Coin Selection

1.0

Generated by Doxygen 1.11.0

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 Action Struct Reference

`#include <fee.h>`

Public Attributes

- long long amount
- operation_type operation
- long long time

3.1.1 Member Data Documentation

3.1.1.1 amount

`long long Action::amount`

3.1.1.2 operation

`operation_type Action::operation`

3.1.1.3 time

`long long Action::time`

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

- include/fee.h

3.2 Coin Struct Reference

`#include <common.h>`

Public Attributes

- long long uniqueId
- Denomination denomination
- time_t creation_timestamp

3.2.1 Member Data Documentation

3.2.1.1 creation_timestamp

`time_t Coin::creation_timestamp`

3.2.1.2 denomination

`Denomination Coin::denomination`

3.2.1.3 uniqueId

`long long Coin::uniqueId`

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

- include/common.h

3.3 coin_wrapper Struct Reference

`#include <coin_selection.h>`

Public Attributes

- Coin * coin
- double score

3.3.1 Member Data Documentation

3.3.1.1 coin

`Coin* coin_wrapper::coin`

3.3.1.2 score

```
double coin_wrapper::score
```

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

- include/coin_selection.h

3.4 Denomination Struct Reference

```
#include <common.h>
```

Public Attributes

- char * name
- long long amount
- Rules rules

3.4.1 Member Data Documentation

3.4.1.1 amount

```
long long Denomination::amount
```

3.4.1.2 name

```
char* Denomination::name
```

3.4.1.3 rules

```
Rules Denomination::rules
```

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

- include/common.h

3.5 Duration Struct Reference

```
#include <common.h>
```

Public Attributes

- long long time

3.5.1 Member Data Documentation

3.5.1.1 time

```
long long Duration::time
```

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

- include/common.h

3.6 Durations Struct Reference

```
#include <common.h>
```

Public Attributes

- Duration legal
- Duration deposit
- Duration withdraw

3.6.1 Member Data Documentation

3.6.1.1 deposit

```
Duration Durations::deposit
```

3.6.1.2 legal

```
Duration Durations::legal
```

3.6.1.3 withdraw

```
Duration Durations::withdraw
```

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

- include/common.h

3.7 Fee Struct Reference

```
#include <common.h>
```

Public Attributes

- long long fee_satoshis
- float percentage_fee

3.7.1 Member Data Documentation

3.7.1.1 fee_satoshis

```
long long Fee::fee_satoshis
```

3.7.1.2 percentage_fee

```
float Fee::percentage_fee
```

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

- include/common.h

3.8 Fees Struct Reference

```
#include <common.h>
```

Public Attributes

- Fee deposit_fee
- Fee refund_fee
- Fee withdraw_fee
- Fee refresh_fee

3.8.1 Member Data Documentation

3.8.1.1 deposit_fee

```
Fee Fees::deposit_fee
```

3.8.1.2 refresh_fee

```
Fee Fees::refresh_fee
```

3.8.1.3 refund_fee

```
Fee Fees::refund_fee
```

3.8.1.4 withdraw_fee

```
Fee Fees::withdraw_fee
```

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

- include/common.h

3.9 GlobalFees Struct Reference

```
#include <common.h>
```

Public Attributes

- Fee wire_fee
- Fee closing_fee

3.9.1 Member Data Documentation

3.9.1.1 closing_fee

```
Fee GlobalFees::closing_fee
```

3.9.1.2 wire_fee

```
Fee GlobalFees::wire_fee
```

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

- include/common.h

3.10 Rules Struct Reference

```
#include <common.h>
```

Public Attributes

- int rsa_keysize
- char * cipher
- Fees fees
- Durations durations

3.10.1 Member Data Documentation

3.10.1.1 cipher

```
char* Rules::cipher
```

3.10.1.2 durations

```
Durations Rules::durations
```

3.10.1.3 fees

```
Fees Rules::fees
```

3.10.1.4 rsa_keysize

```
int Rules::rsa_keysize
```

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

- include/common.h

3.11 ThreadArgs Struct Reference

```
#include <common.h>
```

Public Attributes

- const char * filepath
- Wallet denomination_wallet

3.11.1 Member Data Documentation

3.11.1.1 denomination_wallet

```
Wallet ThreadArgs::denomination_wallet
```

3.11.1.2 filepath

```
const char* ThreadArgs::filepath
```

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

- include/common.h

3.12 User Struct Reference

`#include <user.h>`

Public Attributes

- char * name
- Type type
- Action * actions
- Wallet wallet

3.12.1 Member Data Documentation

3.12.1.1 actions

```
Action* User::actions
```

3.12.1.2 name

```
char* User::name
```

3.12.1.3 type

```
Type User::type
```

3.12.1.4 wallet

```
Wallet User::wallet
```

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

- include/user.h

3.13 Wallet Struct Reference

`#include <common.h>`

Public Attributes

- Coin * coins
- int num_coins
- GlobalFees global_fees

3.13.1 Member Data Documentation

3.13.1.1 coins

```
Coin* Wallet::coins
```

3.13.1.2 global_fees

```
GlobalFees Wallet::global_fees
```

3.13.1.3 num_coins

```
int Wallet::num_coins
```

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

- include/common.h

Chapter 4

File Documentation

4.1 include/coin_selection.h File Reference

```
#include "common.h"
#include "fee.h"
```

Classes

- struct coin_wrapper

Enumerations

- enum strategy {
  MAX_BILLS , MIN_BILLS , CLOSEST_TO_EXPIRE_MIN_BILLS , CLOSEST_TO_EXPIRE_MAX_BILLS ,
  MAX_BILLS_TIME_TO_EXPIRE_WEIGHTED , RANDOM , EVEN_FROM_MIN_TO_MAX , EVEN_FROM_MAX_TO_MIN
  ,
  GREEDY_MIN_TO_MAX , NUMBER_OF_STRATEGIES }

Functions

- Coin * allocate_coins_for_deposit (Wallet wallet, long long amount, strategy strategy, long long time, int *num_allocated_coins, long long *allocated_amount, Wallet denomination_wallet)

  Allocate coins from the wallet according to the specified strategy.
- void remove_selected_coins (Wallet *wallet, Coin *coins, int num_coins)

  Remove selected coins from the wallet.
- void add_coins_to_wallet (Wallet *wallet, Coin *coins, int num_coins)

  Add coins to the wallet.
- long long calculate_total_fee (Coin *coins, int num_coins, operation_type operation)

  Calculate the total fee for a set of coins based on the operation type.
- long long calculate_renew_fee (Wallet wallet, long long time)

  Calculate the renew fee for the wallet based on the current time.
- Coin * generate_withdraw_coins (long long amount, long long time, Wallet default_wallet, int *num_coins)

  Generate coins for withdrawal from the wallet.

4.1.1 Enumeration Type Documentation

4.1.1.1 strategy

```
enum strategy
```

Enumerator

MAX_BILLS	
MIN_BILLS	
CLOSEST_TO_EXPIRE_MIN_BILLS	
CLOSEST_TO_EXPIRE_MAX_BILLS	
MAX_BILLS_TIME_TO_EXPIRE_WEIGHTED	
RANDOM	
EVEN_FROM_MIN_TO_MAX	
EVEN_FROM_MAX_TO_MIN	
GREEDY_MIN_TO_MAX	
NUMBER_OF_STRATEGIES	

4.1.2 Function Documentation

4.1.2.1 add_coins_to_wallet()

```
void add_coins_to_wallet (
            Wallet * wallet,
            Coin * coins,
            int num_coins)
```

Add coins to the wallet.

Parameters

wallet	Pointer to the wallet.
coins	The array of coins to be added.
num_coins	The number of coins to be added.

4.1.2.2 allocate_coins_for_deposit()

```
Coin * allocate_coins_for_deposit (
            Wallet wallet,
            long long amount,
            strategy strategy,
            long long time,
            int * num_allocated_coins,
            long long * allocated_amount,
            Wallet denomination_wallet)
```

Allocate coins from the wallet according to the specified strategy.

Parameters

wallet	The wallet containing the coins.
amount	The target amount to allocate.
strategy	The strategy to use for allocation.
time	The current time in seconds.
num_allocated_coins	Pointer to store the number of allocated coins.
allocated_amount	Pointer to store the total allocated amount.
denomination_wallet	The wallet containing the denomination information.

Returns

An array of allocated coins.

4.1.2.3 calculate_renew_fee()

```
long long calculate_renew_fee (
            Wallet wallet,
            long long time)
```

Calculate the renew fee for the wallet based on the current time.

Parameters

wallet	The wallet containing the coins.
time	The current time in seconds.

Returns

The total renew fee for the wallet.

4.1.2.4 calculate_total_fee()

```
long long calculate_total_fee (
            Coin * coins,
            int num_coins,
            operation_type operation)
```

Calculate the total fee for a set of coins based on the operation type.

Parameters

coins	The array of coins.
num_coins	The number of coins.
operation	The operation type.

Returns

The total fee for the specified operation.

4.1.2.5 generate_withdraw_coins()

```
Coin * generate_withdraw_coins (
            long long amount,
            long long time,
            Wallet default_wallet,
            int * num_coins)
```

Generate coins for withdrawal from the wallet.

Parameters

wallet	The wallet containing the coins.
amount	The target amount to withdraw.
time	The current time in seconds.
default_wallet	The wallet containing the default coins for generation.
num_coins	Pointer to store the number of generated coins.

Returns

An array of generated coins.

4.1.2.6 remove_selected_coins()

```
void remove_selected_coins (
            Wallet * wallet,
            Coin * coins,
            int num_coins)
```

Remove selected coins from the wallet.

Parameters

wallet	Pointer to the wallet.
coins	The array of coins to be removed.
num_coins	The number of coins to be removed.

4.2 coin_selection.h

Go to the documentation of this file.

```
00001 //
00002 // Created by Bohdan Potuzhnyi and Vlada Svirsh on 04.03.2024.
00003 // coin_selection.h
00004 //
00005
00006 #ifndef COIN_SELECTION_H
00007 #define COIN_SELECTION_H
00008
00009 #include "common.h"
00010 #include "fee.h"
00011
00012 typedef enum {
00013     MAX_BILLS,
00014     MIN_BILLS,
00015     CLOSEST_TO_EXPIRE_MIN_BILLS,
00016     CLOSEST_TO_EXPIRE_MAX_BILLS,
00017     MAX_BILLS_TIME_TO_EXPIRE_WEIGHTED,
00018     RANDOM,
00019     EVEN_FROM_MIN_TO_MAX,
00020     EVEN_FROM_MAX_TO_MIN,
00021     GREEDY_MIN_TO_MAX,
00022     NUMBER_OF_STRATEGIES
00023 } strategy;
00024
00025 typedef struct {
00026     Coin *coin;
00027     double score;
00028 } coin_wrapper;
00029
```

```
00030 Coin* allocate_coins_for_deposit(Wallet wallet, long long amount, strategy strategy, long long time,
      int *num_allocated_coins, long long *allocated_amount, Wallet denomination_wallet);
00031
00032 void remove_selected_coins(Wallet* wallet, Coin* coins, int num_coins);
00033
00034 void add_coins_to_wallet(Wallet* wallet, Coin* coins, int num_coins);
00035
00036 long long calculate_total_fee(Coin* coins, int num_coins, operation_type operation);
00037
00038 long long calculate_renew_fee(Wallet wallet, long long time);
00039
00040 Coin* generate_withdraw_coins(long long amount, long long time, Wallet default_wallet, int
      *num_coins);
00041
00042
00043 #endif //COIN_SELECTION_H
```

4.3 include/common.h File Reference

`#include <time.h>`

Classes

- struct Fee
- struct Fees
- struct GlobalFees
- struct Duration
- struct Durations
- struct Rules
- struct Denomination
- struct Coin
- struct Wallet
- struct ThreadArgs

4.4 common.h

Go to the documentation of this file.

```
00001 //
00002 // Created by Bohdan Potuzhnyi and Vlada Svirsh on 04.03.2024.
00003 // common.h
00004 //
00005
00006 #ifndef COMMON_H
00007 #define COMMON_H
00008
00009 #include <time.h>
00010
00011 typedef struct {
00012     long long fee_satoshis;
00013     float percentage_fee;
00014 } Fee;
00015
00016 typedef struct {
00017     Fee deposit_fee;
00018     Fee refund_fee;
00019     Fee withdraw_fee;
00020     Fee refresh_fee;
00021 } Fees;
00022
00023 typedef struct{
00024     Fee wire_fee;
00025     Fee closing_fee;
00026 } GlobalFees;
00027
```

```
00028 typedef struct {
00029     long long time;
00030 } Duration;
00031
00032 typedef struct {
00033     Duration legal;
00034     Duration deposit;
00035     Duration withdraw;
00036 } Durations;
00037
00038 typedef struct {
00039     int rsa_keysize;
00040     char* cipher;
00041     Fees fees;
00042     Durations durations;
00043 } Rules;
00044
00045 typedef struct{
00046     char* name;
00047     long long amount;
00048     Rules rules;
00049 } Denomination;
00050
00051 typedef struct {
00052     long long uniqueId;
00053     Denomination denomination;
00054     time_t creation_timestamp;
00055 } Coin;
00056
00057 typedef struct {
00058     Coin* coins;
00059     int num_coins;
00060     GlobalFees global_fees;
00061 } Wallet;
00062
00063 typedef struct {
00064     const char* filepath;
00065     Wallet denomination_wallet;
00066 } ThreadArgs;
00067
00068 #endif //COMMON_H
```

4.5 include/fee.h File Reference

```
#include "common.h"
```

Classes

- struct Action

Enumerations

- enum operation_type {
  DEPOSIT_OP , WITHDRAW_OP , REFUND_OP , REFRESH_OP ,
  WIRE_OP , CLOSE_OP }

Functions

- long long calculate_fee (Coin coin, operation_type operation)

  Calculate the fee for a given coin and operation type.

4.5.1 Enumeration Type Documentation

4.5.1.1 operation_type

```
enum operation_type
```

Enumerator

DEPOSIT_OP	
WITHDRAW_OP	
REFUND_OP	
REFRESH_OP	
WIRE_OP	
CLOSE_OP	

4.5.2 Function Documentation

4.5.2.1 calculate_fee()

```
long long calculate_fee (
            Coin coin,
            operation_type operation)
```

Calculate the fee for a given coin and operation type.

Parameters

coin	The coin for which the fee is being calculated.
operation	The type of operation (e.g., DEPOSIT_OP, WITHDRAW_OP, REFUND_OP, REFRESH_OP).

Returns

The total fee in satoshis for the specified operation.

4.6 fee.h

Go to the documentation of this file.

```
00001 //
00002 // Created by Bohdan Potuzhnyi and Vlada Svirsh on 04.03.2024.
00003 // fee.h
00004 //
00005
00006 #ifndef FEE_H
00007 #define FEE_H
00008
00009 #include "common.h" // Include the common definitions
00010
00011 typedef enum {
00012     DEPOSIT_OP,
00013     WITHDRAW_OP,
00014     REFUND_OP,
00015     REFRESH_OP,
00016     WIRE_OP,
00017     CLOSE_OP
00018 } operation_type;
00019
00020 typedef struct{
00021     long long amount;
00022     operation_type operation;
00023     long long time;
00024 } Action;
00025
00026 long long calculate_fee(Coin coin, operation_type operation);
00027
00028 #endif // FEE_H
00029
```

4.7 include/generator.h File Reference

```
#include "user.h"
#include "fee.h"
```

Functions

- long long rand_range_long (long long min, long long max)
  Generate a random long long integer between min and max, inclusive.
- void generate_actions_for_user (User user, Action **actions, int *size)
  Generate actions for a given user based on their type.
- void generate_and_save_actions (const char *base_dir, int num_users)
  Generate actions for multiple users and save them to CSV files.

4.7.1 Function Documentation

4.7.1.1 generate_actions_for_user()

```
void generate_actions_for_user (
            User user,
            Action ** actions,
            int * size)
```

Generate actions for a given user based on their type.

Parameters

user	The user for whom actions are being generated.
actions	Pointer to an array of actions to be generated.
size	Pointer to the size of the actions array.

4.7.1.2 generate_and_save_actions()

```
void generate_and_save_actions (
            const char * base_dir,
            int num_users)
```

Generate actions for multiple users and save them to CSV files.

Parameters

base_dir	The base directory where the CSV files will be saved.
num_users	The number of users for whom actions will be generated.

4.7.1.3 rand_range_long()

```
long long rand_range_long (
            long long min,
            long long max)
```

Generate a random long long integer between min and max, inclusive.

Parameters

min	The minimum value.
max	The maximum value.

Returns

A random long long integer between min and max.

4.8 generator.h

Go to the documentation of this file.

```
00001 //
00002 // Created by Bohdan Potuzhnyi on 24.03.2024.
00003 //
00004
00005 #ifndef COIN_SELECTION_C_GENERATOR_H
00006 #define COIN_SELECTION_C_GENERATOR_H
00007
00008 #include "user.h"
00009 #include "fee.h"
00010
00011
00012 long long rand_range_long(long long min, long long max);
00013 void generate_actions_for_user(User user, Action **actions, int *size);
00014 void generate_and_save_actions(const char *base_dir, int num_users);
00015
00016 #endif //COIN_SELECTION_C_GENERATOR_H
```

4.9 include/parser.h File Reference

```
#include "common.h"
```

Functions

- Wallet parse_wallet_config (const char *config_path)

  Parse the wallet configuration from a file.
- Wallet parse_wallet_config_json (const char *config_path)

  Parse the JSON configuration for a wallet.
- long long time_to_seconds (const char *time_str)

  Convert a time string to seconds.

4.9.1 Function Documentation

4.9.1.1 parse_wallet_config()

```
Wallet parse_wallet_config (
            const char * config_path)
```

Parse the wallet configuration from a file.

Parameters

config_path	The path to the configuration file.

Returns

The parsed Wallet structure.

4.9.1.2 parse_wallet_config_json()

```
Wallet parse_wallet_config_json (
            const char * filename)
```

Parse the JSON configuration for a wallet.

Parameters

filename	The path to the JSON configuration file.

Returns

The parsed Wallet structure.

4.9.1.3 time_to_seconds()

```
long long time_to_seconds (
            const char * time_str)
```

Convert a time string to seconds.

Parameters

time_str	The time string to convert.

Returns

The number of seconds represented by the time string, or -1 if the format is invalid.

4.10 parser.h

Go to the documentation of this file.

```
00001 //
00002 // Created by Bohdan Potuzhnyi and Vlada Svirsh on 10.03.2024.
00003 // parser.h
00004 //
00005
00006 #ifndef PARSER_H
00007 #define PARSER_H
00008
00009 #include "common.h"
00010
00011 Wallet parse_wallet_config(const char* config_path);
00012 Wallet parse_wallet_config_json(const char* config_path);
00013
00014 long long time_to_seconds(const char* time_str);
00015
00016 #endif //PARSER_H
```

4.11 include/simulation.h File Reference

```
#include "common.h"
#include "fee.h"
#include "user.h"
#include "coin_selection.h"
```

Functions

- void simulate_user_actions (int user_index, User user, Wallet denomination_wallet, int num_actions, strategy strategy)

  Simulate user actions and write results to a file.

4.11.1 Function Documentation

4.11.1.1 simulate_user_actions()

```
void simulate_user_actions (
            int user_index,
            User user,
            Wallet denomination_wallet,
            int num_actions,
            strategy strategy)
```

Simulate user actions and write results to a file.

Parameters

user_index	The index of the user.
user	The user structure containing user information.
denomination_wallet	The wallet containing denominations.
num_actions	The number of actions to simulate.
strategy	The strategy to use for coin allocation.

4.12 simulation.h

Go to the documentation of this file.

```
00001 //
00002 // Created by Bohdan Potuzhnyi on 29.04.2024.
00003 //
00004
00005 #ifndef COIN_SELECTION_C_SIMULATION_H
00006 #define COIN_SELECTION_C_SIMULATION_H
00007
00008 #include "common.h"
00009 #include "fee.h"
00010 #include "user.h"
00011 #include "coin_selection.h"
00012
00013 void simulate_user_actions(int user_index, User user, Wallet denomination_wallet, int num_actions,
      strategy strategy);
00014
00015 #endif //COIN_SELECTION_C_SIMULATION_H
```

4.13 include/user.h File Reference

```
#include "common.h"
#include "fee.h"
```

Classes

- struct User

Enumerations

- enum Type {
  STUDENT , STUDENT_STATIC , BUSINESS_OWNER , RETIRED ,
  FAMILY , FREELANCER , TEACHER , ARTIST ,
  NUMBER_OF_USERS }

Functions

- void generate_actions_for_student (Action **actions, int *size, int days)
  Generate actions for a student user type.
- void generate_actions_for_student_static (Action **actions, int *size, int days)
  Generate static actions for a student user type.
- void generate_actions_for_business_owner (Action **actions, int *size, int days)
  Generate actions for a business owner user type.
- void generate_actions_for_retired (Action **actions, int *size, int days)
  Generate actions for a retired user type.
- void generate_actions_for_family (Action **actions, int *size, int days)
  Generate actions for a family user type.
- void generate_actions_for_freelancer (Action **actions, int *size, int days)
  Generate actions for a freelancer user type.
- void generate_actions_for_teacher (Action **actions, int *size, int days)
  Generate actions for a teacher user type.
- void generate_actions_for_artist (Action **actions, int *size, int days)
  Generate actions for an artist user type.

4.13.1 Enumeration Type Documentation

4.13.1.1 Type

```
enum Type
```

Enumerator

STUDENT	
STUDENT_STATIC	
BUSINESS_OWNER	
RETIRED	
FAMILY	
FREELANCER	
TEACHER	
ARTIST	
NUMBER_OF_USERS	

4.13.2 Function Documentation

4.13.2.1 generate_actions_for_artist()

```
void generate_actions_for_artist (
            Action ** actions,
            int * size,
            int days)
```

Generate actions for an artist user type.

Parameters

actions	Pointer to the array of actions to be generated.
size	Pointer to the size of the actions array.
days	The number of days to simulate actions for.

4.13.2.2 generate_actions_for_business_owner()

```
void generate_actions_for_business_owner (
            Action ** actions,
            int * size,
            int days)
```

Generate actions for a business owner user type.

Parameters

actions	Pointer to the array of actions to be generated.
size	Pointer to the size of the actions array.
days	The number of days to simulate actions for.

4.13.2.3 generate_actions_for_family()

```
void generate_actions_for_family (
            Action ** actions,
            int * size,
            int days)
```

Generate actions for a family user type.

Parameters

actions	Pointer to the array of actions to be generated.
size	Pointer to the size of the actions array.
days	The number of days to simulate actions for.

4.13.2.4 generate_actions_for_freelancer()

```
void generate_actions_for_freelancer (
            Action ** actions,
            int * size,
            int days)
```

Generate actions for a freelancer user type.

Parameters

actions	Pointer to the array of actions to be generated.
size	Pointer to the size of the actions array.
days	The number of days to simulate actions for.

4.13.2.5 generate_actions_for_retired()

```
void generate_actions_for_retired (
            Action ** actions,
            int * size,
            int days)
```

Generate actions for a retired user type.

Parameters

actions	Pointer to the array of actions to be generated.
size	Pointer to the size of the actions array.
days	The number of days to simulate actions for.

4.13.2.6 generate_actions_for_student()

```
void generate_actions_for_student (
            Action ** actions,
            int * size,
            int days)
```

Generate actions for a student user type.

Parameters

actions	Pointer to the array of actions to be generated.
size	Pointer to the size of the actions array.
days	The number of days to simulate actions for.

4.13.2.7 generate_actions_for_student_static()

```
void generate_actions_for_student_static (
            Action ** actions,
            int * size,
            int days)
```

Generate static actions for a student user type.

Parameters

actions	Pointer to the array of actions to be generated.
size	Pointer to the size of the actions array.
days	The number of days to simulate actions for.

4.13.2.8 generate_actions_for_teacher()

```
void generate_actions_for_teacher (
            Action ** actions,
            int * size,
            int days)
```

Generate actions for a teacher user type.

Parameters

actions	Pointer to the array of actions to be generated.
size	Pointer to the size of the actions array.
days	The number of days to simulate actions for.

4.14 user.h

Go to the documentation of this file.

```
00001 //
00002 // Created by Bohdan Potuzhnyi and Vlada Svirsh on 13.03.2024.
00003 // user.h
00004 //
00005
00006 #ifndef USER_H
00007 #define USER_H
00008
00009 #include "common.h"
00010 #include "fee.h"
00011
00012 typedef enum{
00013     STUDENT,
00014     STUDENT_STATIC,
00015     BUSINESS_OWNER,
00016     RETIRED,
00017     FAMILY,
00018     FREELANCER,
00019     TEACHER,
00020     ARTIST,
00021     NUMBER_OF_USERS
00022 } Type;
00023
00024 typedef struct {
00025     char* name;
00026     Type type;
00027     Action* actions;
00028     Wallet wallet;
00029 } User;
00030
00031 void generate_actions_for_student(Action **actions, int *size, int days);
00032 void generate_actions_for_student_static(Action **actions, int *size, int days);
00033 void generate_actions_for_business_owner(Action **actions, int *size, int days);
00034 void generate_actions_for_retired(Action **actions, int *size, int days);
00035 void generate_actions_for_family(Action **actions, int *size, int days);
00036 void generate_actions_for_freelancer(Action **actions, int *size, int days);
00037 void generate_actions_for_teacher(Action **actions, int *size, int days);
00038 void generate_actions_for_artist(Action **actions, int *size, int days);
00039
00040 #endif // USER_H
```

4.15 src/coin_selection.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include "coin_selection.h"
```

Functions

- double calculate_coin_score (Coin *coin, long long currentTime, long long maxDenom, long long minDenom)

  Calculate the score of a coin based on its expiration time and denomination.
- int compare_creation_time_asc (const void *a, const void *b)

  Comparison function for sorting coins by creation timestamp in ascending order.
- int compare_denomination_asc (const void *a, const void *b)

  Comparison function for sorting denominations in ascending order.
- int compare_denomination_desc_ll (const void *a, const void *b)

  Comparison function for sorting denominations in descending order.
- int compare_coin_wrappers (const void *a, const void *b)

  Comparison function for sorting coin wrappers by score in descending order.
- int compare_coins_desc (const void *a, const void *b)

  Comparison function for sorting coins by denomination amount in descending order.
- int compare_coins_asc (const void *a, const void *b)

  Comparison function for sorting coins by denomination amount in ascending order.
- Coin * allocate_max_bills (Wallet wallet, long long amount, int *num_allocated_coins, long long *allocated↩ _amount)

  Allocate coins from the wallet to maximize the number of bills used.
- Coin * allocate_min_bills (Wallet wallet, long long amount, int *num_allocated_coins, long long *allocated↩ _amount)

  Allocate coins from the wallet to minimize the number of bills used.
- int compare_expiry_time_amount (const void *a, const void *b)

  Comparison function for sorting coins by expiration time, then by denomination amount.
- Coin * allocate_closest_to_expire_min_bills (Wallet wallet, long long amount, int *num_allocated_coins, long long *allocated_amount)

  Allocate coins from the wallet that are closest to expiration and minimize the number of bills used.
- int compare_expiry_time_amount_reverse (const void *a, const void *b)

  Comparison function for sorting coins by expiration time, then by denomination amount in reverse order.
- Coin * allocate_closest_to_expire_max_bills (Wallet wallet, long long amount, int *num_allocated_coins, long long *allocated_amount)

  Allocate coins from the wallet that are closest to expiration and maximize the number of bills used.
- Coin * allocate_random_bills (Wallet wallet, long long amount, int *num_allocated_coins, long long *allocated_amount)

  Allocate coins from the wallet randomly until the desired amount is reached.
- Coin * allocate_max_bills_time_to_expire_weighted (Wallet wallet, long long amount, int *num_allocated_↩ coins, long long *allocated_amount, long long currentTime)

  Allocate coins from the wallet to maximize the number of bills used, weighted by time to expiration.
- Coin * allocate_coins_even_from_min_to_max (Wallet wallet, long long amount, int *num_allocated_coins, long long *allocated_amount, Wallet denomination_wallet)

  Allocate coins from the wallet evenly from the smallest to the largest denomination.
- Coin * allocate_coins_even_from_max_to_min (Wallet wallet, long long amount, int *num_allocated_coins, long long *allocated_amount, Wallet denomination_wallet)

  Allocate coins from the wallet evenly from the largest to the smallest denomination.

- Coin * allocate_coins_greedy_min_to_max (Wallet wallet, long long amount, int *num_allocated_coins, long long *allocated_amount, Wallet denomination_wallet)

  Allocate coins from the wallet using a greedy algorithm from the smallest to the largest denomination.
- Coin * allocate_coins_for_deposit (Wallet wallet, long long amount, strategy strategy, long long time, int *num_allocated_coins, long long *allocated_amount, Wallet denomination_wallet)

  Allocate coins from the wallet according to the specified strategy.
- int compare_denomination_desc (const void *a, const void *b)

  Comparison function for sorting coins in descending order by denomination amount.
- Coin * generate_withdraw_coins (long long amount, long long time, Wallet default_wallet, int *num_coins)

  Generate coins for withdrawal from the wallet.
- void add_coins_to_wallet (Wallet *wallet, Coin *coins, int num_coins)

  Add coins to the wallet.
- void remove_selected_coins (Wallet *wallet, Coin *coins, int num_coins)

  Remove selected coins from the wallet.
- long long calculate_total_fee (Coin *coins, int num_coins, operation_type operation)

  Calculate the total fee for a set of coins based on the operation type.
- long long calculate_renew_fee (Wallet wallet, long long time)

  Calculate the renew fee for the wallet based on the current time.

4.15.1 Function Documentation

4.15.1.1 add_coins_to_wallet()

```
void add_coins_to_wallet (
            Wallet * wallet,
            Coin * coins,
            int num_coins)
```

Add coins to the wallet.

Parameters

wallet	Pointer to the wallet.
coins	The array of coins to be added.
num_coins	The number of coins to be added.

4.15.1.2 allocate_closest_to_expire_max_bills()

```
Coin * allocate_closest_to_expire_max_bills (
            Wallet wallet,
            long long amount,
            int * num_allocated_coins,
            long long * allocated_amount)
```

Allocate coins from the wallet that are closest to expiration and maximize the number of bills used.

Parameters

wallet	The wallet containing the coins.
amount	The target amount to allocate.
num_allocated_coins	Pointer to store the number of allocated coins.
allocated_amount	Pointer to store the total allocated amount.

Returns

An array of allocated coins.

4.15.1.3 allocate_closest_to_expire_min_bills()

```
Coin * allocate_closest_to_expire_min_bills (
            Wallet wallet,
            long long amount,
            int * num_allocated_coins,
            long long * allocated_amount)
```

Allocate coins from the wallet that are closest to expiration and minimize the number of bills used.

Parameters

wallet	The wallet containing the coins.
amount	The target amount to allocate.
num_allocated_coins	Pointer to store the number of allocated coins.
allocated_amount	Pointer to store the total allocated amount.

Returns

An array of allocated coins.

4.15.1.4 allocate_coins_even_from_max_to_min()

```
Coin * allocate_coins_even_from_max_to_min (
            Wallet wallet,
            long long amount,
            int * num_allocated_coins,
            long long * allocated_amount,
            Wallet denomination_wallet)
```

Allocate coins from the wallet evenly from the largest to the smallest denomination.

Parameters

wallet	The wallet containing the coins.
amount	The target amount to allocate.
num_allocated_coins	Pointer to store the number of allocated coins.
allocated_amount	Pointer to store the total allocated amount.
denomination_wallet	The wallet containing the denomination information.

Returns

An array of allocated coins.

4.15.1.5 allocate_coins_even_from_min_to_max()

```
Coin * allocate_coins_even_from_min_to_max (
            Wallet wallet,
            long long amount,
            int * num_allocated_coins,
            long long * allocated_amount,
            Wallet denomination_wallet)
```

Allocate coins from the wallet evenly from the smallest to the largest denomination.

Parameters

wallet	The wallet containing the coins.
amount	The target amount to allocate.
num_allocated_coins	Pointer to store the number of allocated coins.
allocated_amount	Pointer to store the total allocated amount.
denomination_wallet	The wallet containing the denomination information.

Returns

An array of allocated coins.

4.15.1.6 allocate_coins_for_deposit()

```
Coin * allocate_coins_for_deposit (
            Wallet wallet,
            long long amount,
            strategy strategy,
            long long time,
            int * num_allocated_coins,
            long long * allocated_amount,
            Wallet denomination_wallet)
```

Allocate coins from the wallet according to the specified strategy.

Parameters

wallet	The wallet containing the coins.
amount	The target amount to allocate.
strategy	The strategy to use for allocation.
time	The current time in seconds.
num_allocated_coins	Pointer to store the number of allocated coins.
allocated_amount	Pointer to store the total allocated amount.
denomination_wallet	The wallet containing the denomination information.

Returns

An array of allocated coins.

4.15.1.7 allocate_coins_greedy_min_to_max()

```
Coin * allocate_coins_greedy_min_to_max (
            Wallet wallet,
            long long amount,
            int * num_allocated_coins,
            long long * allocated_amount,
            Wallet denomination_wallet)
```

Allocate coins from the wallet using a greedy algorithm from the smallest to the largest denomination.

Parameters

wallet	The wallet containing the coins.
amount	The target amount to allocate.
num_allocated_coins	Pointer to store the number of allocated coins.
allocated_amount	Pointer to store the total allocated amount.
denomination_wallet	The wallet containing the denomination information.

Returns

An array of allocated coins.

4.15.1.8 allocate_max_bills()

```
Coin * allocate_max_bills (
            Wallet wallet,
            long long amount,
            int * num_allocated_coins,
            long long * allocated_amount)
```

Allocate coins from the wallet to maximize the number of bills used.

Parameters

wallet	The wallet containing the coins.
amount	The target amount to allocate.
num_allocated_coins	Pointer to store the number of allocated coins.
allocated_amount	Pointer to store the total allocated amount.

Returns

An array of allocated coins.

4.15.1.9 allocate_max_bills_time_to_expire_weighted()

```
Coin * allocate_max_bills_time_to_expire_weighted (
            Wallet wallet,
            long long amount,
            int * num_allocated_coins,
            long long * allocated_amount,
            long long currentTime)
```

Allocate coins from the wallet to maximize the number of bills used, weighted by time to expiration.

Parameters

wallet	The wallet containing the coins.
amount	The target amount to allocate.
num_allocated_coins	Pointer to store the number of allocated coins.
allocated_amount	Pointer to store the total allocated amount.
currentTime	The current time in seconds.

Returns

An array of allocated coins.

4.15.1.10 allocate_min_bills()

```
Coin * allocate_min_bills (
            Wallet wallet,
            long long amount,
            int * num_allocated_coins,
            long long * allocated_amount)
```

Allocate coins from the wallet to minimize the number of bills used.

Parameters

wallet	The wallet containing the coins.
amount	The target amount to allocate.
num_allocated_coins	Pointer to store the number of allocated coins.
allocated_amount	Pointer to store the total allocated amount.

Returns

An array of allocated coins.

4.15.1.11 allocate_random_bills()

```
Coin * allocate_random_bills (
            Wallet wallet,
            long long amount,
            int * num_allocated_coins,
            long long * allocated_amount)
```

Allocate coins from the wallet randomly until the desired amount is reached.

Parameters

wallet	The wallet containing the coins.
amount	The target amount to allocate.
num_allocated_coins	Pointer to store the number of allocated coins.
allocated_amount	Pointer to store the total allocated amount.

Returns

An array of allocated coins.

4.15.1.12 calculate_coin_score()

```
double calculate_coin_score (
            Coin * coin,
            long long currentTime,
            long long maxDenom,
            long long minDenom)
```

Calculate the score of a coin based on its expiration time and denomination.

Parameters

coin	The coin whose score is to be calculated.
currentTime	The current time in seconds.
maxDenom	The maximum denomination value.
minDenom	The minimum denomination value.

Returns

The calculated score of the coin.

4.15.1.13 calculate_renew_fee()

```
long long calculate_renew_fee (
            Wallet wallet,
            long long time)
```

Calculate the renew fee for the wallet based on the current time.

Parameters

wallet	The wallet containing the coins.
time	The current time in seconds.

Returns

The total renew fee for the wallet.

4.15.1.14 calculate_total_fee()

```
long long calculate_total_fee (
            Coin * coins,
            int num_coins,
            operation_type operation)
```

Calculate the total fee for a set of coins based on the operation type.

Parameters

coins	The array of coins.
num_coins	The number of coins.
operation	The operation type.

Returns

The total fee for the specified operation.

4.15.1.15 compare_coin_wrappers()

```
int compare_coin_wrappers (
            const void * a,
            const void * b)
```

Comparison function for sorting coin wrappers by score in descending order.

Parameters

a	Pointer to the first coin wrapper.
b	Pointer to the second coin wrapper.

Returns

An integer less than, equal to, or greater than zero if the first coin wrapper's score is less than, equal to, or greater than the second coin wrapper's score, respectively.

4.15.1.16 compare_coins_asc()

```
int compare_coins_asc (
            const void * a,
            const void * b)
```

Comparison function for sorting coins by denomination amount in ascending order.

Parameters

a	Pointer to the first coin.
b	Pointer to the second coin.

Returns

An integer less than, equal to, or greater than zero if the first coin's denomination amount is less than, equal to, or greater than the second coin's denomination amount, respectively.

4.15.1.17 compare_coins_desc()

```
int compare_coins_desc (
            const void * a,
            const void * b)
```

Comparison function for sorting coins by denomination amount in descending order.

Parameters

a	Pointer to the first coin.
b	Pointer to the second coin.

Returns

An integer less than, equal to, or greater than zero if the first coin's denomination amount is greater than, equal to, or less than the second coin's denomination amount, respectively.

4.15.1.18 compare_creation_time_asc()

```
int compare_creation_time_asc (
            const void * a,
            const void * b)
```

Comparison function for sorting coins by creation timestamp in ascending order.

Parameters

a	Pointer to the first coin.
b	Pointer to the second coin.

Returns

An integer less than, equal to, or greater than zero if the first coin's creation timestamp is less than, equal to, or greater than the second coin's creation timestamp, respectively.

4.15.1.19 compare_denomination_asc()

```
int compare_denomination_asc (
            const void * a,
            const void * b)
```

Comparison function for sorting denominations in ascending order.

Parameters

a	Pointer to the first denomination.
b	Pointer to the second denomination.

Returns

An integer less than, equal to, or greater than zero if the first denomination is less than, equal to, or greater than the second denomination, respectively.

4.15.1.20 compare_denomination_desc()

```
int compare_denomination_desc (
            const void * a,
            const void * b)
```

Comparison function for sorting coins in descending order by denomination amount.

Parameters

a	Pointer to the first coin.
b	Pointer to the second coin.

Returns

An integer less than, equal to, or greater than zero if the first coin's denomination amount is greater than, equal to, or less than the second coin's denomination amount, respectively.

4.15.1.21 compare_denomination_desc_ll()

```
int compare_denomination_desc_ll (
            const void * a,
            const void * b)
```

Comparison function for sorting denominations in descending order.

Parameters

a	Pointer to the first denomination.
b	Pointer to the second denomination.

Returns

An integer less than, equal to, or greater than zero if the first denomination is greater than, equal to, or less than the second denomination, respectively.

4.15.1.22 compare_expiry_time_amount()

```
int compare_expiry_time_amount (
            const void * a,
            const void * b)
```

Comparison function for sorting coins by expiration time, then by denomination amount.

Parameters

a	Pointer to the first coin.
b	Pointer to the second coin.

Returns

An integer less than, equal to, or greater than zero if the first coin's expiration time is less than, equal to, or greater than the second coin's expiration time, respectively. If the expiration times are equal, it compares by denomination amount.

4.15.1.23 compare_expiry_time_amount_reverse()

```
int compare_expiry_time_amount_reverse (
            const void * a,
            const void * b)
```

Comparison function for sorting coins by expiration time, then by denomination amount in reverse order.

Parameters

a	Pointer to the first coin.
b	Pointer to the second coin.

Returns

An integer less than, equal to, or greater than zero if the first coin's expiration time is less than, equal to, or greater than the second coin's expiration time, respectively. If the expiration times are equal, it compares by denomination amount in reverse order.

4.15.1.24 generate_withdraw_coins()

```
Coin * generate_withdraw_coins (
            long long amount,
            long long time,
            Wallet default_wallet,
            int * num_coins)
```

Generate coins for withdrawal from the wallet.

Parameters

wallet	The wallet containing the coins.
amount	The target amount to withdraw.
time	The current time in seconds.
default_wallet	The wallet containing the default coins for generation.
num_coins	Pointer to store the number of generated coins.

Returns

An array of generated coins.

4.15.1.25 remove_selected_coins()

```
void remove_selected_coins (
            Wallet * wallet,
            Coin * coins,
            int num_coins)
```

Remove selected coins from the wallet.

Parameters

wallet	Pointer to the wallet.
coins	The array of coins to be removed.
num_coins	The number of coins to be removed.

4.16 src/fee.c File Reference

```
#include "fee.h"
```

Functions

- long long calculate_fee (Coin coin, operation_type operation)

  Calculate the fee for a given coin and operation type.

4.16.1 Function Documentation

4.16.1.1 calculate_fee()

```
long long calculate_fee (
            Coin coin,
            operation_type operation)
```

Calculate the fee for a given coin and operation type.

Parameters

coin	The coin for which the fee is being calculated.
operation	The type of operation (e.g., DEPOSIT_OP, WITHDRAW_OP, REFUND_OP, REFRESH_OP).

Returns

The total fee in satoshis for the specified operation.

4.17 src/generator.c File Reference

```
#include "fee.h"
#include "user.h"
#include "coin_selection.h"
#include <time.h>
#include <stdlib.h>
#include <stdio.h>
```

Functions

- void generate_actions_for_user (User user, Action **actions, int *size)
  Generate actions for a given user based on their type.
- void generate_and_save_actions (const char *base_dir, int num_users)
  Generate actions for multiple users and save them to CSV files.

Variables

- const char * StrategyNames [ ]

4.17.1 Function Documentation

4.17.1.1 generate_actions_for_user()

```
void generate_actions_for_user (
            User user,
            Action ** actions,
            int * size)
```

Generate actions for a given user based on their type.

Parameters

user	The user for whom actions are being generated.
actions	Pointer to an array of actions to be generated.
size	Pointer to the size of the actions array.

4.17.1.2 generate_and_save_actions()

```
void generate_and_save_actions (
            const char * base_dir,
            int num_users)
```

Generate actions for multiple users and save them to CSV files.

Parameters

base_dir	The base directory where the CSV files will be saved.
num_users	The number of users for whom actions will be generated.

4.17.2 Variable Documentation

4.17.2.1 StrategyNames

```
const char* StrategyNames[]
```

Initial value:

```
= {
        "MAX_BILLS",
        "MIN_BILLS",
        "CLOSEST_TO_EXPIRE_MIN_BILLS",
        "CLOSEST_TO_EXPIRE_MAX_BILLS",
        "MAX_BILLS_TIME_TO_EXPIRE_WEIGHTED",
        "RANDOM",
        "EVEN_FROM_MIN_TO_MAX",
        "EVEN_FROM_MAX_TO_MIN",
        "GREEDY_MIN_TO_MAX"
}
```

4.18 src/main.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <string.h>
#include <pthread.h>
#include <semaphore.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/stat.h>
#include "fee.h"
#include "parser.h"
#include "user.h"
#include "common.h"
#include "generator.h"
#include "simulation.h"
#include "../src/cjson/cjson.h"
```

Macros

- #define MAX_THREADS 8
- #define USER_NAME_MAX_LENGTH 50
- #define SEMAPHORE_NAME "/semaphore"

Functions

- void update_progress ()

  Update the progress of file processing.
- int count_files_in_directory (const char *directory_path)

  Count the number of files in a directory.
- void clear_directory (const char *directory_path)

  Clear all files in a directory.
- void simulate_actions_from_file (const char *filepath, Wallet denomination_wallet)

  Simulate actions from a file for a given wallet.
- void * thread_function (void *arg)

  Thread function to simulate actions from a file.
- void load_and_simulate_actions (const char *base_dir, Wallet denomination_wallet)

  Load and simulate actions from all files in a directory.
- void check_and_prepare_directory (const char *dir_path)

  Check if a directory exists and create it if it does not. If it exists, clear its contents.
- int main (int argc, char *argv[])

Variables

- sem_t * semaphore
- int total_files = 0
- int processed_files = 0
- time_t start_time

4.18.1 Macro Definition Documentation

4.18.1.1 MAX_THREADS

```
#define MAX_THREADS 8
```

4.18.1.2 SEMAPHORE_NAME

```
#define SEMAPHORE_NAME "/semaphore"
```

4.18.1.3 USER_NAME_MAX_LENGTH

```
#define USER_NAME_MAX_LENGTH 50
```

4.18.2 Function Documentation

4.18.2.1 check_and_prepare_directory()

```
void check_and_prepare_directory (
            const char * dir_path)
```

Check if a directory exists and create it if it does not. If it exists, clear its contents.

Parameters

dir_path	The path to the directory.

4.18.2.2 clear_directory()

```
void clear_directory (
            const char * directory_path)
```

Clear all files in a directory.

Parameters

directory_path	The path to the directory to be cleared.

4.18.2.3 count_files_in_directory()

```
int count_files_in_directory (
            const char * directory_path)
```

Count the number of files in a directory.

Parameters

directory_path	The path to the directory.

Returns

The number of files in the directory.

4.18.2.4 load_and_simulate_actions()

```
void load_and_simulate_actions (
            const char * base_dir,
            Wallet denomination_wallet)
```

Load and simulate actions from all files in a directory.

Parameters

base_dir	The base directory containing the files.
denomination_wallet	The wallet with coin denominations.

4.18.2.5 main()

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

4.18.2.6 simulate_actions_from_file()

```
void simulate_actions_from_file (
            const char * filepath,
            Wallet denomination_wallet)
```

Simulate actions from a file for a given wallet.

Parameters

filepath	The path to the file containing actions.
denomination_wallet	The wallet with coin denominations.

4.18.2.7 thread_function()

```
void * thread_function (
            void * arg)
```

Thread function to simulate actions from a file.

Parameters

arg	The arguments for the thread, including the file path and wallet.

Returns

NULL

4.18.2.8 update_progress()

```
void update_progress ()
```

Update the progress of file processing.

4.18.3 Variable Documentation

4.18.3.1 processed_files

```
int processed_files = 0
```

4.18.3.2 semaphore

```
sem_t* semaphore
```

4.18.3.3 start_time

```
time_t start_time
```

4.18.3.4 total_files

```
int total_files = 0
```

4.19 src/parser.c File Reference

```
#include "parser.h"
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include "cjson/cjson.h"
```

Macros

- #define MAX_LINE_LENGTH 1024
- #define MAX_SECTIONS 100
- #define MAX_BLOCK_SIZE (1024 * 100)

Functions

- long long time_to_seconds (const char *time_str)

  Convert a time string to seconds.
- long long parse_number (const char *input)

  Parse a number from a string, removing any decimal points.
- char * trim_whitespace (char *str)

  Trim whitespace from the beginning and end of a string.
- char * parse_currency_name (const char *config_data)

  Parse the currency name from the [taler] block in the configuration file.
- Coin parse_coin_block (const char *coin_block_data, const char *currency_name)

  Parse a single coin denomination block and return a Coin structure.
- Wallet parse_wallet_config (const char *config_path)

  Parse the wallet configuration from a file.
- Wallet parse_wallet_config_json (const char *filename)

  Parse the JSON configuration for a wallet.

4.19.1 Macro Definition Documentation

4.19.1.1 MAX_BLOCK_SIZE

```
#define MAX_BLOCK_SIZE (1024 * 100)
```

4.19.1.2 MAX_LINE_LENGTH

```
#define MAX_LINE_LENGTH 1024
```

4.19.1.3 MAX_SECTIONS

```
#define MAX_SECTIONS 100
```

4.19.2 Function Documentation

4.19.2.1 parse_coin_block()

```
Coin parse_coin_block (
            const char * coin_block_data,
            const char * currency_name)
```

Parse a single coin denomination block and return a Coin structure.

Parameters

coin_block_data	The data for the coin block.
currency_name	The name of the currency.

Returns

The parsed Coin structure.

4.19.2.2 parse_currency_name()

```
char * parse_currency_name (
            const char * config_data)
```

Parse the currency name from the [taler] block in the configuration file.

Parameters

config_data	The configuration data as a string.

Returns

The parsed currency name, or NULL if not found.

4.19.2.3 parse_number()

```
long long parse_number (
            const char * input)
```

Parse a number from a string, removing any decimal points.

Parameters

input	The input string to parse.

Returns

The parsed number as a long long, or -1 on failure.

4.19.2.4 parse_wallet_config()

```
Wallet parse_wallet_config (
            const char * config_path)
```

Parse the wallet configuration from a file.

Parameters

config_path	The path to the configuration file.

Returns

The parsed Wallet structure.

4.19.2.5 parse_wallet_config_json()

```
Wallet parse_wallet_config_json (
            const char * filename)
```

Parse the JSON configuration for a wallet.

Parameters

filename	The path to the JSON configuration file.

Returns

The parsed Wallet structure.

4.19.2.6 time_to_seconds()

```
long long time_to_seconds (
            const char * time_str)
```

Convert a time string to seconds.

Parameters

time_str	The time string to convert.

Returns

The number of seconds represented by the time string, or -1 if the format is invalid.

4.19.2.7 trim_whitespace()

```
char * trim_whitespace (
            char * str)
```

Trim whitespace from the beginning and end of a string.

Parameters

str	The string to trim.

Returns

The trimmed string.

4.20 src/simulation.c File Reference

```
#include "simulation.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

Functions

- int get_scale (long long amount)

  Get the scale of a given amount.
- void simulate_user_actions (int user_index, User user, Wallet denomination_wallet, int num_actions, strategy strategy)

  Simulate user actions and write results to a file.

4.20.1 Function Documentation

4.20.1.1 get_scale()

```
int get_scale (
            long long amount)
```

Get the scale of a given amount.

Parameters

amount	The amount to determine the scale for.

Returns

The scale of the amount.

4.20.1.2 simulate_user_actions()

```
void simulate_user_actions (
            int user_index,
            User user,
            Wallet denomination_wallet,
            int num_actions,
            strategy strategy)
```

Simulate user actions and write results to a file.

Parameters

user_index	The index of the user.
user	The user structure containing user information.
denomination_wallet	The wallet containing denominations.
num_actions	The number of actions to simulate.
strategy	The strategy to use for coin allocation.

4.21 src/user.c File Reference

```
#include "user.h"
#include <stdlib.h>
#include <math.h>
```

Functions

- int rand_range (int min, int max)

  Generate a random integer between min and max, inclusive.
- long long rand_range_long (long long min, long long max)

  Generate a random long long integer between min and max, inclusive.
- long long generate_normal_ll (double mean, double stddev)

  Generate a random long long integer following a normal distribution.
- void generate_actions_for_student (Action **actions, int *size, int days)

  Generate actions for a student user type.
- void generate_actions_for_student_static (Action **actions, int *size, int days)

  Generate static actions for a student user type.
- void generate_actions_for_business_owner (Action **actions, int *size, int days)

  Generate actions for a business owner user type.
- void generate_actions_for_retired (Action **actions, int *size, int days)

  Generate actions for a retired user type.
- void generate_actions_for_family (Action **actions, int *size, int days)

  Generate actions for a family user type.
- void generate_actions_for_freelancer (Action **actions, int *size, int days)

  Generate actions for a freelancer user type.
- void generate_actions_for_teacher (Action **actions, int *size, int days)

  Generate actions for a teacher user type.
- void generate_actions_for_artist (Action **actions, int *size, int days)

  Generate actions for an artist user type.

4.21.1 Function Documentation

4.21.1.1 generate_actions_for_artist()

```
void generate_actions_for_artist (
            Action ** actions,
            int * size,
            int days)
```

Generate actions for an artist user type.

Parameters

actions	Pointer to the array of actions to be generated.
size	Pointer to the size of the actions array.
days	The number of days to simulate actions for.

4.21.1.2 generate_actions_for_business_owner()

```
void generate_actions_for_business_owner (
            Action ** actions,
            int * size,
            int days)
```

Generate actions for a business owner user type.

Parameters

actions	Pointer to the array of actions to be generated.
size	Pointer to the size of the actions array.
days	The number of days to simulate actions for.

4.21.1.3 generate_actions_for_family()

```
void generate_actions_for_family (
            Action ** actions,
            int * size,
            int days)
```

Generate actions for a family user type.

Parameters

actions	Pointer to the array of actions to be generated.
size	Pointer to the size of the actions array.
days	The number of days to simulate actions for.

4.21.1.4 generate_actions_for_freelancer()

```
void generate_actions_for_freelancer (
            Action ** actions,
            int * size,
            int days)
```

Generate actions for a freelancer user type.

Parameters

actions	Pointer to the array of actions to be generated.
size	Pointer to the size of the actions array.
days	The number of days to simulate actions for.

4.21.1.5 generate_actions_for_retired()

```
void generate_actions_for_retired (
            Action ** actions,
            int * size,
            int days)
```

Generate actions for a retired user type.

Parameters

actions	Pointer to the array of actions to be generated.
size	Pointer to the size of the actions array.
days	The number of days to simulate actions for.

4.21.1.6 generate_actions_for_student()

```
void generate_actions_for_student (
            Action ** actions,
            int * size,
            int days)
```

Generate actions for a student user type.

Parameters

actions	Pointer to the array of actions to be generated.
size	Pointer to the size of the actions array.
days	The number of days to simulate actions for.

4.21.1.7 generate_actions_for_student_static()

```
void generate_actions_for_student_static (
            Action ** actions,
            int * size,
            int days)
```

Generate static actions for a student user type.

Parameters

actions	Pointer to the array of actions to be generated.
size	Pointer to the size of the actions array.
days	The number of days to simulate actions for.

4.21.1.8 generate_actions_for_teacher()

```
void generate_actions_for_teacher (
            Action ** actions,
            int * size,
            int days)
```

Generate actions for a teacher user type.

Parameters

actions	Pointer to the array of actions to be generated.
size	Pointer to the size of the actions array.
days	The number of days to simulate actions for.

4.21.1.9 generate_normal_ll()

```
long long generate_normal_ll (
            double mean,
            double stddev)
```

Generate a random long long integer following a normal distribution.

Parameters

mean	The mean of the normal distribution.
stddev	The standard deviation of the normal distribution.

Returns

A random long long integer following a normal distribution.

4.21.1.10 rand_range()

```
int rand_range (
            int min,
            int max)
```

Generate a random integer between min and max, inclusive.

Parameters

min	The minimum value.
max	The maximum value.

Returns

A random integer between min and max.

4.21.1.11 rand_range_long()

```
long long rand_range_long (
            long long min,
            long long max)
```

Generate a random long long integer between min and max, inclusive.

Parameters

min	The minimum value.
max	The maximum value.

Returns

A random long long integer between min and max.

Index