList();
00177     struct tm timeinfo, timeinfo_tmp;
00178
```

```
00179     //set the time start<end so that we can use difftime() to compare the date.
00180     timeinfo = *localtime (&start_date);
00181     timeinfo.tm_hour = 1;
00182     timeinfo_tmp = timeinfo;
00183     start_date = mktime(&timeinfo_tmp);
00184
00185     timeinfo = *localtime (&end_date);
00186     timeinfo.tm_hour = 2;
00187     timeinfo_tmp = timeinfo;
00188     end_date = mktime(&timeinfo_tmp);
00189
00190     timeinfo = *localtime (&start_date);
00191     while(difftime(start_date, end_date)<0)
00192     {
00193         AddAppointmentFromList(empty_list, GetEmptyTimeSlotInDay(userList, start_date));
00194
00195         timeinfo.tm_mday++;
00196         timeinfo_tmp = timeinfo;
00197         start_date = mktime(&timeinfo_tmp);
00198     }
00199     return empty_list;
00200 }
00201
00202 static time_t GetEarliestStartTime(struct AppointmentList *list)
00203 {
00204     struct Appointment *item = list->head;
00205     time_t earliest = item->start;
00206     while(item)
00207     {
00208         if(difftime(item->start, earliest)<0)
00209             earliest = item->start;
00210         item = item->next;
00211     }
00212     return earliest;
00213 }
00214
00215 static time_t GetLatestEndTime(struct AppointmentList *list)
00216 {
00217     struct Appointment *item = list->head;
00218     time_t latest = item->end;
00219     while(item)
00220     {
00221         if(difftime(latest, item->end)<0)
00222             latest = item->end;
00223         item = item->next;
00224     }
00225     return latest;
00226 }
00227
00228 static struct AppointmentList* GetContinueTimeslotFromList(const struct
      AppointmentList *list, time_t duration)
00229 {
00230     struct Appointment *ptr;
00231     struct Appointment *item = list->head;
00232     struct AppointmentList *ret_list = CreateAppointmentList();
00233     int timeslot = duration / 60 / 30 - 1;  //how many half hour
00234     while(item)
00235     {
00236         ptr = item;
00237         for(int i=0; i<timeslot; i++)
00238             ptr = ptr->next;
00239         if(!ptr)
00240             return ret_list;
00241         if(difftime(ptr->end, item->start)==duration)
00242             AddAppointmentOrdered(ret_list, item);
00243         item = item->next;
00244     }
00245     return ret_list;
00246 }
00247
00248 struct Summary* Schedual_FCFS(struct AppointmentList *inputList)
00249 {
00250     struct Appointment *ptr = inputList->head;
00251     while(ptr)
00252     {
00253         if(IsAllAvailable(ptr))
00254         {
00255             AddToAllAccept(ptr);
00256         }
00257         else
00258         {
00259             AddToAllReject(ptr);
00260         }
00261         ptr = ptr->next;
00262     }
00263
00264     //Summary
```

```
00265     struct Summary *summary = (struct Summary *)malloc(sizeof(struct Summary));
00266     summary->start = GetEarliestStartTime(inputList);
00267     summary->end = GetLatestEndTime(inputList);
00268 
00269     SetAppointmentAccepted(inputList);
00270     ptr = inputList->head;
00271     while(ptr)
00272     {
00273         if(ptr->is_accepted)
00274             summary->total_accepted++;
00275         else
00276             summary->total_rejected++;
00277         ptr = ptr->next;
00278     }
00279     for(int i=0; i<NumOfUser; i++)
00280     {
00281         summary->accepted[i] = user[i].accepted->count;
00282         summary->rejected[i] = user[i].rejected->count;
00283         summary->empty_timeslot[i] = GetEmptyTimeSlotInRange(user[i].accepted, summary->
      start, summary->end)->count;
00284 
00285         // PrintAppointmentList(GetContinueTimeslotFromList(GetEmptyTimeSlotInRange(user[i].accepted,
       summary->start, summary->end), 2*60*30));
00286         // PrintAppointmentList(GetEmptyTimeSlotInRange(user[i].accepted, summary->start, summary->end));
00287     }
00288     return summary;
00289 }
00290 
00291 struct Summary* Schedual_PRIO(struct AppointmentList *inputList)
00292 {
00293     struct Appointment *ptr = inputList->head;
00294     while(ptr)
00295     {
00296         if(IsAllAvailablePriority(ptr))
00297         {
00298             AddToAllAcceptForced(ptr);
00299         }
00300         else
00301         {
00302             AddToAllReject(ptr);
00303         }
00304         ptr = ptr->next;
00305     }
00306 
00307     //Summary
00308     struct Summary *summary = (struct Summary *)malloc(sizeof(struct Summary));
00309     summary->start = GetEarliestStartTime(inputList);
00310     summary->end = GetLatestEndTime(inputList);
00311 
00312     SetAppointmentAccepted(inputList);
00313     ptr = inputList->head;
00314     while(ptr)
00315     {
00316         if(ptr->is_accepted)
00317             summary->total_accepted++;
00318         else
00319             summary->total_rejected++;
00320         ptr = ptr->next;
00321     }
00322     for(int i=0; i<NumOfUser; i++)
00323     {
00324         summary->accepted[i] = user[i].accepted->count;
00325         summary->rejected[i] = user[i].rejected->count;
00326         summary->empty_timeslot[i] = GetEmptyTimeSlotInRange(user[i].accepted, summary->
      start, summary->end)->count;
00327         // PrintAppointmentList(GetEmptyTimeSlotInRange(user[i].accepted, summary->start, summary->end));
00328     }
00329     return summary;
00330 }
00331 
00332 struct Summary* Schedual_OPTI(struct AppointmentList *inputList)
00333 {
00334     struct Summary *summary = Schedual_PRIO(inputList);
00335     time_t day = 60*60*24;
00336 
00337     //For each user, try to reschedule their rejected jobs
00338     for(int i=0; i<NumOfUser; i++)
00339     {
00340         struct Appointment *item = user[i].rejected->head;
00341         NEXT_ITEM:
00342         while(item)
00343         {
00344             time_t ori_start = item->start;
00345             time_t ori_end = item->end;
00346             time_t duration = difftime(item->end, item->start);
00347             struct AppointmentList *list = GetEmptyTimeSlotInRange(user[i].accepted, item->
      start, item->start+3*day);
```

```
00348             struct AppointmentList *c_list = GetContinueTimeslotFromList(list, duration);
00349
00350             struct Appointment *timeslot = c_list->head;
00351             while(timeslot)
00352             {
00353                 item->start = timeslot->start;
00354                 item->end = item->start + duration;
00355
00356                 if(IsAllAvailable(item))
00357                 {
00358                     item->rescheduled = 1;
00359                     AddToAllAccept(item);
00360                     struct Appointment *temp = item;
00361                     item = item->next;
00362                     for(int j=0; j<NumOfUser; j++)
00363                         RemoveItemFromList(user[j].rejected, temp);
00364                     goto NEXT_ITEM;
00365                 }
00366                 item->start = ori_start;
00367                 item->end = ori_end;
00368                 timeslot = timeslot->next;
00369             }
00370             item = item->next;
00371         }
00372     }
00373
00374     //Re-calculate the summary
00375     memset(summary, 0, sizeof(struct Summary));
00376     summary->start = GetEarliestStartTime(inputList);
00377     summary->end = GetLatestEndTime(inputList);
00378     SetAppointmentAccepted(inputList);
00379     struct Appointment *ptr = inputList->head;
00380     while(ptr)
00381     {
00382         if(ptr->is_accepted)
00383             summary->total_accepted++;
00384         else
00385             summary->total_rejected++;
00386         ptr = ptr->next;
00387     }
00388     for(int i=0; i<NumOfUser; i++)
00389     {
00390         summary->accepted[i] = user[i].accepted->count;
00391         summary->rejected[i] = user[i].rejected->count;
00392         summary->empty_timeslot[i] = GetEmptyTimeSlotInRange(user[i].accepted, summary->
      start, summary->end)->count;
00393         // PrintAppointmentList(GetEmptyTimeSlotInRange(user[i].accepted, summary->start, summary->end));
00394     }
00395     return summary;
00396 }
00397
00398 static int rdn(int y, int m, int d) {
00399     if (m < 3)
00400         y--, m += 12;
00401     return 365*y + y/4 - y/100 + y/400 + (153*m - 457)/5 + d - 306;
00402 }
00403
00404 void PrintSummary(struct Summary *summary)
00405 {
00406     float total = summary->total_accepted+summary->total_rejected;
00407     struct tm timeinfo, timeinfo2;
00408     int days;
00409
00410     printf("Performance:\n");
00411     timeinfo = *localtime(&summary->start);
00412     printf("Date start: %4d-%02d-%02d\n", timeinfo.tm_year+1900, timeinfo.tm_mon+1, timeinfo.tm_mday);
00413     timeinfo2 = *localtime(&summary->end);
00414     printf("Date end: %4d-%02d-%02d\n\n", timeinfo2.tm_year+1900, timeinfo2.tm_mon+1, timeinfo2.tm_mday);
00415
00416     printf("Total Number of Appointment Assigned: %d (%.1f%%)\n", summary->
      total_accepted, summary->total_accepted/total*100);
00417     printf("Total Number of Appointment Rejected: %d (%.1f%%)\n", summary->
      total_rejected, summary->total_rejected/total*100);
00418
00419     days = rdn(timeinfo2.tm_year, timeinfo2.tm_mon+1, timeinfo2.tm_mday) - rdn(timeinfo.tm_year, timeinfo.
      tm_mon+1, timeinfo.tm_mday) + 1;
00420     printf("Utilization of Time Slot: (%d days)\n", days+1);
00421     for(int i=0; i<NumOfUser; i++)
00422     {
00423         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
00424     }
00425
00426 }
```

5.11 scheduler.h File Reference

Secheduling 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 folloing 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 folloing 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.

5.11.1 Detailed Description

Secheduling algorithms.

Author

oneonestar `oneonestar@gmail.com`

Version

1.0

Copyright

2015

5.11.2 LICENSE

Definition in file **scheduler.h**.

5.11.3 Function Documentation

5.11.3.1 void PrintSummary (struct Summary * *summary*)

Print out the summary about the scheduling.

Definition at line **404** of file **scheduler.c**.

5.11.3.2 struct Summary* Schedual_FCFS (struct AppointmentList * *inputList*)

First come first served. The order is folloing the input order. The result will be putted into the each user's appointment lists (accept / reject).

Definition at line **248** of file **scheduler.c**.

5.11.3.3 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).

Definition at line **332** of file **scheduler.c**.

5.11.3.4 struct Summary* Schedual_PRIO (struct AppointmentList * *inputList*)

Priority. The order is folloing the pre-defined priority. The result will be putted into the each user's appointment lists (accept / reject).

Definition at line **291** of file **scheduler.c**.

5.12 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* Schedual_FCFS(struct AppointmentList *inputList);
00043
00048 struct Summary* Schedual_PRIO(struct AppointmentList *inputList);
00049
00054 struct Summary* Schedual_OPTI(struct AppointmentList *inputList);
00055
00059 void PrintSummary(struct Summary *summary);
00060
00061 #endif
```

5.13 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 **strcicmp** (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.

5.13.1 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.c**.

5.13.3 Function Documentation

5.13.3.1 int GetUserID (const char * *username*)

Return the user id that have the same username.

Parameters

username	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**.

5.13.3.2 void PrintAccepted (const struct User * *user*)

Print the accepted list for user.

Definition at line **48** of file **user.c**.

5.13.3.3 void PrintAllUser ( )

Print the accepted & reject list for all users.

Definition at line **63** of file **user.c**.

5.13.3.4 void PrintRejected (const struct User * *user*)

Print the rejected list for user.

Definition at line **56** of file **user.c**.

5.13.3.5 int strcicmp (char const * *a*, char const * *b*)

Definition at line **29** of file **user.c**.

5.14 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 strcicmp(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 (!strcicmp(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 }
```

5.15 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**

5.15.1 Detailed Description

Handling each users.

Author

oneonestar oneonestar@gmail.com

Version

1.0

Copyright

2015

5.15.2 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**.

5.15.3 Macro Definition Documentation

5.15.3.1 #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**.

5.15.3.2 #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**.

5.15.4 Function Documentation

5.15.4.1 int GetUserID (const char * *username*)

Return the user id that have the same username.

Parameters

username	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**.

5.15.4.2 void PrintAccepted (const struct User * *user*)

Print the accepted list for user.

Definition at line **48** of file **user.c**.

5.15.4.3 void PrintAllUser ( )

Print the accepted & reject list for all users.

Definition at line **63** of file **user.c**.

5.15.4.4 void PrintRejected (const struct User * *user*)

Print the rejected list for user.

Definition at line **56** of file **user.c**.

5.15.5 Variable Documentation

5.15.5.1 int NumOfUser

Global variable storing current number of user.

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

5.15.5.2 struct User user[USER_NUMBER]

Global variable storing the user data.

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

5.16 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