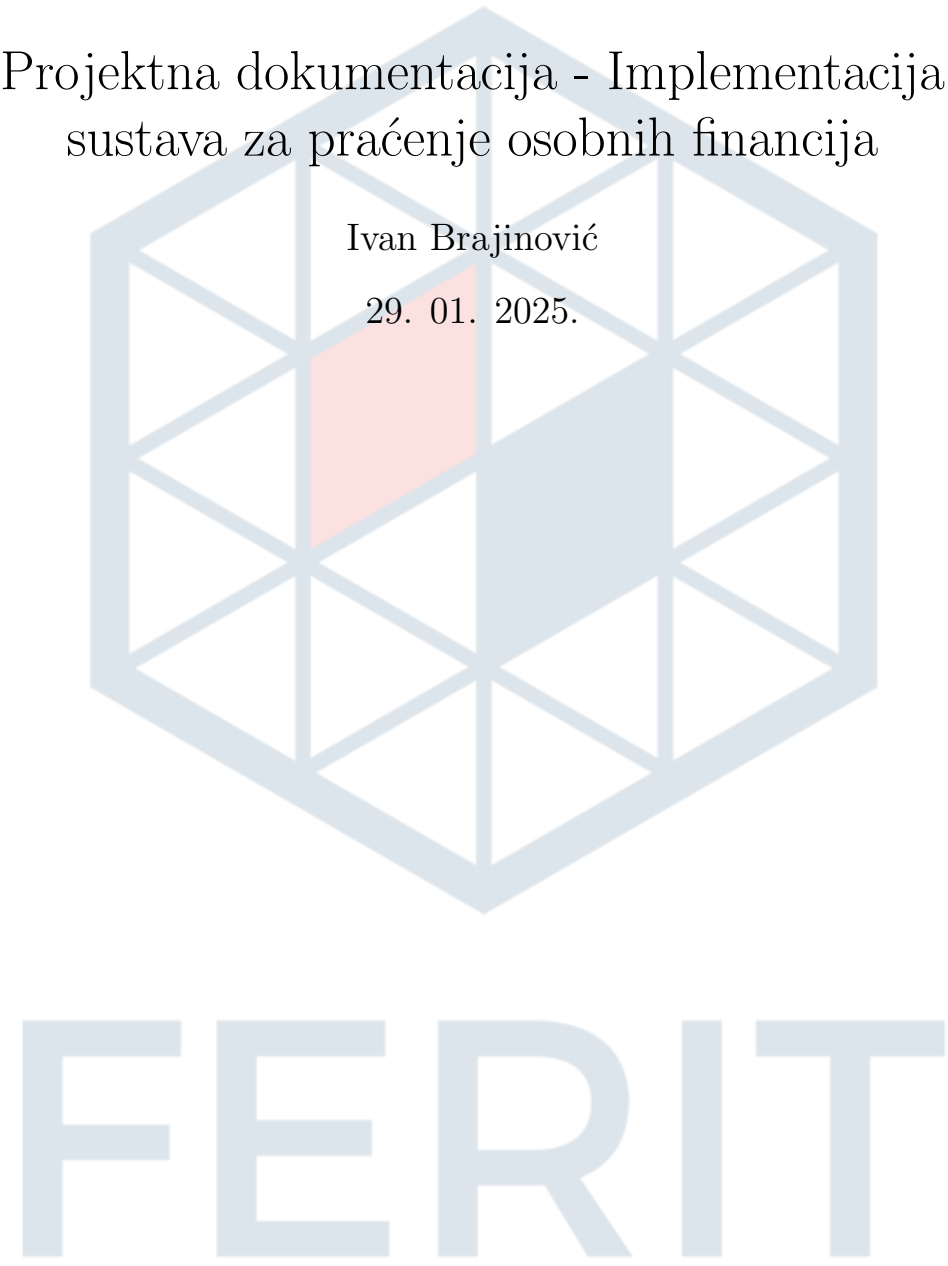
Projektna dokumentacija - Implementacija sustava za praćenje osobnih financija

Ivan Brajinović

29. 01. 2025.



FERIT

1 Uvod

U sklopu kolegija *Napredno programiranje*, je zadan zadatak *Implementacije sustava za praćenje osobnih financija*. Izgled programa, zahtjevi funkcionalnosti te implementacije su od samog početka zadani te su zapisani u *readme.md* datoteci te samo spomenuti u nastavku dokumenta. Problem koji se rješava ovom aplikacijom je praćenje osobnih transakcija s obzirom na datum uspostave transakcije te pripadne kategorije transakcije.

2 Opis projekta

2.0.1 Struktura podataka

Kao struktura za pohranu podataka se koristi stog. Nakon što su implementirane osnovne metode, kreirana je dinamička biblioteka .so nastavka. Izgled sučelja dinamičke biblioteke je prikazan pod dijelom *Zaglavne datoteke*, stack.h.

2.0.2 Glavna aplikacija

Glavna aplikacija rješava problem opisan u uvodu. Točni zahtjevi se vide u podnaslovu Funkcionalnosti. Korištena je već spomenuta dinamička biblioteka te struktura transakcije koja predstavlja jednu transakciju. Konkretan kod je predstavljen i opisan dalje u dokumentu.

2.1 Funkcionalnosti

- Dodavanje nove transakcije
- Poništavanje zadnje transakcije
- Pregled zadnje transakcije
- Ispis svih transakcija
- Dodatno - Pregled po kategorijama
- Dodatno - Mjesečni izvještaj troškova
- Dodatno - Pretraživanje transakcija (po danu i/ili mjesecu)

2.2 Datotečna struktura

Datotečna struktura projekta izgleda ovako:

```
Makefile
src
    include
        stack.h
        transaction.h
    stack.c
    main.c
bin
src_dynamic
    include
        stack.h
        transaction.h
    main.c
bin_dynamic
libstack.so
main.pdf
main.tex
images
```

Za prevođenje i pokretanje programa, kreiranje dinamičke biblioteke te generiranje PDF dokumentacije se koristi Makefile datoteka. Programski kod i zaglavne datoteke se nalaze u *src* i *src_dynamic* direktorijima. Izvršni program se nalazi u *bin* i *bin_dynamic* direktorijima. Direktoriji *src* i *src_dynamic* su identični. Razlika je u tome što *src* sadržava datoteku *stack.c* dok ne ista u direktoriju *src_dynamic* uklonjena kako se koristi dinamička biblioteka *libstack.so*. Isto se odnosi i na *bin* i *bin_dynamic*.

2.3 Direktorij src

2.4 Zaglavne datoteke

Zaglavne datoteke se nalaze unutar direktorija *include*. U ovom projektu smo napisali 2 zaglavne datoteke:

- stack.h
- transaction.h

Izlistaj navedenih zaglavnih datoteka slijedi u nastavku. Datoteke neće biti pojašnjavanje kako su komentari sami po sebi obavili taj posao. Uvjetno uključivanje datoteke i ostale pretprocesorske varijable neće biti prikazane.

2.4.1 transaction.h

```
enum transaction_category
{
  FOOD,
  UTILITIES,
  FUN,
  OTHER
};

struct transaction
{
  float total;
  enum transaction_category category;
  int day;
  int month;
  char description[50];
};
```

Zaglavna datoteka koja sadržava strukturu transakcije.

2.4.2 stack.h

```
struct stack
{
  struct transaction** transaction_history;
  int top;
  int max_size;
};

// create a stack instance
// allocate memory for the transaction_history
// set the head
// @ returns struct stack* if successful
// @ returns FAIL if not
struct stack* initialize(void);

// add a new transaction to the top
// increment top index counter
// @ returns SUCCESS if successful
// @ returns FAIL if not
int push(struct transaction* transaction, struct stack* stack);

// get a pointer to the transaction from the top and delete it from the stack
// set the top transaction to the second to last one by decrementing the
// stack->top counter value
// @ returns struct transaction* if successful
// @ returns FAIL if not
struct transaction* pop(struct stack*);

// get a pointer to the transaction from the top
// @ returns struct transaction* if successful
// @ return FAIL if not
struct transaction* peek(struct stack*);

// check if the stack is empty
// @ returns SUCCESS if stack is empty
// @ returns FAIL if not
int isEmpty(struct stack* stack);

// check if the stack is full
// @ returns SUCCESS if stack is full
// @ returns FAIL if not
int isFull(struct stack* stack);
```

```c
// prints transaction
void printTransaction(struct transaction* transaction);

// print all contents of the stack
void printContent(struct stack*);

// print all transactions that fit inside the given category
// @ param stack containing transactions
// @ param category is the category we want to print all transactions of
void printContentByCategory(struct stack* stack, int category);
```

Zaglavlje dinamičke biblioteke, opisuje sučelje za korištenje navedene biblioteke. Čitava biblioteka ovisi o zaglavnoj datoteci transaction.h.

2.5 Glavni program - main.c

Datoteka *main.c* se sastoji od 3 funkcije:

- printMenu()
- printWelcomeScreen()
- main()

2.5.1 printMenu()

Ova funkcija služi kako bi se ispisao izbornik programa. Ne sadrži ništa posebno gledao logike izvršavanja te neće biti posebno objašnjavana.

2.5.2 printWelcomeScreen()

Ova funkcija služi za ispis početke poruke prilikom prvog pokretanja programa. Kao i funkcija *printMenu()*, ne sadrži nikakvu posebnu logiku te neće biti detaljnije opisivana.

2.5.3 main()

```c
// structure elements
float total;
int category;
int day;
int month;
char description[50];

struct stack* stack = initialize();
if (stack == NULL)
{
  printf("Stack initialization failed!\n");
  printf("Exiting program!");
```

```
    return FAIL;
  }

  // array used to store the user input
  char user_input[100];
  // variable used for the menu navigation
  char* menu_index = (char*)malloc(sizeof(char) * 2);
  // high impedence, default value
  if (menu_index == NULL)
  {
    printf("Failed to allocate memory for menu_index!\n");
    return FAIL;
  }
```

Početni dio koda koji sadrži deklaracije i definicije varijabli koje će biti korištene tijekom rada programa. Ovdje možemo odmah vidjeti varijable namijenjene za popunjavanje strukture transakcija kao i varijablu *stack* koja predstavlja strukturu podataka koju smo implementirali u projektu. Pored njih, imamo i dvije varijable koje koristimo za pohranu korisničkog unosa, bilo u svrhu navigacije izbornikom ili općenito. Dodatno, za svaki slučaj dinamičke alokacije memorije, provjerava se uspješnost te radnje.

```
#if DEBUG == 1
struct transaction* transaction0 = (struct transaction*)malloc(sizeof(struct transaction));
transaction0->total = 10.0;
transaction0->category = 0;
transaction0->day = 1;
transaction0->month = 1;
strcpy(transaction0->description, "Lunch.");

if (push(transaction0, stack) == FAIL)
{
  printf("Failed to push transaction to stack!\n");
} else
{
}
#endif
```

Kao pomoć u ispitivanju rada programa su ručno napravljene transakcije (njih 10) od kojih je samo jedna prikazana kako bi zadržali preglednost čitavog dokumenta. Ostalih 9 izgleda potpuno isto tako da nema potrebe da se sve prikazuju. Nakon što su transakcije kreirane, dodane su na stog pomoću funkcije *push*. Ove ispitne transakcije se mogu isključiti iz programa tako da se unutar zaglavne datoteke *stack.h* vrijednosti konstante *DEBUG* postavi u 0 (nula).

```
printWelcomeScreen();
while(1)
```

Nakon ispitnih transakcija ispisujemo početnu poruku programa te ulazimo u beskonačnu petlju gdje će program provesti ostatak vremena.

```
do
{
  printMenu();
  fgets(menu_index, 2, stdin);
```

```
    while(getchar() != '\n');
    // 0 - 48
    // 7 - 55
    // c - 99
  // if the input is outside of the allowed range, repeat the entry
  } while (!(menu_index[0] >= 48 && (menu_index[0] <= 55) || (menu_index[0] == 99)));
```

Generična funkcija korisničkog unosa podataka. Svaka od njih ima neku poruku na početku koja korisnika navodi na očekivani unos. Nakon toga se obrađuje korisnički unos tako da se miču neželjeni znakovi ili se unos koji je znak pretvara u broj (cijeli ili decimalan) s pomoću funkcija *atoi* i *atof*. U ovom primjeru te dvije funkcije nisu korištene kako je moguće na osnovu ovog primjera iz koda zaključiti kako se radi o korisničkom unosu. Na kraju vidimo određene uvijete kojima zapravo zadajemo dozvoljeni korisnički unos.

```
  if (menu_index[0] == '0')
  {
    ... unos korisničkih podataka ...

    other_transaction = (struct transaction*)malloc(sizeof(struct transaction));
    other_transaction->total = total;
    other_transaction->category = category;
    other_transaction->day = day;
    other_transaction->month = month;
    strcpy(other_transaction->description, description);

    if (push(other_transaction, stack) == FAIL)
    {
      printf("Failed to push transaction to stack!\n");
    } else
    {
    }
  }
```

Slučaj kada se stvara nova transakcija, korisnički unos 0 (nula). Od korisnika se traži unos za svaki od elemenata strukture transakcije te se nakon unosa ista pohranjuje u memoriju. Nakon što se pohrani, pokazivač na strukturu se pohrani u stog.

```
else if (menu_index[0] == '1')
{
  printf(" Chosen option: 1 - delete last transaction.\n");
  pop(stack);
}
```

U slučaju kada je korisnik unio 1 (jedan), zadnja unesena struktura se briše i miče sa stoga.

```
else if (menu_index[0] == '2')
{
  printf(" Chosen option: 2 - check last transaction.\n");
  printTransaction(peek(stack));
}
```

U slučaju da je korisnik unio 2 (dva), ispisuje se sadržaj zadnje unesene strukture.

```
else if (menu_index[0] == '3')
{
  printf(" Chosen option: 3 - transaction history overview.\n");
  printContent(stack);

}
```

U slučaju da je korisnik unio 3 (tri), ispisuje se sadržaj svih transakcija.

```
else if (menu_index[0] == '4')
{
  do
  {
    printf(" Chosen option: 4 - browse by category.\n");
    printf("Enter search categoriy: \n");
    printf(" 0 - FOOD\n");
    printf(" 1 - UTILITIES\n");
    printf(" 2 - FUN\n");
    printf(" 3 - OTHER\n");

    // read the input as a string
    fgets(user_input, 100, stdin);
    while(getchar() != '\n');
    category = atoi(user_input);
    // atof converts string to float if valid input
    // otherwise it returns 0.0
  }while(!(category >= 0 && category <= 3));
  // print all transactions with that categoriy
  printContentByCategory(stack, category);

}
```

U slučaju da je korisnik unio 4 (četiri), transakcije se ispisuju s obzirom na pripadajuću kategoriju. Funkcija *printContentByCategoriy* sadrži kod u kojem se prolazi kroz sve transakcije čiji pokazivači se nalaze na stogu. Za svaku transakciju se provjerava da li pripada ta odabranoj kategoriji, te ako pripada, sadržaj transakcije se ispisuje.

```
else if (menu_index[0] == '5')
{
  int month = 0;
  double sum_fun = 0.0;
  double sum_utilities = 0.0;
  double sum_food = 0.0;
  double sum_other = 0.0;

  printf(" Chosen option: 5 - monthly cost summary.\n");
  do
  {
    printf(" Enter a number between 1 and 12 representing the month of interest: \n");

    // read the input as a string
    fgets(user_input, 100, stdin);

    // atof converts string to float if valid input
    // otherwise it returns 0.0
    month = atoi(user_input);

  }while(!(month >= 1 && month <= 12));

  int index = 0;
  // search through all of the transactions
  // sum the costs by category for the given month
  while ( index < stack->top)
  {
    if ( stack->transaction_history[index]->month == month)
    {
      int cost_category = stack->transaction_history[index]->category;
```

```
      if (  cost_category == 0)
      {
        sum_food += stack->transaction_history[index]->total;
      } else if ( cost_category == 1)
      {
        sum_utilities += stack->transaction_history[index]->total;
      } else if (cost_category == 2)
      {
        sum_fun += stack->transaction_history[index]->total;
      } else
      {
        sum_other += stack->transaction_history[index]->total;
      }
    }
    index = index + 1;
  }
  printf(" Cost summary for the month: %d\n", month);
  printf("\tCategory FOOD: %.2f\n", sum_food);
  printf("\tCategory UTILITIES: %.2f\n", sum_utilities);
  printf("\tCategory FUN: %.2f\n", sum_fun);
  printf("\tCategory OTHER: %.2f\n\n\n", sum_other);
  // based on the user input (months)
  // calculate the sum of costs for each category
}
```

U slučaju da je korisnik unio 5 (pet), vrši se mjesečni obračun po kategorijama za odabrani mjesec. Prolazi se kroz svaku od transakcija te se gleda pripada li odabranom mjesecu. Ako pripada, brojač za pripadajuću kategoriju se uvećava za iznos transakcije. Na kraju slijedi ispis svih brojača.

```
else if (menu_index[0] == '6')
{
  int browse_option;
  int day;
  int month;

  do
  {
    // the user should be able to browse the transactions
    // based on day (1-31) and/or month (12)
    // select one of the two options
    printf(" Chosen option: 6 - Date based transaction browser.\n");
    printf(" Enter 0 if you want to browse transactions based on the day.\n");
    printf(" Enter 1 if you want to browse transactions based on the day and month.\n");
    // read the input as a string
    fgets(user_input, 100, stdin);
    browse_option = atoi(user_input);
    // atof converts string to float if valid input
    // otherwise it returns 0.0
  }while(!(browse_option >= 0 && browse_option <= 1));

  if ( browse_option == 0)
  {
    do
    {
      printf(" Chosen option: Browse by day!\n");
      printf(" Enter a number between 1 and 31.\n");

      // read the input as a string
      fgets(user_input, 100, stdin);
      day = atoi(user_input);
      // atof converts string to float if valid input
      // otherwise it returns 0.0
    }while(!(day >= 1 && day <= 31));

    printf("Search criteria: \n"),
    printf("\tDay: %d\n\n", day);

    int index = 0;
    while ( index <= stack->top )
    {
      if (stack->transaction_history[index]->day == day)
```

```
      {
        printTransaction(stack->transaction_history[index]);
      }
      index = index + 1;
    }
  } else
  {
    do
    {
      printf(" Chosen option: Browse by day and month!\n");
      printf(" Enter a number between 1 and 31.\n");
      // read the input as a string
      fgets(user_input, 100, stdin);
      day = atoi(user_input);
      // atof converts string to float if valid input
      // otherwise it returns 0.0
    }while(!(day >= 1 && day <= 31));

    do
    {
      printf(" Enter a number between 1 and 12.\n");
      // read the input as a string
      fgets(user_input, 100, stdin);
      month = atoi(user_input);
      // atof converts string to float if valid input
      // otherwise it returns 0.0
    }while(!(month >= 1 && month <= 12));

    printf("Search criteria: \n"),
    printf("\tDay: %d\n", day);
    printf("\tMonth: %d\n\n", month);

    int index = 0;
    while ( index <= stack->top )
    {
      if (stack->transaction_history[index]->day == day && stack->transaction_history[index]->month == month)
      {
        printTransaction(stack->transaction_history[index]);
      }
      index = index + 1;
    }

  }
}
```

U slučaju da je korisnik unio 6 (šest), ispisuju se transakcije s obzirom na dan i/ili mjesec stvaranja. Ako se unese 0 (nula), transakcije se ispisuju s obzirom na dan koji je unesen (ponovo novi unos, prvo korisnik unese 6, pa 0, pa sad dan). Ako se unese 1 (jedan), transakcije se ispisuju s obzirom na dan i mjesec koji je unesen.

```
else if (menu_index[0] == '7')
{
  break;
}
```

U slučaju da korisnik unese 7 (sedam), program prekida s radom.

```
else if (menu_index[0] == 'c')
{
  printf("Press \"c\" to print out the menu.");
  printMenu();
}
```

U slučaju da korisnik unese c (slovo c), ispisuje se izbornik.

```
printf("To show the menu, press \"c\".\n");
// reset input
menu_index[0] = 'z';
```

Na kraju beskonačne petlje se ispisuje prikazana poruka i varijabla *menu_index* se postavi na vrijednost slova z kao neaktivno stanje.

3 Direktorij bin

U ovom direktoriju se nalaze izvršne datoteke koje se stvore korištenjem naredbi iz *Makefile* datoteke.

4 Instalacija i pokretanje

4.1 Preuzimanje projekta

Projekt se nalazi na adresi https://github.com/Brajinovic/napredno. Isti je potrebno preuzeti, sa grane *master*. Npr. s pomoću *Git* naredbe:

```
git clone https://github.com/Brajinovic/napredno.git
```

Nakon što je program preuzet, potrebno je otvoriti naredbedbeni redak u preuzetom direktoriju. U nastavku su opisane dvije opcije pokretanja aplikacije. Za *Linux* operacijske sustave nisu potrebne nikakve dodatne instalacije kako se koristi *GNU gcc* prevoditelj koji dođe već instaliran.

Za prevođenje i pokretanje programa bez dinamičke biblioteke (koriste se src i bin direktoriji):

```
make compile
```

Za prevođenje i pokretanje programa s dinamičkom bibliotekom (koriste se src_dynamic i bin_dynamic direktoriji):

```
make dynamic_lib
export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH
make compile_dynamic
```

5 Tehnički detalji

5.1 Opis podataka

Osnovni podatak je transakcija. Ovaj podatak je opisan strukturom *transaction* koja je definirana u datoteci transaction.h. Transakcije se pohranjuju u strukture podataka koje se zovu stogovi. Stog je definiran u dinamičkoj biblioteci *libstack.so* čije se sučelje može vidjeti u zaglavnoj datoteci.

5.2 Glavne funkcije

Glavna funkcija *main()* se koristi kao pokretač i korisnik svih ostalih funkcija. Tu ponajprije spada funkcionalnost izbornika koja sama po sebi ne ovisi o drugim funkcijama, međutim funkcionalnosti koje se kroz izbornik pružaju koriste funkcije definirane u biblioteci *libstack.so*.

6 Zaključak

Kroz ovaj kratki projekt je predstavljen koncept rješavanja problema praćenja osobnih transakcija. Korištena je općenita struktura stoga kao medij za pohranu korisničkih podataka, koji je samostalno implementiran te se nisu razmatrale druge mogućnosti i rješenja. Ovo rješenje pruža pohranu, pregled i brisanje transakcija ali ne na optimalan način. Prvi korak ka poboljšanju bi bilo odabir neke pogodnije strukture podataka za pohranu kao što su na primjer *skip liste* koje omogućavaju brži način pretrage transakcija. Također nema nikakav mehanizam sigurnosne pohrane, tako da ako slučajno izađemo iz aplikacije, nema nikakvih mogućnosti povratka unesenih informacija tako da bi trebalo unesene podatke pohraniti u neku bazu podataka te tako riješiti taj nedostatak. Sve u svemu, ovaj projekt služi kao zgodan početni zadatak kako bi se primijenile stvari spomenute na predavanjima li se svakako nalazi daleko od gotovog komercijalnog proizvoda.