

---

# Analyse af overvægt, Ernæringshjælpemidler og udvikling af et alternativ

---

Project Report  
Group B2-24

Aalborg University  
Det Teknisk Naturvidenskabelige Fakultet  
Skibbrogade 5  
DK 9000 Aalborg



**AALBORG UNIVERSITY**  
STUDENT REPORT

Det Teknisk Naturvidenskabelige  
Fakultet  
Skibbrogade 5  
DK-9000 Aalborg  
<http://cs.aau.dk>

**Title:**

Analyse af overvægt, Ernæringshjælpe midler og udvikling af et alternativ

**Theme:**

Ernæring og IT

**Project Period:**

P1 1. Semester

**Project Group:**

B2-24

**Participant(s):**

Nikolaj Ljørring  
Kevin Wolff  
Benjamin Jhaf Madsen  
Jacob Sloth Thomsen  
Rasmus Jespersen  
Christian Holmgaard  
Tobias Klitgaard

**Supervisor(s):**

Nis Anders Bornø  
Carla Smink

**Copies:** 5

**Page Numbers:** 69

**Date of Completion:**

December 18, 2015

**Abstract:**

With a percentage of 17.5%, obesity is on the rise in Denmark. The need for dietary guidance has never been higher, and the obesity epidemic continues to grow. As software engineering students, this presents a unique opportunity to analyze the problem with an IT-perspective. This report will analyze, obesity in the Danish population, and select programs that are already offering advice on dietary needs, to begin the process of developing a program, that can offer an alternative to modern dietary and weight loss programs. In an attempt to keep, the population motivated in their struggle for a lower weight line.



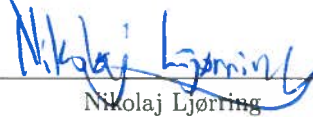
Jacob Thomsen  
<jsth15@student.aau.dk>



Kevin Wolff  
<kwolff15@student.aau.dk>



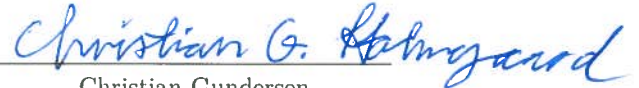
Rasmus Jespersen  
<rjespe15@student.aau.dk>



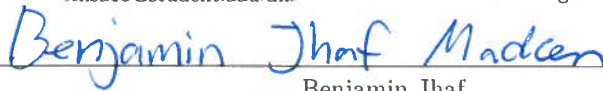
Nikolaj Ljørring  
<nljarr15@student.aau.dk>



Tobias Klitgaard  
<tksa15@student.aau.dk>



Christian Gundersen  
<cholmg15@student.aau.dk>



Benjamin Jhaif  
<bjma15@student.aau.dk>

## Preface

### Terminology

**De 10 kostråd** De 10 kostråd er en samling af råd til den danske befolkning som bliver udgivet af Sundhedsstyrelsen. Rådene skal hjælpe danskerne med at få et overordnet bedre indtag af næringsstoffer, og leve et sundere liv. Rådene er primært baseret på ønsket om at øge sundheden i befolkningen. De 10 kostråd, er som følger:

- Spis varieret, ikke for meget og vær aktiv.
- Spis frugt, og mange grøntsager.
- Spis mere fisk.
- Vælg fuldkorn.
- Vælg magert kød og pålæg.
- Vælg magre mejeriprodukter.
- Spis mindre mættet fedt.
- Spis mad med mindre salt.
- Spis mindre sukker
- Drik vand.

## Contents

<b>Preface</b>	<b>iii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Forord . . . . .	1
1.2 Introduktion . . . . .	1
<b>2 Problemanalyse</b>	<b>3</b>
2.1 Definition af overvægt . . . . .	3
2.2 Kostinformationer, forskning og offentlige myndigheder . . . . .	4
2.3 Sociale normer og madkultur . . . . .	5
2.4 Ernæring og fysisk aktivitet . . . . .	6
2.5 Delkonklusion . . . . .	7
2.6 Afgrænsning . . . . .	7
2.7 Teknologianalyse . . . . .	7
2.7.1 Kalorieberegner . . . . .	8
2.7.2 Madplanlæggere . . . . .	8
2.7.3 AI Coaches . . . . .	9
2.7.4 Fora . . . . .	9
2.8 Undersøgte programmer i detaljer . . . . .	9
2.8.1 Kalorieberegner . . . . .	9
2.8.2 Madplaner . . . . .	10
2.8.3 Fora . . . . .	11
2.8.4 Coaching applikationer . . . . .	11
2.9 Delkonklusion . . . . .	13
<b>3 Evaluering</b>	<b>14</b>
3.1 Design . . . . .	14
3.1.1 Success Kriterier . . . . .	15
3.2 Implementering . . . . .	16
3.2.1 Profil opsætning . . . . .	16
3.2.2 Programmets hovedmenu . . . . .	21
3.2.3 Hjælpesiden . . . . .	22

3.2.4	Databaseopsætning . . . . .	27
3.2.5	Fil indlæser og indkøbsliste sortering . . . . .	29
3.2.6	Algoritmen som vælger retter til brugeren . . . . .	35
3.3	Program udførsel . . . . .	41
3.4	Vurdering . . . . .	42
3.4.1	Fejl og mangler i programmet . . . . .	42
3.5	Konklusion . . . . .	46
3.6	Perspektivering . . . . .	47
<b>Bibliography</b>		<b>48</b>
<b>A Appendix</b>		<b>51</b>

## 1.1 Forord

Rapporten er skrevet med henblik på kronologisk læsning, med del konklusioner der leder ind til de næstkommende kapitler. Den brugte reference metode er Vancouver, denne anvendes pga. sin logiske nummereringsform.

## 1.2 Introduktion

Overvægt er i stigende grad blevet et problem igennem det seneste årti, der har opnået epidemisk niveau.[17] Det er en stor belastning for det danske samfund, hvor en undersøgelse lavet af Svendsen OL, Heitmann BL, Mikkelsen KL, Raben A, Rytting KR, Sørensen TIA et al. 2011 Om Fedme i Danmark, viser at man estimerer at 8% af de samlede omkostninger på sundhedssektoren, går til behandlingen af svær overvægt. Kombineres dette med de faktuelle tal fra danske regioner [20] kan vi udregne et total på 8,24 mia. kr. årligt. Blot ved behandlingen af sygdommen. Dette er et minimumstal for omkostningerne ved overvægt, da der ikke er medregnet arbejdsmarkedsomkostninger, evt. førtidspension og medicin omkostningerne for individet.

Dette giver et naturligt økonomisk incitament for at gennemgå en analyse af problemet overvægt, der kan være med til at skabe en evt. løsning på det økonomiske pres der bliver lagt på staten.

Samtidig, er antallet af overvægtige i Danmark stigende, [23] hvilket hypotetisk set, kan resultere i større udgifter til behandling af overvægt i fremtiden.

Udover de økonomiske omstændigheder der inddrager staten i omtanken om Fedmeepidemien, ligger de overvægtige med individuelle sociologiske og fysiske problemer, der bør tages højde for. Sociale normer som mænd der ikke spiser salat pga. en frygt for at blive stemplet som feminin [8], og en manglende forståelse af de 10 kostråd [10], kan hypotetisk set være skyld i at mange individer helt undgår at forsøge sig med et vægttab. Samtidig har overvægt en række fysiske konsekvenser, heriblandt øget risiko for kræft og diabetes 2, der

begge kan nedsætte individets livskvaliteten [23].

Samtidig er der de normalvægtige, der ligger lidt på sidelinjen i forhold til rapportens indhold, men disse skal dog nævnes for sammenligning. Disse individer har ikke en enlig interesse i vægttab, udover at de kan være pårørende til et individ med vægt problemer. Hovedårsagen til at inddrage dem i rapporten, er at de har en mulig interesse i at bibeholde deres nuværende vægt og dette kan få indflydelse på rapportens resultater.

I rapporten vil der blive taget udgangspunkt i Body Mass Index (BMI). Dette gøres på baggrund af at BMI, er bredt anvendt til at måle individers overvægtsniveau, og derfor anvendes i mange af de undersøgelser som denne rapport er bygget op om. Imidlertid, er BMI også et forfejlet system som ikke kan overføres direkte til alle dele af verden [4]. Systemet kan dog korrigeres til forskellige lande.

Rapporten vil undersøge i hvordan IT-programmer, kan hjælpe med at forebygge og behandle moderat til klasse I overvægt, som set i 2.1, og hvordan nuværende systemer kan forbedres til at imødekomme individets behov for en bedre ernæringsbalance, uden samtidig at tage unødvendig tid ud af deres dagligdag til behandling af deres overvægt, eller forebyggelsen heraf. Derfor har vi valgt det initierende problem:

*"Hvordan kan individets ernæring ændres, uden at lægge et unødvendigt tidskrav på individets dagligdag?"*

Som følger af at individer med moderat til klasse I svær overvægt, lægger et økonomisk pres på staten, i form af sundhedsudgifter. Vil der blive bearbejdet en analyse af problemerne bag overvægten. Dette er til formål for at give indsigt ind i individets behov og motivationer for at påbegynde et vægttab. Endvidere vil normalvægtige blive inddraget i analysen, med henblik på at undersøge mulighederne for at bibeholde sin vægt. De individuelle behov fra normal- og overvægtige vil blive benyttet til at analysere forskellige programmer, som ud fra deres bruger base, popularitet på IOS og Android og innovation, vil blive vurderet med henblik på at finde en evt. bedre løsning.



## Problemanalyse

Ernæring er et bredt, og meget udefineret emne, der strækker sig fra hvad vi indtager på daglig basis, til de politiske holdninger vi har til vores fødevarer. I denne rapport, tager vi udgangspunkt i ernæring som dagligt indtag hos tre forskellige hovedgrupper af interessenter moderat overvægtige, svært overvægtige og normalvægtige. Samtidig baserer rapporten også på statslige og kommercielle interessenter, som har en økonomisk interesse i vores tre interessent grupper. Samlet set, vil dette give et billede af problematikker og interesser indenfor feltet ernæring med henblik på vægttab.

### 2.1 Definition af overvægt

Det er først og fremmest vigtigt at forstå, at individer med vægtproblemer er inddelt i forskellige grupper [23], da ikke alle vægtproblemer indeholder de samme risiko faktorer [23], er dette en logisk inddeling når der tales om helbredsrisiko ved evt. vægt problemer. Inddelingerne er som vist i 2.1 [23]

WHO klassifikation	Alternativ benævnelse	BMI ( $\text{kg}/\text{m}^2$ )	Helbredsrisiko
Undervægt		$< 18,5$	Afhænger af årsagen til undervægten
Normalvægt		$18,5 - 24,9$	Normal
Overvægt		$\geq 25,0$	
Moderat overvægt		$25,0 - 29,9$	Let øget
Svær overvægt		$\geq 30,0$	Middel til meget øget
Klasse 1	Fedme	$30,0 - 34,9$	Middel øget
Klasse 2	Svær fedme	$35,0 - 39,9$	Kraftigt øget
Klasse 3	Ekstrem svær fedme	$\geq 40,0$	Ekstremt øget

Table 2.1: Tabel over vægtklassifikationer.

Som det ses på 2.1, er der mange inddelinger efter vægt. Denne rapport tager udgangspunkt i normal- til klasse I svært overvægtige. Dette gøres fordi individer med klasse II - III svær overvægt, hypotetisk set har behov for en mere

specialiseret kost, eller kirurgisk indgreb for at ændre på deres vægt. Dette ligger udenfor rapportens fagområde, og den egentlige idé bag rapporten om vægttab, hvor vi vil se hvordan IT kan åbenbare en ændring i ernæringen hos individer med overvægt.

## 2.2 Kostinformationer, forskning og offentlige myndigheder

Definitionen af interessenterne til rapporten, vil nu give os mulighed for at bearbejde og analysere på hvordan information bliver givet til det enkelte individ med vægtproblemer. Her er særligt de offentlige myndigheder, der bør være en frontløber i bekæmpelsen af fedmeepidemien, sat i fokus for at analysere hvorfor fedmeepidemien stadig er stigende, på trods af at befolkningen er givet oplysninger som de 10 kostråd [25].

Sundhedsstyrelsen er en gren af staten, der beskæftiger sig med fremmelsen af befolkningens sundhed og sygdomsbehandling. Sundhedsstyrelsen udgiver let tilgængelige informationer om kost, overvægt og livsstilsforbedringer. Informationerne herfra bygger på teorien "små skridt", hvis grundlag ligger i at lave små ernærings og aktivitets ændringer, der kan give en større effekt hvis de bibeholdes over længere tid. [26]

Denne information kan ifølge sundhedsstyrelsen bruges til at ændre radikalt på individets overvægt, hvor et eksempel er en ændring fra at spise en pose slik, til at spise en halv pose slik hver aften. Kan opnå en vægt nedgang på 11.8kg på en periode på et år. [22]

Ifølge en gruppe forskere bestående af, Astrup et al, Bitz cand. Science og I. A. Sørensen prof 2012, er dette ikke tilfældet, og at sundhedsstyrelsens råd blot bidrager til stigmatisering af overvægtige. [22]

Et eksempel er posen med slik, der ifølge Astrup et al, Bitz cand. Science og I. A. Sørensen prof 2012 er et urealistisk punkt. Ved den førnævnte besparelse på en halv pose slik. Vil individet spare 300kcal om dagen. Dette er forudsat at individet ikke indtager noget for at supplere de 300kcal som kroppen ellers er vant til. Men selv i bedste tilfælde kan det ifølge forskerne ikke blive til mere end 2,6kg på et år. Samtidig fortæller Astrup et al, Bitz cand. Science og I. A. Sørensen prof 2012, at kroppens forbrænding og energikrav bliver mindre når overskydende vægt tabes, bl.a. fordi kroppen bliver lettere og derfor ikke kræver så meget energi at holde aktiv. [22]

Samtidig konstatere en undersøgelse [30] om myter om ernæring fra 2013, at "små skridt" metodens grundlag er forfældet, og der refereres til at grundlaget for "små skridt" metoden, bygger på undersøgelser der er mere end et halvt århundrede år gamle. [30]

Herved analyseres det at der er forskellige holdninger til ernæring, iblandt de førnævnte forskere og hos de offentlige myndigheder. Da de offentlige myn-

digheder får deres oplysninger fra div. forskere, må det altså konstateres at de offentlige myndigheder ikke har mulighed for at verificere og implementere den nyeste forskning hurtigt og effektivt nok til at holde de givne informationer opdateret med den nyeste forskning.

Dette kan hypotetisk set skabe forvirring hos de individer der søger den professionelle hjælp til at imødekomme en sundere ernæring, dog er der ikke pt. Lavet undersøgelser der påviser denne hypotese. Samtidig peger dette også på at der er en større debat om hvad "sund kost" enlig er, og hvordan man bedst taber de uønskede kilo, hvilket også kan påvirke den førnævnte forvirring omkring sundere ernæring.

## 2.3 Sociale normer og madkultur

Overvægtiges situation afhænger, som tidligere nævnt, ikke kun af individets diæt, men i ligeså stor grad af individets sociale forhold og omgivelser.

Det betyder at de overvægtiges situation, kan styrkes eller svækkes igennem en stigmatisering, eller en afstigmatisering af deres tilstand. De sociale normer der forefindes i Danmark, kan ifølge Astrup et al 2007, resultere i mobning og udelukkelse, samt diskrimination ved jobsøgning og uddannelse af voksne og børn der lider af overvægt eller fedme. Dette kan ifølge Astrup et al 2007, ende med at blive en psykisk invaliderende tilstand, hvor man lider af depression der leder til trøstespisning. [16]

Samtidig, Ifølge Holm et al 2009, er der også en rangering af madvarer i Danmark, hvor madvarer er rangeret efter et madhierarki, hvor bøffer ligger som den absolutte vinder. Samtidig er forskellige kostgrupper forbundet til et maskulint eller feminint stigma. Hvor det, ifølge Holm et al. 2009, kan forekomme at individer bliver udsat for sociale repressalier hvis man bryder dette mønster. Fx er det normalt for en mand at drikke sig meget beruset i byen, hvor det er mindre velset hvis en kvinde gør det samme. Modsat er det normalt for en kvinde at spise meget frugt og grønt, hvor en mand ifølge sociale standarder foretrækker bøffer med sovs og kartofler.

Herved siger Holm et al også at mænd, i forhold til ernæring har nogle udfordringer i form af disse stigma. [8] Derved kan der ifølge Holm et al. 2009, være problemer med at ændre på sine ernæringsvaner, fordi madvarer har et bestemt hierarki, som vist i 2.2, og dermed en bestemt status i forhold til individets valg af madprodukter, og fordi madkulturen har et maskulint eller feminint stigma over sig, [8]

Dette kan være skyld i at det bliver svære at undslippe den psykisk invaliderende tilstand, da individets sociale omgangskreds kan være med til hypotetisk at forværre deres situation. Dette gør situationen endnu svære for de moderat til svært overvægtige mennesker der står med vægt problemer, og gerne vil ændre på deres situation. [11]

Rangordning af måltider	Rangordning og kombination af fødevarer
1. Middag/aftensmad	Kødet navngiver hovedretten Grøntsager Kartofler (erstatte kornprodukter)
2. Frokost	Pålægsprodukter navngiver <del>maden</del> Grøntsags- oste, ægge-, fiske- eller kødprodukter Mindre synlig del: rugbrød
3. Morgenmad	Kornprodukter navngiver retten og udgør den dominerende bestanddel

Table 2.2: Rangordning af måltider.

## 2.4 Ernæring og fysisk aktivitet

En livstilsændring betyder at individet ændrer store dele af sin måde at leve på. Dette indeholder blandt andet også individets spisevaner og mængden af motion personen dyrker. Disse livstilsændringer kan opnås igennem mange forskellige metoder, heriblandt kan fitness skabe et incitament for at bibeholde et vægttab. Ifølge en undersøgelse om forholdet imellem motion og ernæring udført af Universitetet i Rio de Janeiro om "Energy metabolism during the post-exercise recovery in men" [3] bibeholdte, 50% af de deltagere der både dyrkede motion og ændrede kostplan, deres vægt efter endt forsøg. Hvor imod 100% af deltagerne der kun fik ændret kostplan, gik tilbage til deres gamle kostvaner og bibeholdte ikke deres vægttab. [15] Undersøgelser [15] viser at individers viden vedrørende sund ernæring, ikke nødvendigvis hænger sammen med motionsdyrkelse. Dette medfører at individet ikke ændrer ernæringsvaner selv ved regelmæssig motion, og hellere bibeholder sine mindre sunde ernæringsvaner selv ved forsøg på vægttab. Dette er på trods af at undersøgelser viser at ernæringsmæssigt korrekte vaner, i kombination med regelmæssig motion giver de bedste resultater for vægttab og motionsudbytte. [15] Ernæring har, ifølge en undersøgelse lavet af Instituttet for social medicin, ved Universitetet i Rio de Janeiro om "Energy metabolism during the postexercise recovery in men" [3], en udpræget indvirkning på individets motions præstationer. Med den rigtige ernæring optimeres personens præstationsniveau og formindskes restitutionstiden, samtidig med at det forbedrer individets generelle sundhed. [3] Dermed kan en sund ernæring, der er afbalanceret til personens træningsform give bedre resultater end hvis ernæringen ikke bliver medtaget som en faktor i individets fysiske aktivitet. [29]

Individer der dyrker regelmæssig motion, har ikke nødvendigvis en større viden om sund ernæring. Undersøgelser viser dog at personer der dyrker regelmæssig motion, over tid, opnår en ændring i deres attitude, som medfører en ændring i deres kostvaner. [7]

For personer i alle aldre, bør fordelingen af de tre kost hovedgrupperne (fedt, kulhydrater og protein) ske balanceret. Hvor fedt, kulhydrat og protein udgør henholdsvis 25-40%, 45-60% og 10-20% af personens daglige indtag. Forskellen i procenterne udgøres af individets daglige aktivitets niveau. [5] Hvor en person der styrketræner (træning med hensigt om at opbygge større muskelmasse) vil have brug for et meget højt indtag af protein, for at opretholde

og opbygge muskelmasse.[6]

## 2.5 Delkonklusion

Ud fra analysen af offentlige myndigheder og forskere, kan vi konkludere at hvor informationerne er til rådighed, og i mange situationer let tilgængelige, bliver de ikke brugt af den brede befolkning. Dette kan hypotetisk set skyldes at informationerne er svære at forstå, eller at man som vist i de sociologiske afsnit, vælger ikke at benytte sig af informationerne, fordi de ville være til for stor ulejlighed for individet. Dette giver en stor udfordring i form af hvordan kostinformationer og råd bør formidles til de individuelle interessent grupper.

Samtidig rejser det også spørgsmålet ”bør individet informeres om sund kost, for at skabe en ernæringsmæssig ændring?”, dette spørgsmål er relevant, da informationerne til at leve et sundere liv er tilgængelige, men ikke bliver benyttet, og derved ville information om sund ernæring, hypotetisk set, ikke have den ønskede effekt om at hjælpe individer med overvægt til at ændre deres ernæringsvaner, her ville en fastlagt plan med korrekt ernæring muligvis give bedre resultater, da individet så ikke skal læse og forstå svær information om ernæring.

Samtidig bør motion kombineres med en sundere ernæring, som påvist i afsnittet om ernæring og fysisk aktivitet. Det vil hjælpe individerne med at bibeholde deres vægttab efter endt diæt, og samtidig giver det en større succes rate, end hvis man vælger kun at lave ernæringsmæssige ændringer til individets livsstil.

## 2.6 Afgrænsning

Ud fra analysen fortsætter rapporten med moderat til klasse I svær overvægtige som interessenter. Da disse udgør 47,4% af den danske befolkning (19), er der tale om en større sundhedsmæssig problemstilling i det danske samfund. Samtidig vil en ændring i antallet af overvægtige i Danmark også bidrage til en lettelse af statens udgifter til behandlingen af disse interessent grupper, herved drager staten altså også nytte af at man tager disse interessenter.

Da informationer om sund kost og ernæring, er meget forskellige og ikke lige til at definere, vil der som nævnt i delkonklusionen, ikke være incitament for at arbejde videre på at udbyde information, men hellere en struktureret planlægning af ernæringsændringer.

## 2.7 Teknologianalyse

For at kunne skabe et overblik over de forskellige teknologiske løsningsmodeller, har vi valgt at inddele disse løsningsmodeller i fire forskellige kategorier, som vist i 2.3, som repræsenterer de individuelle programmer. Dette bliver gjort for at skabe overblik over de forskellige løsningsmodeller der er til stede pt.

Samtidig er der valgt nogle specifikke programmer, som eksempler på løsningsmodeller på de enkelte programtyper. Disse er valgt ud fra antal down-

Programtype	Specifikke valgte programmer
Kalorieberegner	Madlog Supertracker
Madplanlæggere	Fitness Meal Planner 8Fit
AI Coaches	Noom Strava
Fora	Motion-online Iform

Table 2.3: Programtyper og valgte programmer.

loades og/eller bruger base. Dette er gjort fordi populariteten af et program hypotetisk set, viser kategorien fra den bedst mulige vinkel. Samtidig er der valgt to forskellige programmer indenfor hver programtype, for at etablere et sammenligningsgrundlag.

Disse programtyper og specifikke programmer henvender sig til forskellige problemstillinger, fx kan kalorieberegnerne gøre det muligt at optimere individuel ernæring. Men hvis individet hellere vil have en nemmere løsning til madlavningen og dermed spare tid på planlægningen, så kan individet gøre brug af en madplanlægger der opstiller en ugentlig madplan.

Der findes også fora hvor individet kan stille spørgsmål og få svar fra individer i samme situation eller individer med mere erfaring. Individet kan med forummet skabe en social omgangskreds med individer der har samme problem, mere erfaringer og ideer til netop deres problem. De individer der kan have svært ved at motivere sig selv til f.eks. træning ville kunne gøre brug af en AI coach, som ville kunne fungere som en personlig coach eller træner. Disse teknologier vil være repræsenteret, og defineret som følger.

### 2.7.1 Kalorieberegner

Kalorieberegnerne specialiserer sig i at udregne det anbefalede daglige indtag af kalorier til deres brugere. Disse er ofte sammenkoblet med et bogføringssystem hvor individer kan bogføre de mål, motions og spise vaner som individet så kan hente tilbage, og benytte til at udregne en lang række data. I denne rapport vil vi koncentrere os om to forskellige kalorieberegner, baseret på deres brugerbase.

- SuperTracker
- Madlog

### 2.7.2 Madplanlæggere

Kategorien madplanlæggere strækker sig over en lang række programmer, som alle har tilfælles at de kan hjælpe individet med at sammensætte en madplan. Nogle af disse programmer har støtte programmer til motion, kalorieberegning og sociale medier. I denne rapport vil vi dog koncentrere os om programmerne.

- Fitness Meal Planner
- 8Fit

### 2.7.3 AI Coaches

AI Coaches er normalvis tiltænkt til fitness træning, dog er der nogle af dem der indeholder en madplanlægger, som kalkulere og laver en madplan, baseret på din daglige fysiske aktivitet. Da der er mange forskellige AI Coaches, vælger vi at tage udgangspunkt i.

- Noom
- Strava

### 2.7.4 Fora

Fora består af sociale grupperinger der deler informationer om ernæring, sport og tips til vægttab. Disse emner er ret udbredte, og derfor eksisterer der også mange forskellige fora. Her i rapporten vil vi derfor tage udgangspunkt i fora.

- Motion-Online
- Iform

## 2.8 Undersøgte programmer i detaljer

### 2.8.1 Kalorieberegnerne

En kalorieberegner er et program der kan bruges til at beregne individets daglige kalorie indtag. I dette afsnit bliver sådanne kalorieberegner analyseret.

Individer med overvægt, kan benytte sig af kalorieberegnerne for at imødekomme en bedre balance af energi indtag og energi forbrug. Kalorieberegnerne som det danske Madlog [12] og det amerikanske SuperTracker [27], er gode bud på løsninger af en sådan programtype.

Programmer forholder sig hovedsageligt til en balance imellem ernæring og motion, hvor begge programmer har en indbygget mad database, der kan indtastes som et datasæt og beregne individets daglige indtag, og give et estimeret tal for dagligt forbrug.

For at fuldføre et datasæt, skal brugeren indtaste vægtnål på samtlige kalorieindtag i løbet af dagen. Samt det daglige aktivitets niveau, der defineres i forskellige typer motion. Denne data anvendes som før nævnt til et estimeret tal om dagligt energi forbrug. Dette giver anledning til en del bruger relateret registrering af div. Datatyper.

Madlog skriver om sig selv "Danmarks største og mest komplette fødevarer-database". Dette kan skyldes en funktion hvor individet der anvender madlog,

har mulighed for at indtaste nye mad informationer, der derefter bliver registreret i madlogs database over madvarer. [13]

Hertil har SuperTracker ikke en direkte sammenlignelig funktion. Her anvendes en indrapporteringsfunktion, hvor man kan sende navnet på en madvare, som opdateres på periodisk basis. [28]

Programmerne kan begge anvendes til at øge, bibeholde eller tabe vægt. Samtidig kan de anvendes til at supplere som kalorieberegner til en professionelt lagt kost og motionsplan. Dette giver programmerne meget fleksibilitet i forhold til de forskellige kost og motions former.

Ifølge gruppen som har testet programmerne, er programmernes store svaghed at der benyttes en stor mængde tid på at registrere individets daglige aktivitetsniveau og næringsindtag, som vist i 2.4, sammen med programmernes generelle fordele.

Program	Mad-database	Motionsplanlægger	Coach function	Informationsforum	Madplanlægger	Dagligt tidsforbrug
Madlog	X	X				20 -30 min
SuperTracker	X			X		20 -30 min

Table 2.4: Tidsforbrug af kalorieberegnerne.

## 2.8.2 Madplaner

En anden metode for overvægtige at ændre deres ernæringsvaner, er ved at anvende en madplanlægger. Der hvor disse applikationer adskiller sig fra kalorieberegnerne er ved at præsentere en fuld madplan for individet, med denne plan kan individet holde styr på kalorierne på forhånd, og forsøge at imødekomme daglige mål om kalorie indtag, uden at skulle registrere lige så mange oplysninger som ved en kalorieberegner. I dette afsnit bliver der analyseret nogle af disse madplanlæggere.

Applikationer som 8Fit [1] og Fitness Meal Planner [2], anvender sig af indtastede bruger data som, igennem en algoritme, planlægger en kostplan for individet, der er tilpasset individets personlige fitness og ernæringsmål.

8Fit og Fitness Meal Planner, anvender sig derfor ikke af registreringen af datasæt fra enkelte måltider, men anvender datasæt om individet, og individets mål. Datasættene for individet er højde, vægt, alder og køn. Individernes mål bliver håndteret forskelligt af de to applikationer, hvor 8Fit opstiller en række forskellige kure, som individet kan anvende til at nå sine mål. Her benytter Fitness Meal Planner, sig af indtastede data for fx kulhydrater, protein og fedt.

Samtidig, tilbyder applikationerne indkøbslister, som er lavet på baggrund af de forskellige retter applikationerne foreslår. Samtidig har 8Fit en motionsplanlægger, der opstiller en motionsplan med intense træningssessioner, der har indstillingsmuligheder til tilpasning af træningstider.

Udbyttet af disse applikationer er at de ikke har det samme tidskrav til individet, som kalorieberegnerne har. Dette giver individet mulighed for at



benytte tiden på andet end at registrere data i applikationerne.

Program	Mad database	Motionsplanlægger	Coach function	Informationsforum	Madplanlægger	Dagligt tidsforbrug
Fitness Meal Planner		X			X	5 - 10 min hver uge
8Fit		X			X	2 - 7 min pr. dag

Table 2.5: Tidsforbrug af madplaner.

### 2.8.3 Fora

Fora udgør en unik mulighed for individer med let til svær overvægt. Her kan disse individer søge information om div. ernærings, motions og væggtabs relaterede emner, når de har tid, og i mange situationer søge hjælp ved professionelle, uden at skulle betale for den samme service, som de ville skulle i en analog butik. Derfor analysere vi videre på hovedkategorien, og supplere med nogle specifikke eksempler, som henvender sig til vores interessenter, let til svært overvægtige.

Madplaner og kalorieberegner har ikke store mængder tilgængelig information om individuelle erfaringer med kost og motionsplaner. Dette kan anskaffes igennem forskellige fora, over alt på internettet. Fora som Iform [9] og motion-online [14], er eksempler på sådanne fora, der specialiserer sig i at formidle information om korrekt ernæring og motionsvaner. Samtidig med at individer kan kreere oplæg med egne erfaringer og evt. forslag til andre individer på disse fora.

Iform og Motion-online, udlægger et forum med forskellige hovedkategorier, heriblandt: Løb, kost og træning og sund vægt. Under disse individuelle hovedkategorier, kan individer oplægge deres erfaringer, og få feedback fra andre brugere eller professionelle, der dertil også kan oplægge deres viden om det enkelte emne.

Styrken i disse fora er den sociale interaktion som de forskellige individer har med hinanden. Dette kan være med til at bidrage til en succes i form af bibeholdelse af gode ernærings og motionsvaner. Svaghederne ligger i den misinformation der kan være til stede på sådanne fora. Dette kan samtidig give anledning til problemer med selv samme ernærings og motionsvaner. Samtidig er der ingen garanti for et svar på individets spørgsmål inde på forummet.

Program	Mad-database	Motionsplanlægger	Coach function	Informationsforum	Madplanlægger	Dagligt tidsforbrug
Motion-Online				X		Ikke relevant
Iform				X		Ikke relevant

Table 2.6: Tidsforbrug af fora.

### 2.8.4 Coaching applikationer

Coaching applikationer er et forholdsvist nyt fænomen. Her forsøger udgiverne at samle madplaner, kalorieberegner og en personlig træner i en applikation til

en mobil enhed. Dette er særligt interessant, da denne applikation forsøger at tage de sociale forhold med ind i designet af applikationer til ernæring og motion.

Udover de individuelle råd fra professionelle og amatører fra fora, forefindes AI coaches, som anvender en algoritme til at motivere individet til at fortsætte nogle individuelle mål.

AI coaches bygger på mange af de samme ideer fra madplanlæggere. Applikationerne adskiller sig fra madplanlæggere ved at have den førnævnte AI coach indbygget i applikationen. Denne reagerer, igennem algoritmer, på de forskellige mål og giver individet feedback og forslag til forbedring af individets motions og ernæringsplaner.

Applikationen Noom [19], registrerer et stort datasæt om individet, heriblandt: Højde, vægt, alder, køn, spise- og motionsvaner samt attitude til de førnævnte vaner, og forsøger at medberegne dette i en kost og motionsplan. Kost og motionsplanen bliver konstrueret ud fra SMART-modellen. Denne model er tidligere nævnt i rapporten, som "små skridt" modellen.

Hertil bliver en kost og motionsplan konstrueret med små individuelle skridt, som beregnes ud fra individets datasæt, dette datasæt anvendes i en kalorieberegner. Denne kalorieberegner, beregner herefter, på baggrund af en database som er lavet af brugere, USDA, MyNetDiary og FatSecret analyseret af Noom, hvad du har behov for af dagligt indtag. Individet registrerer herefter hvilke energi indtag og energi forbrug individet har haft i løbet af en dag. Dette beregnes løbende, og AI coachen kommer med forslag til hvad individet skal spise mere eller mindre af, og om, der er behov for at øge individets motionsplan.

Applikationen, kan ikke ses som en erstatning til en professionel træner, men som et supplement til det. Dette er applikationens styrke, og dets svaghed ligger i mængden af datasæt der er krævet for at applikationen kan registrere det daglige kalorieindtag og kalorieforbrug.

Herunder kan der introduceres en anden form for digital coaching i form af Strava applikationen. [24] Der vil blive fokuseret primært på dets egenskab til at interagere med sociale medier, som en del af dets overordnede interface.

Strava er en udviklet kilometertæller som man kender fra cykelcomputere. Den holder styr på hvor man har løbet eller cyklet, hvornår og hvor hurtigt man gjorde det. Applikationen synkroniserer via GPS tracking og stiller brugerens personlige tider automatisk op mod andres tider på samme rute. Strava er i sig selv næsten et helt socialt netværk da man kan kigge på andre brugeres tider og sammenligne med sin egen. Man kan få stillet et digitalt kort op der viser alle bruger registrerede ruter i brugerens omegn og man kan udfordre de hurtigste tider. Derfra kommer det konkurrerende aspekt og som fremlagt tidligere virker det yderst engagerende for brugeren. [21]

Applikationens brugeroverflade er sat op til at man kan vælge om man er ude at cykle eller løbe med et enkelt tryk. Den viser derefter de registrerede ruter og så er det op til brugeren om hvilken en man vil udfordre andre og sig

selv på. Man har mulighed for at opdatere sine bedrifter igennem andre sociale sites, såsom Facebook.

Applikationen fungerer som en anden form for digital coach da man kun interagerer med andre mennesker.

Program	Mad-database	Motionsplanlægger	Coach function	Informationsforum	Madplanlægger	Dagligt tidsforbrug
Noom	X	X	X	X		10 min setup
Strava		X	X			10 min setup

**Table 2.7:** Tidsforbrug af coaching applikationer.

## 2.9 Delkonklusion

Interessenterne moderat til svær overvægtig klasse I, har som påvist i inter-essent analysen, mange forskellige problemstillinger, som udgør en risiko for enten deres helbred, motivation eller sociale forhold. Dette kan behjælpes ved at give individerne med moderat til svært overvægtige klasse I, nogle redskaber til at håndtere et vægttab, uden at redskaberne giver anledning til et fald i motivation.

Teknologianalysen heri viser, at der er mange redskaber som allerede hen-vender sig til bibeholdning af motivationen hos individer der forsøger sig med et vægttab. Heriblandt kan AI coaches og fora nævnes som holdbare løsninger på at bistå individet i at bibeholde deres motivation, fordi henholdsvis fora og AI coaches har en interaktion med individet, og særligt med fora, er der mulighed for at opbygge sig en social relation til andre individer med samme problemstilling, og dermed opnå godkendelse hos andre.

Modsat fald er madplanlæggere og kalorieberegnerne bedre til at overvåge og informere brugeren om deres kalorie indtag. Disse har dog ikke samme interaktion som fora og AI coaches, og lider derfor under den manglende sociale bekræftelse som der forefindes hos fora og AI coaches.

Herved kan det konstateres at madplanlæggere og kalorieberegnerne kan drage mest nytte af at minimere den tid der skal bruges på at anvende programtype-rne, for bedre at kunne holde individet motiveret til at bruge dem. Kan der skabes en løsning hvor datasættene fra programtyperne bibeholdes, og mæng-den af tid krævet ved anvendelse af programmerne minimeres. Dette kan skabe incitament for holde brugeren motiveret, ved ikke at tage store tidsmængder ud af deres dagligdag, men ved at gøre deres dagligdag lettere.

Dette vil samtidig også gøre programtyperne interessante for normalvægtige, der ved at bruge programmet kan spare tid, samtidig med at de opnår en for-nuftig ernæringsbalance, der holder dem på deres nuværende vægt. Herved kan vi formulere følgende problemformulering:

*Hvordan kan madplaner og kalorieberegnerne kombineres, så de ikke tager tid ud af dagligdagen og bibeholder motivationen for et vægttab?*

**Figure 2.1:** Problemformulering

### 3.1 Design

Ud fra problemformulering, designes der et program, der kan imødekomme kravet om tidsbesparing. Dette gøres da evidensen der ligger til grund for problemformuleringen, fortæller os at individer med overvægt kan miste motivationen, hvis programmer tager for meget tid at anvende.

Denne udfordring kan hypotetisk set håndteres ved at anvende brugerens kcal, som essensen bag en madplansudregning. Dette vil fjerne tidskravene, der fx, er tillagt anvendelsen af Madlog.dk, hvor brugeren skal bruge 10-20 minutter dagligt på at anvende programmet, med intastning af forskellige ingredienser og afvejning af disse, før bogføring. Samtidig kan der implimenteres en indkøbsliste funktion, som samler og udskriver en indkøbsliste til brugeren og derved spare brugeren for de nødvendige udregninger der normalt er forbundet med indkøb til større madplaner. Som sidste led i tidsbesparelsen, kan der implimenteres en funktion, der gør opskrifter og ingrediensliste tilgængelig for brugeren. Denne giver brugeren den nødvendige viden til at udføre de forskellige opskrifter, og bespare dem tiden det vil tage at søge igennem fx. en kokebog for at finde opskriften.

For at kunne begynde på at udarbejde disse tidsbesparende funktioner, er der visse andre funktioner der er nødvendige at udarbejde først. Programmet skal kunne udregne brugeren kcal. behov. Dette kan gøres ved at designe en algoritme der tager udgangspunkt i kcal. behovet for en person i respirator. Med dette som udgangspunkt kan aktivitetsniveau, vægt, højde, alder og køn tilføjes til beregningen. Dette skulle hypotetisk set, lave en kcal. beregner der kan give os et estimat af hvad brugerens daglige kcal. behov er.

Samtidig skal der oprettes en opskrifts database, som indeholder opskrifter på henholdsvis morgen, middag og aftensmadsretter. Disse skal opstilles intuitivt, med en identifierer så retterne kan findes frem mod forlang. Opskrifterne, skrives sammen med deres respektive ingredienser, så disse kan hentes samtidig. Samtidig laves der en database og neutriner, som kobles sammen med identifieren, og derved skaber en kobling imellem opskrifterne og deres respektive kalorieindhold. Samtidig skal opskrifterne skrives i et portionsformat, så det er muligt at skalere dem til brugerens individuelle behov. Dette gør det

muligt at designe opskrifterne omkring brugerens specifikke behov til en madplan, og begrænser samtidig det nødvendige antal af opskrifter påkrævet før en madplan med variation kan opnås.

Ud over de førnævnte funktioner, bør programmet også indeholde forskellige *"utility"* funktioner, der kan give brugeren adgang til de forskellige informationer om deres madplan, og deres bruger data. Så disse kan ændres igennem brugerens anvendelses forløb. For at dette kan lade sig gøre, skal programmet først og fremmest kunne gemme en profil med brugerens data, der senere kan redigeres. Dette kan gøre via en database, hvor brugerens oplysninger gemmes, og hentes igen, til der er brug for dataen. En funktion til oprettelse af bruger data, skal samtidig indeholde spørgsmål om brugerens fysiske udformning og aktivitetsniveau. Disse data anvendes til at udregne brugerens kcal. behov, senere i anvendelsen af programmet. Brugeren skal samtidig have adgang til at kunne ændre i deres oplysninger, så brugeren kan redigere fx. deres vægt, og algoritmen opdateres, så der kommer nye krav til madplanen. Dette gør at madplanen ændre sig dynamisk sammen med brugerens specifikke behov.

Alle disse funktioner skal samles i en menu funktion, for at gøre det nemt for brugeren at finde rundt i programmet. Funktionen skal kunne køre imens brugeren har programmet aktiveret. Samtidig skal menuen kunne finde og printe de forskellige færdigudregnede informationer til brugeren, så brugeren kan anvende dem i dagligdagen.

### 3.1.1 Success Kriterier

Ud fra designet, kan vi opstille nogle specifikke success kriterier, som programmet skal kunne opfylde for at imødekomme designet. Success kriterierne indeholder kun det bare minimum af datastrukturen der er nødvendig for at fuldføre programmet.

- Programmet skal indeholde en kalorieberegner, dette er for at kunne udregne hvor mange kalorier det enkelte individ der benytter sig af programmet skal have. Dette ligger samtidig base for hvordan en evt. madplan skal udarbejdes af programmet.
- Programmet skal indeholde en madplanlægger, som skal kunne vælge retter ud fra en database, og hvor mange kalorier det enkelte individ skal have på daglig basis.
- Programmet skal kunne dette, ud fra brugerens fysiske fremtoning og aktivitetsniveau. Uden at skulle bruge daglige inputs fra brugeren og dermed spare brugeren tid ved håndtering af programmet.
- Samtidig skal programmet kunne lave en indkøbsliste over de forskellige retter, som skal indeholde en samlet mængde af råvarer for madplans perioden, gældende over 7 dage. Dette vil potentielt spare brugeren for tid, ved at forberede en hurtig indkøbsliste, som så kan anvendes i div. butikker.
- Programmet skal kunne indstilles til at kalkulere med henblik på vægttab, altså en nedgang i kalorie indtag. Dette skal udregnes igennem kalorie udregneren, og brugeren skal periodisk indtaste nye vægttal (periodisk hver uge).

## 3.2 Implementering

I dette afsnit er der programmeret et C program som løsningen til problemformuleringen. Der vil blive argumenteret undervejs i implementeringsafsnittet for valg af programmets opsætning. Programmet består af en kalorie beregner og en madplan. Det er programmeret således at brugeren får mindre tidspild når der indtastes brugerdata og fordi brugeren ikke nødvendigvis skal benytte programmet dagligt.

### 3.2.1 Profil opsætning

Til profil funktionen laves der et globalt struct som tilgås flere steder i programmet. For at undgå at skulle have mere end højst nødvendigt i main, er structured ud af programmets main og oprettet globalt. Der bruges strings til både fornavn, efternavn og køn, de er alle sat til længden 50 således der er allokeret nok plads til individer med lange for- eller efternavne.

```

23 struct profile_info
24 {
25     char first_name[MAX_STRING_LGT];
26     char last_name[MAX_STRING_LGT];
27     char gender[MAX_STRING_LGT];
28     int age;
29     int height;
30     double weight;
31     double calorie_need;
32     int protein_need;
33     int carbon_hydrates_need;
34     int fat_need;
35     int activity_level;
36     int nutrition_choice;
37 }user_profile;

```

Figure 3.1: Struct til profil

gender skal dog skrives som enten m eller f, for male eller female. Dette gøres for at spare tid på input for brugeren og indeholder den samme information som at skrive det fulde køn. age er en integer da det ikke er nødvendigt at have som decimalt tal. Ligeledes er højden en int, hvor der bedes om et tal i centimeter som alment anvendte i Danmark. Denne information, indgår som en del af beregningerne af kcal. indtag. Til sidst er vægten af

brugeren en double. Dette gøres fordi vægt ønskes i decimaltal.

```

142 void new_or_old_profile()
143 {
144     char new_or_old_profile_answer;
145     printf("Hey there! and welcome to the new and improved meal planner 1.0.0\n");
146     printf("Would you already have a profile set up and are ready to go? Y/N: ");
147     scanf(" %c", &new_or_old_profile_answer);
148     if((new_or_old_profile_answer == 'y' || new_or_old_profile_answer == 'Y' || new_or_old_profile_answer == 'n' || new_or_old_profile_answer == 'N'))
149     {
150         print_error("INPUT ERROR!", "NEW OR OLD PROFILE", "IRRELEVANT");
151     }
152     else
153     {
154         switch(new_or_old_profile_answer)
155         {
156             case 'y': case 'Y':
157                 user_profile_loader();
158                 break;
159             case 'n': case 'N':
160                 new_user_basic_input();
161                 break;
162         }
163     }
164 }
165 }
166 }

```

Figure 3.2: Udsnit af funktionen New or old profile

New\_user\_basic\_input er en funktion der spørger brugeren, som ønsker at lave en ny profil, om deres fysiske fremtoning, aktivitetsniveau og navn. På

figuren 3.3 er der taget et udsnit af funktionen `New_user_basic_input`. funktionen anvender `printf` og `scanf` som henholdsvis udskriver og indlæser brugerens respons. Samtidig indeholder funktionen en fail safe, der forhindrer brugeren i at skrive ukorrekte oplysninger. Denne køres i en `while` løkke, indtil brugeren indtaster korrekte oplysninger. `New_user_basic_input` slutter med at kalde funktionen `profile_saver` som set i figur 3.4.

```

171 void New_user_basic_input()
172 {
173     int i = 1;
174     printf("Welcome to the new user account creation\n");
175
176     printf("Please give us your name\n");
177     scanf("%s", user_profile.first_name);
178
179     printf("Please give us your surname\n");
180     scanf("%s", user_profile.last_name);
181
182     printf("What is your gender? M/F\n");
183     scanf("%s", user_profile.gender);
184
185     while((i))
186     {
187         if((strcmp(user_profile.gender, "M") || (strcmp(user_profile.gender, "W") || (strcmp(user_profile.gender, "L") || (strcmp(user_profile.gender, "I"))
188         {
189             i = 1;
190
191         }
192         else if(strcmp(user_profile.gender, "M") != 0 || strcmp(user_profile.gender, "W") != 0 || strcmp(user_profile.gender, "L") != 0 || strcmp(user_profile.gender
193         {
194             printf("Error: Please give us your gender again. M/F\n");
195             printf("%s", user_profile.gender);
196         }
197     }
198 }

```

Figure 3.3: Udsnit af funktionen New user basic input

I funktionen `profile_saver` bruges `FILE *` også, denne laver en file pointer, den kaldes her `file_pointer`, som er en stream. Dette skal bruges til at åbne en tekst fil som indeholder brugerens profildata, dette er en basal form for database, den bruges derefter sammen med `stdlib` funktionen `fopen`, som åbner en fil og bruger file pointeren som et stream til filen. Filen åbnes i `w+`, som betyder at vi både kan skrive til, eller læse fra filen, hvis filen allerede eksisterer bliver den overskrevet med de nye informationer som brugeren har indtastet.

Herefter bruges der `fprintf`, som er fil versionen af `printf`. Denne skriver informationerne over i filen fra standard input funktionen. Til sidst lukkes filen, per god orden, for at undgå læsningsfejl, og tillade andre dele af programmet adgang til databasen.

```

278 void profile_saver()
279 {
280     FILE *file_pointer;
281
282     file_pointer = fopen("test.txt", "w+");
283     if(file_pointer == NULL)
284     {
285         printf("FILE NOT FOUND", "PROFILE_SAVER", "FATAL");
286     }
287
288     fprintf(file_pointer, "%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n",
289             user_profile.first_name, user_profile.last_name, user_profile.gender, user_profile.age, user_profile.height,
290             user_profile.weight, user_profile.calorie_need, user_profile.protein_need, user_profile.carbon_hydrates_need,
291             user_profile.fat_need, user_profile.activity_level, user_profile.nutrition_choice);
292
293     fclose(file_pointer);
294 }
295

```

Figure 3.4: Udsnit af funktionen Profile saver

I funktionen `user_profile_loader`, hentes der data fra filen og indlæses i structet `user_profile`, hvor informationerne ligger tilgængelig til databehandling, skulle det blive nødvendigt. (ses i figur 3.5) Først tages der igen brug af `FILE *` file pointeren og `fopen` ved brug af `r`, da det kun er nødvendigt at læse data fra filen, og dermed ikke er nødvendigt at have filen åben således der kan skrives til den. Ligesom man ville erklære variabler, som konstanter, i funktions

parametre således de ikke bliver uhensigtsmæssigt ændret.

Herefter bliver der taget brug af en `fscanf`, som har til formål at indlæse informationerne fra filen. Alle variabler bliver her brugt i `fscanf`, da programmet bliver mere kompakt, ved kun at anvende en enkelt `fscanf`. Efterfølgende bliver alt informationen printet ud til skærmen ved brug af `printf`, her kan brugeren så bekræfte om informationen er korrekt.

```

297 void user_profile_loader()
298 {
299     FILE *file_pointer;
300     file_pointer = fopen("test.txt", "r");
301
302     if(fscanf(file_pointer, "%s\n%s\n%s\n%d\n%d\n%d\n%d\n%d\n",
303             user_profile.first_name, user_profile.last_name, user_profile.gender, &(user_profile.age), &(user_profile.height),
304             &(user_profile.weight), &(user_profile.calorie_need), &(user_profile.protein_need), &(user_profile.carbon_hydrates_need),
305             &(user_profile.fat_need), &(user_profile.activity_level), &(user_profile.nutrition_choice)) != 11)
306     {
307         print_error("INCORRECT ASSIGNMENT", "FR FILE LOADER", "FATAL");
308     }
309     printf("\nFollowing profile has been loaded\n");
310     "\nName: %s\n",
311     "\nSurname: %s\n",
312     "\nGender: %s\n",
313     "\nAge: %d\n",
314     "\nHeight: %d\n",
315     "\nWeight: %d\n",
316     "\nDaily kcal: %d\n",
317     "\nProtein: %d\n",
318     "\nCHydrates: %d\n",
319     "\nFat: %d\n",
320     "\nActivity lvl: %d\n",
321     "\nNut: %s\n",
322     user_profile.first_name, user_profile.last_name, user_profile.gender, user_profile.age, user_profile.height,
323     user_profile.weight, user_profile.calorie_need, user_profile.protein_need, user_profile.carbon_hydrates_need,
324     user_profile.fat_need, user_profile.activity_level, user_profile.nutrition_choice);
325
326     fclose(file_pointer);
327     user_profile.calorie_need = calorie_need_calculator();
328     nutrition_distribution();
329 }
330
331

```

Figure 3.5: Udsnit af funktionen User profile loader

Funktionen `User_profile_changer` er opsat således at den først indlæser profilen fra tekst filen, hvorefter den spørger brugeren om hvilken del af deres profil de godt kunne tænke sig at ændre. (ses i figur 3.6) Dette foregår ved at brugeren får alle sine muligheder, hvorefter deres valg tjekkes i en `switch`, der herefter tillader dem at ændre en enkelt værdi af gangen.

Herefter kaldes `profile_saver` igen, for at gemme dataen på ny, så informationerne kan anvendes andre steder i programmet.



```

350 switch(user_change_choice)
351 {
352     case 1:
353         printf("Please enter your new first name ");
354         scanf("%s", user_profile.first_name);
355         break;
356     case 2:
357         printf("Please enter your new last name ");
358         scanf("%s", user_profile.last_name);
359         break;
360     case 3:
361         printf("Please enter your new gender ");
362         scanf("%s", user_profile.gender);
363         break;
364     case 4:
365         printf("Please enter your new age ");
366         scanf("%d", &(user_profile.age));
367         break;
368     case 5:
369         printf("Please enter your new height ");
370         scanf("%d", &(user_profile.height));
371         break;
372     case 6:
373         printf("Please enter your new weight ");
374         scanf("%d", &(user_profile.weight));
375         break;
376     case 7:
377         printf("Please enter your new activity level");
378         scanf("%d", &(user_profile.activity_level));
379         break;
380     case 8:
381         nutrition_distribution(1);
382         break;
383     default:
384         print_error("INPUT ERROR!", "PROFILE CHANGER", "IRRELEVANT");
385 }
386
387 nutrition_distribution(0);
388 profile_saver();
389

```

Figure 3.6: Udsnit af funktionen User profile changer

```

847 double calorie_need_calculator()
848 {
849     double tmp;
850
851     if(strcmp(user_profile.gender, "m") == 0 || strcmp(user_profile.gender, "M") == 0)
852     {
853         if(user_profile.age < 10)
854         {
855             tmp = ((0.413 * (user_profile.weight) + 0.1 * (user_profile.height / TO_PERCENT)) - 7.5) * KCAL_CONSTANT);
856         }
857         else if(user_profile.age >= 10 && user_profile.age < 18)
858         {
859             tmp = ((0.481 * (user_profile.weight) + 0.15 * (user_profile.height / TO_PERCENT)) + 1.29) * KCAL_CONSTANT);
860         }
861         else if(user_profile.age >= 18 && user_profile.age < 30)
862         {
863             tmp = ((0.73 * (user_profile.weight) + 0.15 * (user_profile.height / TO_PERCENT)) + 1.44) * KCAL_CONSTANT);
864         }
865         else if(user_profile.age >= 30 && user_profile.age < 40)
866         {
867             tmp = (((0.01) * (user_profile.weight)) + (1.1 * (user_profile.height / TO_PERCENT)) + 0.475) * KCAL_CONSTANT);
868         }
869         else if(user_profile.age >= 40 && user_profile.age < 50)
870         {
871             tmp = ((0.456 * (user_profile.weight) + 0.16 * (user_profile.height / TO_PERCENT)) - 0.214) * KCAL_CONSTANT);
872         }
873         else if(user_profile.age > 50)
874         {
875             tmp = ((0.432 * (user_profile.weight) + 0.17 * (user_profile.height / TO_PERCENT)) - 0.219) * KCAL_CONSTANT);
876         }
877     }
878     else
879     {
880         printf("something is wrong with your age");
881     }
882 }

```

Figure 3.7: calorie need calculator

I funktionen `calorie_need_calculator`, udregnes energi behovet for brugeren. Først erklæres en midlertidig `double` som skal bruges til at opbevare værdien inden den returneres.

Funktionen består primært af en `if else`, som skal differentiere om brugeren er mand eller kvinder, hvorefter der efter hvilken aldersgruppe brugeren har, skal laves en bestemt udregning.

I samme funktion bliver tallet senere multipliceret med tallet for brugerens

aktivitetsniveau, dette udregner brugerens totale kcal behov. Denne anvendes i programmets andre funktioner til at udregne en madplan til brugeren.

```
916 switch(user_profile.activity_level)
917 {
918     case 1:
919         tmp *= 1.4;
920         break;
921     case 2:
922         tmp *= 1.2;
923         break;
924     case 3:
925         tmp *= 1.0;
926         break;
927     case 4:
928         tmp *= 0.8;
929         break;
930     case 5:
931         tmp *= 0.6;
932         break;
933     case 6:
934         tmp *= 0.4;
935         break;
936     case 7:
937         tmp *= 0.2;
938         break;
939     case 8:
940         tmp *= 0.0;
941         break;
942     case 9:
943         tmp *= 0.0;
944 }
```

Figure 3.8: Aktivitetsniveau

### 3.2.2 Programmets hovedmenu

Funktionen `check_for_which_function` samler alle funktioner. Denne funktion definere hvor brugeren har mulighed for at gå hen i programmet. I denne funktion kalder de største funktioner, i programmet. brugeren kan dermed vælge imellem programmets forskellige delprogrammer som udgør dets helhed.

Funktionen `check_for_which_function`, benytter sig af strings til at navigere, arrayet `control` tager imod brugerens input, der herefter sammenlignes med de forskellige programvalgsmuligheder. Dette er gjort i en `if else` kæde 3.9, da det ikke er muligt at benytte `strcmp` i `switches`. Samtidig med at funktionen orientere brugerene om deres valg, er funktionen også holder for data der sendes til og fra de forskellige funktioner. Med andre ord, er `check_for_which_function` programmets main menu. Dette er gjort for at give brugeren en let og intuitiv måde at føre sig igennem programmet.

```

406 while(!(i))
407 {
408     printf("What would you like to do?\n\n")
409     "Logout type:          \tlogout\n"
410     "create the weeks shopping list: shop\n"
411     "change your profile type: \tprofilechange\n"
412     "create the weeks's menu type: mealplan\n"
413     "view your profile type; \tviewprofile\n"
414     "if you need help please type: \thelp\n"
415     "quit type:            \tquit\n\n");
416     scanf("%s", control);
417
418     ++i;
419     if(!(strcmp(control, "logout")))
420
421     {
422         /*Call to */
423         system("cls");
424         new_or_old_profile(user_profile);
425     }
426     else if (!(strcmp(control, "mealplan")))
427     {
428         /*call to algoritme to make the menu*/
429         create_meal_composition(meal_id, meal_scaler, GoalValue);
430         meal_plan_printer(meal_id, meal_scaler, week_menu_weekday, dish);
431     }
432     else if (!(strcmp(control, "profilechange")))
433     {
434         /*call to a new profile, that can change the profile*/
435         system("cls");
436         user_profile_changer();
437         system("cls");
438         printf("change was succesfully made.. \n");
439     }

```

Figure 3.9: Funktionen check for which function while løkke

### 3.2.3 Hjælpesiden

Hjælpesiden til programmet er en liste af Frequently Asked Questions (FAQ) og en videre henvisning til gruppens e-mail (SW1B2-24@student.aau.dk). Brugeren bliver promptet til hvilke informationer man vil have fat i, for at forsøge at pinpointe spørgsmålet.

Funktionens data gør brug af et struct der kaldes `help_options`. Structet (se figur 3.10) kan virke overflødigt og kunne være blevet lavet med lokale variabler i stedet. Dog giver structet bedre overskuelighed for alle involverede programmer.

```
485 void main_menu_help_page()
486 {
487     FILE *fp = fopen("generalhelp.txt", "r");
488
489     help_options help;
490
491     answerswitch(help, fp);
492
493 }
```

Figure 3.10: help options structet

Programdelen er sat op af en række informationer til brugeren. Brugeren tager en beslutning i starten af programmet ved at gå ind i en skifte-funktion der kalder en anden funktion i programdelen ved bruger input. Det første man kan vælge er en yderligere hjælpeside der printes fra et tekstdokument eller en FAQ side. `answerswitch`(se side figur 3.11) promter brugeren med forskellige valg af spørgsmåls grupperinger. Her sættes en række muligheder op som brugeren kan vælge imellem. Man kan også skrive quit for at gå tilbage. valgmulighederne er sat op i en if else kæde, hvor der bliver kaldt forskellige funktioner, alt efter hvilke informationer brugeren ønsker. Dette gør det muligt at indskærpe hvilke informationer der skal præsenteres for brugeren, for at besvare deres spørgsmål.

```

496 void answerswitch(help_options help, FILE *fp)
497 {
498     unsigned int i;
499     printf("FAQ and general QsA.\n\n"
500           "Please make you selection\n"
501           "Access FAQ \t\t[FAQ]\n"
502           "Additional help \t[HELP]\n"
503           "Return to main menu \t[QUIT]\n"
504           ": ");
505     scanf(" %s", help.answer);
506
507     for(i = 0; i < strlen(help.answer); i++)
508     {
509         help.answer[i] = toupper(help.answer[i]);
510     }
511
512     if(!(strcmp(help.answer, "faq") || !strcmp(help.answer, "FAQ")))
513     {
514         system("cls");
515         FAQ(help, fp);
516     }
517     else if(!(strcmp(help.answer, "help") || !strcmp(help.answer, "HELP")))
518     {
519         system("cls");
520         general_help(help, fp);
521     }
522     else if(!(strcmp(help.answer, "quit") || !strcmp(help.answer, "QUIT")))
523     {
524         system("cls");
525         return;
526     }
527     else
528     {
529         system("cls");
530         print_error("INPUT ERROR", "HELP OPTIONS", "IRRELEVANT");
531         answerswitch(help, fp);
532     }
533     return;
534 }
535
536

```

Figure 3.11: Answerswitch funktion

På linje 549 (se figur 3.12) er der en **for-løkke**, som går igennem længden på `help.choice` og på hver i. plads bliver et stort bogstav assignet over til `help.choice` i. plads. Dette gøres fordi `if else` kæden kun skal tjekke de store bogstaver, fordi der senere hen kun anvendes store bogstaver. Til hjælpemenueen er der opsat to FAQ sider, en til brugerens profil og en til generelle spørgsmål til programmets brug. Disse er opdelt for at kunne indskærpe hvilket spørgsmål brugeren måtte have.

```

540 void FAQ(help_options help, FILE *fp)
541 {
542     unsigned int i = 0;
543     printf("\nFrequently Asked Questions (FAQ)\n"
544           "please make your selection\n\n"
545           "User profile \t [USER]\n"
546           "General use \t [GENERAL]\n");
547     scanf("%s", help.choice);
548
549     for(i = 0; i < strlen(help.choice); i++)
550     {
551         help.choice[i] = toupper(help.choice[i]);
552     }
553
554     if(!(strcmp(help.choice, "GENERAL")))
555     {
556         general_FAQ(help, fp);
557     }
558     else if(!(strcmp(help.choice, "USER")))
559     {
560         user_FAQ(help, fp);
561     }
562     else
563     {
564         print_error("INPUT ERROR!", "FAQ", "IRRELEVANT");
565
566         FAQ(help, fp);
567     }
568 }

```

Figure 3.12: FAQ

Funktionen `general_FAQ` har en `printf` som udskriver 5 eksempler på spørgsmål der kunne være om programmet. Spørgsmålene har fået et nummer angivet, hvor brugeren, ved brug af ints, skal skrive hvilket spørgsmål der skal vælges. Længere nede befinder sig en `switch` som køre med `help.question` som argument. Denne `switch` består af 5 cases, hvor hver case er bestående af et eksempel på et svar på tidligere nævnte spørgsmål, hvis brugerne har problemer med programmet, kunne de benytte sig af dette. Dette er skrevet for begge FAQ menuer der hver indeholder et spørgsmål og et tilhørende svar. Programmet kan derfor have flere eller færre spørgsmål ved at indsætte en case mere eller mindre i switchen.

```

571 void general_FAQ( help_options help, FILE *fp)
572 {
573     system("cls");
574     printf("General Sample Question 1\n\n"
575           "General Sample Question 2\n\n"
576           "General Sample Question 3\n\n"
577           "General Sample Question 4\n\n"
578           "General Sample Question 5\n\n"
579           "Quit          0\n\n"
580           "Please make your selection\n: ");
581     scanf("%d", &help.question);
582     if(! (help.question))
583     {
584         return;
585     }
586     else
587     {
588         switch(help.question)
589         {
590             case 1:
591                 system("cls");
592                 printf("\nGeneral Sample Answer 1\n\n");
593                 printf("Please make your selection\n"
594                       "Return to FAQ \t\t[FAQ]\n"
595                       "Return to General \t[QUIT]\n");
596                 scanf("%s", help.answer);
597
598                 if(! (strcmp(help.answer, "yes") || ! (strcmp(help.answer, "YES"))))
599                 {
600                     system("cls");
601                     FAQ(help, fp);
602                 }
603                 else
604                 {
605                     system("cls");
606                     answerswitch(help, fp);
607                 }
608                 break;
609             case 2:
610                 system("cls");
611                 printf("\nGeneral Sample Answer 2\n\n");

```

Figure 3.13: General FAQ

Den generelle hjælpemenu (se figur 3.13) aflæser alt teksten fra dokumentet: `generalhelp.txt`. Dette gør den med `fgetc`, hvor den gradvist aflæser hver enkelt char i dokumentet og printer dem i et `while`-loop indtil den når slutningen på filen. Dette giver lidt bedre fleksibilitet til hvad der egentlig skal stå i `generalhelp.txt` som vi så kan ændre i undervejs.

```
819 void general_help( help_options help, FILE *fp)
820 {
821     char helptextfile;
822
823     errorgeneralhelp(help, fp);
824
825     while((helptextfile = fgetc(fp)) != EOF)
826     {
827         printf("%c", helptextfile);
828     }
829
830     fclose(fp);
831
832     printf("\nReturning to main menu. \n\n");
833
834     answerswitch(help, fp);
835 }
```

Figure 3.14: Funktionen der printer tekstfilen generalhelp.txt

Til denne funktion er der blevet lavet funktionen **errorgeneralhelp**. Denne printer en error i det tilfælde at programmet ikke kan finde filen, **generalhelp.txt** hvori der står en vejledning til hjælp(se figur 3.14).



### 3.2.4 Databaseopsætning

For at kunne imødekomme de forskellige krav der vil være til datasæt med opskrifter og neutrino er det oplagt at opsætte en database til at opbevare disse informationer. Denne database udformer sig i sin mest primitive form, som en tekst fil, hvor informationerne der er nødvendige for programmets funktion opbevares. Dette giver muligheden for at opbevare større datasæt, der ikke skal hardcodes ind i programmet. Dette giver endvidere muligheden for at rette i informationerne uden at programmet behøver at blive omskrevet eller ændret i nogen form. Databasen bliver udarbejdet med informationer fra Madlog.dks database, men da vi ikke har konkrete informationer om hvordan Madlog.dk udregner kcal. i deres retter, skal retterne der er indskrevet i databasen kun ses som eksempel retter, hvis formål er at vise programmets funktionalitet.

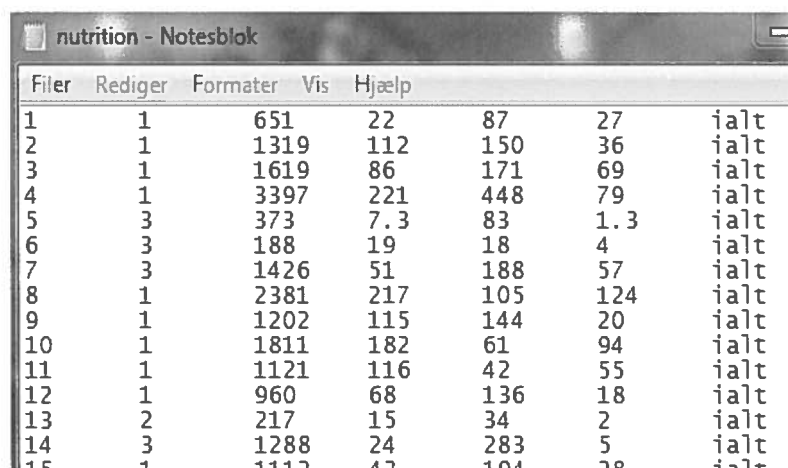
For at gøre opsætningen af databasen simple, og indlæsningen af filen det samme, bliver databasen indskrevet med 1 ret for hver linje. Dette gør det nemmere at indlæse dataen i programmet, da `\n`, der anvendes som linje terminator, kan bruges til at skilne imellem de forskellige opskrifter. endvidere er det nødvendigt at opstille nogle specifikke regler for hvordan informationen opstilles i filen før databasen kan opsættes:

1. Programmet skal have en identifikation på retten.
  - Dette kan gøres via en `int` eller en `char`. programmet anvender en `int` fordi retternes ID nummer, så kan anvendes som `int` i `for` eller `while` løkker.
  - Dernæst skal retten også have en type definition: morgen, middag og aftens ret. Dette gøres for at programmet kan identificere om retterne er morgen, middag eller aftensmad. Disse gives henholdsvis tallene 1. aftensmad 2. middagsmad og 3. morgenmad som indikator.

Med identifikationen på plads er næste spørgsmål om hvordan informationer om retten opdeles. Da madplans algoritmen kun skal benytte de næringsmæssige informationer, og indkøbsliste- og opstillingsfunktioner skal anvende opskriften og ingredienserne. er der en klar mulighed for at dele databasen i 2, så programmet ikke skal sortere i unødvendig data, hver gang der skal søges efter en ret. Dataene kan af denne grund fordeles til 2 filer(databaser): nutrition og ingredients.

Nutrition database: indeholder specifikke informationer om næringsindholdet af de individuelle opskrifter. Altså kcal, proteiner, kulhydrater og fedt. Disse oplysninger skal som tidligere nævnt benyttes i madplans algoritmen, hvor de sammenlignes med brugerens data.

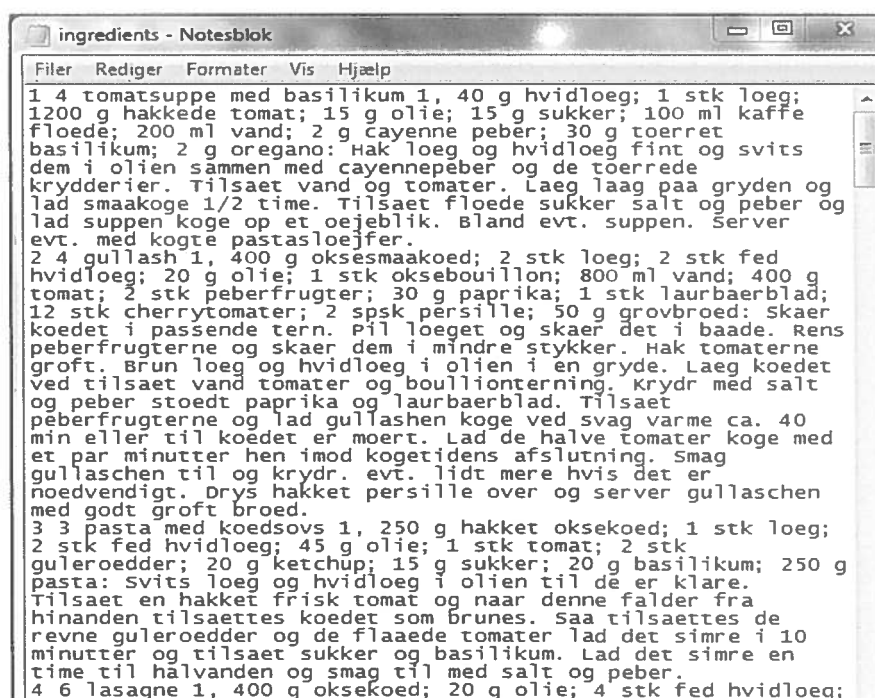
Ingredients database: indeholder de specifikke opskrifter der er anvendt i programmet. Alt skalering af retten foregår i programmets funktioner. Alle retter er derfor indskrevet som de er læst på Madlog.dk. Dette er med henblik på at forholdet imellem kcal og mængde bevares indtil disse bliver redefineret i programmets algoritmer.



	Filer	Rediger	Formater	Vis	Hjælp	
1	1	651	22	87	27	ialt
2	1	1319	112	150	36	ialt
3	1	1619	86	171	69	ialt
4	1	3397	221	448	79	ialt
5	3	373	7.3	83	1.3	ialt
6	3	188	19	18	4	ialt
7	3	1426	51	188	57	ialt
8	1	2381	217	105	124	ialt
9	1	1202	115	144	20	ialt
10	1	1811	182	61	94	ialt
11	1	1121	116	42	55	ialt
12	1	960	68	136	18	ialt
13	2	217	15	34	2	ialt
14	3	1288	24	283	5	ialt
15	1	1117	47	104	70	ialt

Figure 3.15: nutrition.txt

Se figur 3.15. Tabellen bliver indlæst i algoritmen efter: ID, type, kcal, protein, kulhydrater, fedt og mængde. Dette gør den for alle retter. Algoritmen sammenligner herefter dataen med oplysninger fra den indlæste profil og sender disse data videre til funktionerne for indkøbslisten.



	Filer	Rediger	Formater	Vis	Hjælp
1 4 tomatsuppe med basilikum 1, 40 g hvidloeg; 1 stk loeg; 1200 g hakkede tomat; 15 g olie; 15 g sukker; 100 ml kaffe floede; 200 ml vand; 2 g cayenne peber; 30 g toerret basilikum; 2 g oregano: Hak loeg og hvidloeg fint og svits dem i olien sammen med cayennepeber og de toerrede krydderier. Tilsaet vand og tomater. Laeg laag paa gryden og lad smaakoge 1/2 time. Tilsaet floede sukker salt og peber og lad suppen koge op et oejeblik. Bland evt. suppen. Server evt. med kogte pastasloejfer.					
2 4 gullash 1, 400 g oksesmaakoed; 2 stk loeg; 2 stk fed hvidloeg; 20 g olie; 1 stk oksebouillon; 800 ml vand; 400 g tomat; 2 stk peberfrugter; 30 g paprika; 1 stk laurbaerblad; 12 stk cherrytomater; 2 spsk persille; 50 g grovbroed: Skaer koedet i passende tern. Pil loeget og skaer det i baade. Rens peberfrugterne og skaer dem i mindre stykker. Hak tomaterne groft. Brun loeg og hvidloeg i olien i en gryde. Laeg koedet ved tilsaet vand tomat og boullionterning. Krydr med salt og peber stoedt paprika og laurbaerblad. Tilsaet peberfrugterne og lad gullashen koge ved svag varme ca. 40 min eller til koedet er moert. Lad de halve tomater koge med et par minutter hen imod koetidens afslutning. Smag gullaschen til og krydr. evt. lidt mere hvis det er noedvendigt. Drys hakket persille over og server gullaschen med godt groft broed.					
3 3 pasta med koedsovs 1, 250 g hakket oksekoed; 1 stk loeg; 2 stk fed hvidloeg; 45 g olie; 1 stk tomat; 2 stk guleroedder; 20 g ketchup; 15 g sukker; 20 g basilikum; 250 g pasta: Svits loeg og hvidloeg i olien til de er klare. Tilsaet en hakket frisk tomat og naar denne falder fra hinanden tilsaettes koedet som brunes. Saa tilsaettes de revne guleroedder og de flaaede tomater lad det simre i 10 minutter og tilsaet sukker og basilikum. Lad det simre en time til halvanden og smag til med salt og peber.					
4 6 lasagne 1, 400 g oksekoed; 20 g olie; 4 stk fed hvidloeg;					

Figure 3.16: ingredients.txt

Se figur 3.16. Filen bliver indlæst i funktionerne efter: ID, antal personer

(**mængde**), navn på retten + type, ingredienser separeret med et (;), der afsluttes med et (:) der fortæller programmet at fremgangsmåden begynder. Funktionerne indlæser dernæst, ligesom den anden database alle retter, og bruger disse oplysninger sammen med algoritmens udregninger, til at bearbejde en personlig madplan og indkøbsliste til brugeren.

### 3.2.5 Fil indlæser og indkøbsliste sortering

Fil indlæser og indkøbsliste funktionen, har til formål at hente allerede logførte opskrifter og ingredienser, med henblik på at skabe en let og overskuelig indkøbsliste til brugeren. De logførte opskrifter er udvalgt ifølge programmets hoved algoritme, som er beskrevet i tidligere afsnit.

For at undgå at lave et fastlagt system, har del programmet en funktion, der håndterer antallet af linjer i databasen. Dette er gjort for at man, på et senere tidspunkt, kan tilføje et ubegrænset antal linjer til databasen, uden at det vil have en effekt på hvordan programmet udfører sine opgaver. Samtidig har del programmet en funktion, hvor denne også finder antallet af ingredienser i opskriften, som senere vil blive benyttet i andre dele af del programmet. For at imødekomme dette krav, har vi opbygget en funktion der håndterer de specifikke krav til data håndtering i programmet.

```

969 void count_file_lines(char *lines, int identifier[])
970 {
971     FILE *pfile;
972     char ch;
973     int i = 0;
974
975     pfile = fopen("ingredients.txt", "r");
976
977     if(pfile == NULL)
978     {
979         print_error("FILE NOT FOUND", "COUNT_FILE_LINES", "FATAL");
980         return;
981     }
982
983     while(!feof(pfile))
984     {
985         ch = fgetc(pfile);
986         if(ch == '\n')
987         {
988             i++;
989         }
990         else if(ch == ';')
991         {
992             identifier[i]++;
993         }
994     }
995     *lines = i;
996     fclose(pfile);
997 }

```

Figure 3.17: Linje tæller funktion

Funktionen tager udgangspunkt i en `while` løkke, hvor en `file pointer` til tekst filen `ingredients.txt` har rettigheder til at læse i filen (dette ses ved `r` på linje 975). `while` løkken kører så længe at `fgetc` ikke er lig med `EOF`, som også kan læses som `-1` i `ansi C`. Se figur 3.17. `fgetc` anvendes for at finde hvert eneste karakter i filen, dette er valgt over `fgets`, som indlæser hele linjen som en string, fordi det så ikke ville være muligt at indhente informationerne til at adskille de

individuelle ingredienser i hver linje.

For hver gang at `fgetc` rammer linje terminatoren `\n`, tæller den `i` op med 1, for til sidst at have det totale antal linjer i tekst filen. Samtidig tæller den de individuelle ingredienser, som er inddelt ved brug af `;`. Disse informationer bliver lagret i et array, som under linje tallet, refererer til antallet af ingredienser for den specifikke linje, og dermed den enkelte opskrift.

Herefter sendes `i`, som indeholder det totale antal linjer, videre som pointeren `*lines`, for at informationerne kan anvendes i andre funktioner i del programmet. Det samme sker for arrayet `identifier` som indeholder det specifikke antal ingredienser. Som sidste led lukkes filen, for at undgå komplikationerne senere hen i programmet.

### File loader

Fil indlæser funktionen (se figur 3.18), har til formål at hente den nødvendige data fra programmets database. Herefter uploades informationerne ind i et globalt structure, så dataene kan benyttes i senere funktioner.

Funktionen starter som linje tæller funktionen. Der defineres en `FILE` pointer, og denne anvendes i tekst filen `ingredients.txt` hvor denne kan læse fra.

Herefter er der en `if else` kæde, som fungerer som en fail safe, i tilfældet af at der skulle være problemer med tekst filen, der indeholder dataene som skal benyttes i programmet. Denne sender et kald videre til en print funktion, der informere brugeren igennem prompten, om det specifikke problem, og hvor det er opstået. Samtidig sender `printf` en string med, som fortæller senere funktioner, om sværhedsgraden af fejlen der er opstået.

```

1003 int load_file(dish_data dish[], int *lines, int identifier[])
1004 {
1005     FILE *ptfile;
1006     int j, i;
1007
1008     ptfile = fopen("ingredients.txt", "r");
1009     if(ptfile == NULL)
1010     {
1011         print_error("FILE NOT FOUND", "LOAD_FILE", "FATAL");
1012         return(0);
1013     }
1014     else
1015     {
1016         for(i = 0; i < *lines + 1; i++)
1017         {
1018             fscanf(ptfile, "%d %d %[^,], %d ",
1019                 &dish[i].dish_id,
1020                 &dish[i].dish_amt,
1021                 dish[i].dish_name,
1022                 &dish[i].meal_time_id);
1023             for(j = 0; j < identifier[i] + 1; j++)
1024             {
1025                 fscanf(ptfile, "%lf %s %[^:];",
1026                     &dish[i].ingredients[j].ingamt,
1027                     dish[i].ingredients[j].ingvalue,
1028                     dish[i].ingredients[j].ingname);
1029             }
1030             fscanf(ptfile, ": %[^\\n]", dish[i].recipe);
1031         }
1032     }
1033     fclose(ptfile);
1034     return(0);
1035 }
1036

```

Figure 3.18: File Loader

Hvis dette imidlertid ikke sker, fortsætter programmet i sin `else` funktion, der anvender `*lines` og `identifier` fra linje tæller funktionen, til at sortere igennem de forskellige linjer af databasen. I disse 2 for løkker, indlæses informationen fra tekst filen, i et dobbelt structure, hvor den første for løkke håndterer selve retten, og anden for løkke, håndterer ingredienserne. Dette er gjort for at undgå at have en meget lang ingrediens structure, som ville være svær at definere. Samtidig gør det data håndteringen nemmere i fremtidige funktioner, fordi man både kan sortere i specifikke opskrifter, samtidig med at man har muligheden for at finde individuelle ingredienser. Til sidst lukker funktionen tekst filen, og returnere 0 til programmets main funktion.

Da div. structures er globale, er der nu mulighed for at tilkalde informationerne fra databasen fra en hvilket som helst funktion i programmet. Dette giver god mulighed for at håndtere dataene senere hen i programmet.

### Data sortering

Data sortering, bearbejder den medtagede data fra madplanlægningsalgoritmen, og indsamler data fra de individuelle structures med retter, for at lave en indkøbsliste til madplanen (Se figur 3.19). Data sorteringen, gøres så de individuelle ingredienser kan bearbejdes senere i programmet, og der ikke er en nød-

vendighed i at ændre på div. structures med den originale kopi fra databasen. Herved kan der altid hentes data fra div. structures uden at programmet har behov for at loade data fra databasen igen.

```

1044 void get_dish_info(dish_data dish[], int meal_id[], double meal_scaler[], int *lines, int identifier[])
1045 {
1046     int i, j, g, array_lgt = 0, k = 0;
1047     ingredients_data handler[MAX_STRUCT_LGT];
1048     ingredients_data shop_lst[MAX_STRUCT_LGT];
1049
1050     for(i = 0; i < MAX_MEALPLAN_LGT; i++)
1051     {
1052         if(meal_scaler[i] < 0)
1053         {
1054             print_error("INCORRECT SCALER", "FATAL_INGREDIENTS_AMT", "FATAL");
1055         }
1056         else if(meal_id[i])
1057         {
1058             for(j = 0; j < *lines + 1; j++)
1059             {
1060                 if(meal_id[i] == dish[j].dish_id)
1061                 {
1062                     for(g = 0; g < identifier[j] + 1; g++, k++)
1063                     {
1064                         strcpy(handler[k].ingname, dish[j].ingredients[g].ingname);
1065                         strcpy(handler[k].ingvalue, dish[j].ingredients[g].ingvalue);
1066                         handler[k].ingamt = (meal_scaler[i] * dish[j].ingredients[g].ingamt);
1067                     }
1068                     break;
1069                 }
1070             }
1071         }
1072         else
1073         {
1074             print_error("INCORRECT ID", "FATAL_INGREDIENTS_AMT", "NON_FATAL");
1075             break;
1076         }
1077     }
1078     array_lgt = k;
1079     sort_shp_lst(handler, shop_lst, array_lgt);
1080 }

```

Figure 3.19: Overfører data til struct og ganger med scaler

Funktionen modtager input fra linje tælleren og fil loaderen, samtidig modtager funktionen også informationer fra hoved algoritmen, som sender variablerne `meal_id` og `meal_scaler`, som indeholder ID på 21 forskellige retter (svarende til en uges madplan, med 3 forskellige retter om dagen) og en scalar der definerer om kalorie mængden i de forskellige retter skal ganges op, eller ned for at imødekomme de forskellige behov individet måtte have.

Funktionen starter med at gentage sig selv 21 gange, for at imødekomme alle 21 retter i madplanen. Herefter tjekker en `if else` kæde, om der er de korrekte informationer i henholdsvis `meal_scaler` og `meal_id`. Hvis disse informationer er korrekte, så fortsætter funktionen med at finde de relevante opskrifter, ved at sammenligne `meal_id` med de id'et på de forskellige retter der er i databasen. Når den korrekte opskrift bliver fundet, så kopieres ingredienserne over i et array, der midlertidigt holder informationerne til data behandling. Samtidig tælles det samlede antal ingredienser op, så mængden af ingredienser kan benyttes senere. Efter at disse informationer er blevet overført til `array_lgt` sendes der et funktionskald til sorteringsfunktionen `sort_shp_lst`.

### Sammenlægning og sortering

For at gøre det nemmere for brugeren, er der lavet en funktion, som fjerner alle dobbelt ingredienser, så fx løg ikke optræder flere gange på indkøbslisten, men blot en gang sammenlagt, og i alfabetisk rækkefølge (Se figur 3.20). Dette er for at indkøbslisten skal være nemmere for brugeren at anvende, da man ellers ved gennemgang af indkøbslisten, blot ville få ingredienserne i samme rækkefølge som opskrifterne optræder i madplanen. Samtidig giver det en mulighed for at købe alle ingredienserne på en indkøbstur, og giver dermed brugeren mulighed

for at begrænse sine impulskøb.

```

1086 void sort_shp_lst(ingredients_data handler[], ingredients_data shop_lst[], int array_lgt)
1087 {
1088     int i, j, g, identifier, shop_lst_size;
1089     shop_lst_size = 0;
1090     for(j = 0, i = 0; j < array_lgt; j++)
1091     {
1092         if(handler[j].ingamt)
1093         {
1094             for(g = 0, identifier = 0; g < shop_lst_size; g++)
1095             {
1096                 if (!strcmp(shop_lst[g].ingname, handler[j].ingname))
1097                 {
1098                     identifier = 1;
1099                     shop_lst[g].ingamt += handler[j].ingamt;
1100                 }
1101             }
1102             if (!identifier)
1103             {
1104                 shop_lst[i++] = handler[j];
1105                 shop_lst_size++;
1106             }
1107         }
1108     }
1109     shop_lst_size--;
1110     qsort(shop_lst, shop_lst_size, sizeof(ingredients_data), compare_sort_shp_lst);
1111     print_func_ing(shop_lst, shop_lst_size, "sort_shp_lst");
1112 }
1113

```

Figure 3.20: Data sortering

Funktionen starter med at anvende `int` med det totale antal ingredienser på indkøbslisten, herefter køre det et tjek på arrayet `handler` hvis denne indeholder den korrekte information, så fortætter funktionen, ellers tæller `for` løkken videre og finder næste ret. Herefter køres en `for` løkke, der kun opdatere hvis `identifier` (der er en boolsk værdi) opdateres ved det efterfølgende `if` check. Herved sættes alle ingredienser der ikke er ens med allerede eksisterende ingredienser på listen, ind på listen, og ingredienser der er ens, bliver lagt sammen med de eksisterende ingredienser på listen, under deres respektive råvarer. Herefter tælles størrelsen af indkøbslisten op, og dette anvendes til at kalde `qsort` funktionen, der sorterer ingredienserne efter alfabetisk rækkefølge. Til sidst kaldes `print` funktionen (se figur 3.21), der viser indkøbslisten for brugeren.

### Opskrifts håndtering

Nu hvor brugeren har alle ingredienserne til at kunne lave de specifikke opskrifter på deres madplan, mangler brugeren blot at vide hvad de skal stille op med de forskellige råvarer de har været ude og købe (Se figur 3.21). Hertil kommer opskrifts håndtering. For at gøre det nemt for brugeren at opstille en liste over de forskellige opskrifter, der kan hentes ned så brugeren kan anvende opskrifterne til at lave retterne.

```

1119 void recipe_data_handling(double meal_id[], dish_data dish[], int identifier[])
1120 {
1121     int i, j, g, id, recipe_list_size = 0;
1122     int ingredient_lgt[MAX_STRING_LGT];
1123     dish_data handler[MAX_MEALPLAN_LGT];
1124
1125     for(j = 0, i = 0; j < MAX_MEALPLAN_LGT; j++, i++)
1126     {
1127         if(meal_id[j])
1128         {
1129             for(g = 0, id = 0; g < MAX_MEALPLAN_LGT; g++)
1130             {
1131                 if(dish[g].dish_id == meal_id[j])
1132                 {
1133                     id = 1;
1134                     handler[i] = dish[g];
1135                     ingredient_lgt[i] = identifier[g];
1136                     break;
1137                 }
1138             }
1139             if(id)
1140             {
1141                 recipe_list_size++;
1142             }
1143         }
1144     }
1145     print_func_dish(handler, ingredient_lgt, recipe_list_size, "recipe_data_handling");
1146 }

```

Figure 3.21: Opskrifternes data bliver behandlet

Funktionen starter som før, ved at søge igennem den totale mængde af opskrifter i madplanen. Herefter testes der om der er de relevante informationer funktionerne har brug for, for at kunne udføre de specifikke beregninger. Hvis de specifikke krav er til stede, køres der en sammenligning imellem de to relevante structures, hvor hvis sammenligningen stemmer over ens, bliver dataene assignet til et array af structures typen og et int array, for at lagre antallet af ingredienser der er i den specifikke ret. Samtidig opdateres id, der er en boolsk værdi, til positiv og opskriftslistens længde opdateres med 1. Herefter sendes de specifikke informationer til en print funktion, der viser de relevante informationer til brugeren. Herefter kan brugeren anvende opskriften til at lave den specifikke ret, uden at skulle tænke på at registrere sine indtag bagefter.



### 3.2.6 Algoritmen som vælger retter til brugeren

Algoritmen er programmets hovedfunktion, heri kalkuleres madplanen ud fra brugerens input, og forsøger at skabe det bedst mulige match til brugerens behov. Efter kalkulationen sendes kalkulerede data videre til andre dele af programmet, der ud fra de givne data, præsenterer algoritmens valg af madplan. Dette spare brugeren for en masse tid, da brugeren ikke selv skal tænke på at lave madplanen, eller tage stilling til forskellige retter.

Algoritmens datastruktur er opbygget omkring to globale structs, disse bliver `typedef`, for lettere tilgang og øget læsbarhed. Hvilket vil sige at meal kan tilgås på næsten samme måde som en data type fx. `int` eller `char`.

```

50  typedef struct
51  {
52      double id, type, kcal;
53      double prot, carbs, fat;
54      char mass[MAX_STRING_LGT];
55  }meal;
56
57  typedef struct
58  {
59      double id;
60      int ss, mm, hh, day, month, year;
61  }ntime;

```

Figure 3.22: Meal og ntime structs

Structed meal i figur 3.22 indeholder dataen fra et enkelt måltid. De forskellige måltider defineres i et array af denne type, for at kunne indeholde informationerne om retterne der bliver arbejdet med, fra databasen `nutrition.txt`. Alle variabler under meal bliver erklæret som doubles. Dette gøres fordi der i algoritmen er behov for at benytte informationer med decimaler. Disse kan senere `typecastes` til `int` når dette bliver relevant. Structed `ntime` indeholder informationer om hvornår et specifikt måltid fra databasen er blevet brugt sidst. Dette er for at bibeholde variation i måltiderne, og undgå at algoritmen vælger de samme retter hver eneste uge, men i stedet venter 14 dage før den samme ret kan vælges igen. Denne funktion er dog ikke fuldt funktionel, da programmet ikke kunne færdigudvikles.

```

1506 void load_database(meal *m)
1507 {
1508     /* Pick up data from the database... */
1509     FILE *input_file_pointer = fopen("Nutrition.txt", "r");
1510     int i;
1511     if(input_file_pointer != NULL)
1512     {
1513         for(i=0; i < NUMBER_OF_DATABASE_ROWS; i++)
1514         {
1515             fscanf(input_file_pointer,
1516                 "%lf %lf %lf %lf %lf %s",
1517                 &m[i].id, &m[i].type, &m[i].kcal, &m[i].prot,
1518                 &m[i].carbs, &m[i].fat, m[i].mass);
1519         }
1520         fclose(input_file_pointer);
1521     }
1522     else
1523     {
1524         printf("Error loading file, bye\n");
1525         perror("Error");
1526     }
1527 }

```

Figure 3.23: load database henter data for nutrition.txt

Funktionen `load_database`, i 3.23 henter dataene fra `nutrition.txt` (figur 3.25) databasen som indeholder informationerne for de forskellige retters' næringsindhold. Den hentede data, bliver efterfølgende lagt ind i et array af typen `meal` som vil være den primære data kilde til at kunne vælge en endelig ret.

```

1412 double determine_dish(double *scale_factor, double exception_list[], double *exceptions_amount, double meal_type, meal nut_meal, meal *a, ntime *time_list)
1413 {
1414     double proposed_array_dish[NUTRITION_CATEGORIES - NUT_CAT_END_SKIP], tmp_scale;
1415     best_approximated_dish_id = -1, nutrition_array[NUTRITION_CATEGORIES - NUT_CAT_END_SKIP];
1416     last_comparing_factor = 0, current_comparing_factor;
1417     /* Assigning last_comparing_factor with a high value,
1418     so we can assign lower values as best_comparing_factor */
1419     int i, j, exception = 0, deviation = 0;
1420     meal proposed_meal;
1421     ntime time_present = current_time();
1422     /* Initialize Nutrition array from nutrition meal */
1423     struct_to_array(nut_meal, nutrition_array);
1424     for(i = 0; i <= NUMBER_OF_DATABASE_ROWS; i++)
1425     {
1426         /* The exception is a flag */
1427         /* If exception = 0, the function will run for the current dish id (i)
1428         if exception = 1, the function current iteration of i, will be skipped, as the i represents the current dish id,
1429         is an id of a dish that should not be chosen */
1430         exception = 0;
1431         if(*exceptions_amount > 0)
1432         {
1433             for(j = 0; j < ((int)*exceptions_amount); j++)
1434             {
1435                 if(exception_list[j] == i)
1436                 {
1437                     exception = 1;
1438                 }
1439             }
1440             /* This part doesn't work as intended,
1441             if a dish is used more than 14 days ago, it can be
1442             reused multiple times.
1443             Since the function "struct_to_array" from time variation.txt doesn't work */
1444             if((time_list[j], time_present) >= 14) && (exception_list[j] == i)
1445             {
1446                 exception = 1;
1447             }
1448         }
1449     }
1450     /* If the current dish should not be evaluated, or if the current dish is of the wrong type
1451     */

```

## (a) Determine dish funktionens første del

```

1453     (eg. if we are looking for a breakfast meal, and current dish is of type dinner -> skip current dish) */
1454     if((exception == 1) || (meal_type != m[i].type))
1455     {
1456         continue;
1457     }
1458     /* Get (current dish) struct meal from Database */
1459     /* Fill our proposed_dish_array with information from one specific dish */
1460     /* From Database */
1461     proposed_meal = m[i];
1462     /* Change our proposed_array_dish to store information from our proposed meal */
1463     struct_to_array(proposed_meal, proposed_array_dish);
1464     /* Scale the proposed dish array, to fit our nutrition needs (from nutrition array) */
1465     /* the desired scale value is found by comparing the total kcal value of our nutrition_array and from our proposed dish array */
1466     tmp_scale = scale_array(nutrition_array, proposed_array_dish, (int)meal_type);
1467     /* Get the average comparing factor by comparing every value in the two arrays, with each other */
1468     /* See -> get_comparing_factor function */
1469     current_comparing_factor = get_comparing_factor(nutrition_array, proposed_array_dish, &deviation);
1470     /* Compare the two arrays
1471     if the current dish's comparing factor is lower than the last
1472     (eg. the dish is a better match to our goal values of nutrition_array) */
1473     /* abort if the deviation is too big
1474     (also, the definition of 'too big deviation' is
1475     defined as a global variable MAX_DEVIATION */
1476     if((current_comparing_factor < last_comparing_factor) && !(deviation))
1477     {
1478         last_comparing_factor = current_comparing_factor;
1479         /* Save the id of the current dish */
1480         /* If no better dish is found, this dish's id will be returned as the best approximated dish */
1481         best_approximated_dish_id = (double)i;
1482         /* Store information about current scale factor of current dish, to the scale_factor pointer
1483         If this dish happens to be the final (best approximated) dish, the scale_factor wont be
1484         overwritten anymore, and the value is safely stored */
1485         *scale_factor = tmp_scale;
1486     }
1487     deviation = 0;
1488 }

```

## (b) Determine dish funktionens anden del

Funktionen `determine_dish`, som vist i figur 3.24a og 3.24b vælger en ret ud fra arrayet af typen `meal`. Funktionen er stillet op til at vurdere hver enkel næringsværdi i en ret (fedt, proteiner eller kulhydrater) og hvis en af værdierne afviger mere en x% fra mål værdien, bliver retten kasseret. Hvor x er angivet som `MAX_DEVIATION` som en `define`, dermed kan denne nemt tilgås, og rettes hvis det skulle blive nødvendigt.

17	2	312	31	32	6.5	ialt
18	2	9806	345	1673	154	ialt
19	2	1368	40	139	76	ialt
20	2	301	39	10	12	ialt
21	2	1771	116	193	58	ialt

Figure 3.25: Nutrition.txt udsnit

determine\_dish anvender en for løkke, til at sortere igennem de forskellige data linjer i databasen nutrition.txt, her sortere funktionen igennem de forskellige retter, og bearbejder dataen igennem en række if else kæder, for at finde den bedste sammensætning af retter til madplanen. Samtidig samler den nødvendig data til behandling i andre funktioner, for bedre at kunne sammenligne værdierne i de forskellige arrays af typen struct. Denne funktion er udviklet til at være centrum i algoritmen, hvor andre dataen sendes frem og tilbage til denne funktion for til sidst at erklære den bedste sammensætning.

```

1578 double get_comparing_factor(const double const_array[], double compare_array[], int *deviation)
1579 {
1580     int i;
1581     double comparing_factor = 0;
1582     /* Initializing i to nutrition category start skip + 1, to make it skip kcal comparing */
1583     for(i = NUT_CAT_START_SKIP + 1; i < NUTRITION_CATEGORIES - NUT_CAT_END_SKIP; i++)
1584     {
1585         /* Get the percentage deviation */
1586         comparing_factor += (sqrt((const_array[i] - compare_array[i]) * (const_array[i] - compare_array[i])) / const_array[i]);
1587         if (get_is_deviating((double)const_array[i], (double)compare_array[i]))
1588             *deviation = 1;
1589     }
1590     /* Find the average of all the percentage deviations of the two arrays's indexes */
1591     comparing_factor /= (NUTRITION_CATEGORIES - 1);
1592     return comparing_factor;
1593 }

```

Figure 3.26: Funktionen get comparing factor

Funktionen get\_comparing\_factor vist i figur 3.26 bearbejder informationer fra de samlede arrays, og en int variabel, med henblik på at finde gennemsnittet af differencen mellem kcal, fedt, protein og kulhydrat. Herefter sikre funktionen at udregningen returnere et positivt tal. Dette sendes videre i et funktionskald til get\_is\_deviating vist i figur 3.27. Denne sammenligner de 2 værdier, og returnere den procentvise difference imellem de 2 værdier. Rettens difference er større en MAX\_DEVIATION definen, returnere funktionen 1, til funktionskaldet. Og retten kasseres, som værende upassende til madplanen. Disse 2 funktioner, arbejder tæt sammen om at finde de forskellige retter som ikke kan benyttes i en madplan, pga. variationen i kcal, protein, fedt eller kulhydrater.

```

1602 int get_is_deviating(double static_value, double test_value)
1603 {
1604     double c;
1605     int result = 0;
1606
1607     /*Get the absolute value of the difference between static_value and test_value*/
1608     c = sqrt((static_value - test_value) * (static_value - test_value));
1609     c /= static_value;
1610
1611     /* If the deviation of the two values are greater than MAX_DEVIATION -> deviation = true. */
1612     if (c > MAX_DEVIATION)
1613     {
1614         result = 1;
1615     }
1616     return result;
1617 }

```

Figure 3.27: get is deviating funktionen

Samtidig med at tidligere funktioner, tjekker om kcal, protein, fedt og kulhydrater passer til brugerens specifikke behov. Herefter skal der sikres at algoritmen ikke vælger de samme retter hver gang den laver en madplan. Dette kan

gøres ved at sætte nogle specifikke tidskrav.

Funktionen `current_time`), indeholder en række funktioner fra header filen `time.h`, disse anvendes til at få dato og tid fra computerens operativsystem. Disse informationer gemmes i et structet `ntime`, hvor det senere bliver anvendt til sammenligning med allerede logførte opskrifter. Funktionen `write_timeline_to_file` (vist i figur 3.32) logfører de allerede anvendte retter, ved at tilføje et *"tidsstemplet"* til de forskellige retter som algoritmen har anvendt indenfor 14 dage. Dette gør at outputtet fra `current_time` kan sammenlignes med de logførte retter senere hen i databehandlingen.

Når prgrammet køres, og en ny madplan skal laves, kaldes funktionen `time_between` som beregner tiden imellem `current_times`' output og stemplet på den givne ret. Funktionen er vist i figur 3.28.

```

1685 ntime current_time()
1686 {
1687     time_t timer;
1688     ntime t;
1689     char cur_time[40];
1690     struct tm* tm_info;
1691     time(&timer);
1692     tm_info = localtime(&timer);
1693
1694     strftime(cur_time, 20, "%H:%M:%S - %d:%m:%Y", tm_info);
1695     sscanf(cur_time, "%d:%d:%d - %d:%d:%d",
1696            &t.hh, &t.mm, &t.ss,
1697            &t.day, &t.month, &t.year);
1698     t.id = 0;
1699     return t;
1700 }

```

Figure 3.28: current time funktionen

Hvis stemplet ikke er tilgængeligt, eller hvis antal dage siden retten blev anvendt er mere end 14. Kan retten anvendes, ellers stopper funktionen selectionen af retten her, kassere den og finder en ny. Denne funktion indeholder

anvendte dele fra undervisningen, hvor en undtagelses funktion `is_leap_year`, vist i figur 3.29[18], anvendes til at beregne om det er et skudår. Dette gør at funktionen kan anvendes på alle årstal, uden at fx. d 29 feb, kommer i vejen for programmet.

```

1853 int is_leap_year(int year)
1854 {
1855     return (year % 400 == 0) || ((year % 4 == 0) && (year % 100 != 0));
1856 }

```

Figure 3.29: is leap year funktionen fra undervisningen til at beregne skudår.

Algoritmen har nu de forskellige retter på plads, herefter skal algoritmen udregne en skaleringsfaktor for de forskellige retter, for at få disse til, bedst muligt, at passe sammen med brugerens behov.

`scale_array` funktionen (figur 3.30) starter med at udregne en skalar imellem 2 forskellige array, med data om de retter der indtil videre er valgt af algoritmen. når disse er skaleret, sammenlignes de med deres måltidstype (morgen, middag eller aftensmad) og ganger opskriftens kcal, indhold med en skalar der udregner fordelingen af kcal, på de forskellige måltider henover dagen.

Dette gøres for at div. opskrifter kan tilpasses til brugerens kcal. behov. Samtidig sørger funktionen for at en enkelt ret ikke kommer til at optage hele dagens kcal indtag.

```

1537 double scale_array(double const_array[], double target_array[], int meal_type)
1538 {
1539     int i;
1540     double scale_factor, new_target_value, modifier;
1541     switch(meal_type)
1542     {
1543     case 0:
1544         modifier = 1;
1545         break;
1546     case 1:
1547         modifier = MEAL_SHARE_DINNER;
1548         break;
1549     case 2:
1550         modifier = MEAL_SHARE_LUNCH;
1551         break;
1552     case 3:
1553         modifier = MEAL_SHARE_BREAKFAST;
1554         break;
1555     }
1556     /* Calculate how much we should scale target_array */
1557     /* Based on the const array total kcal */
1558     scale_factor = (const_array[2] * modifier) / target_array[2];
1559     /* Scale every value in the target_array */
1560     for(i = NUT_CAT_START_SKIP; i < NUTRITION_CATEGORIES - NUT_CAT_END_SKIP; i++)
1561     {
1562         new_target_value = (double)target_array[i] * scale_factor;
1563         target_array[i] = new_target_value;
1564     }
1565     return scale_factor;
1566 }
1567

```

Figure 3.30: scale array funktionen skalerer imellem to arrays med data om madtyper

`create_meal_composition` som vist i figur 3.31 denne funktion virker som algoritmens main. denne funktion bidrager ikke selv med udregninger, men samler blot informationerne for bedre overblik.

```

1335 void create_meal_composition(int meal_array[], double scale_array[], meal nutrition_meal)
1336 {
1337     int i;
1338     double exceptions_count = 0, meal_type = 0, exception_list[20];
1339     char *file_name = "Food_Variations.txt";
1340     ntime except_time_list[20];
1341
1342     /* Nutrition Database List */
1343     meal nut_db_list(NUMBER_OF_DATABASE_ROWS);
1344
1345     /* Find the amount of exceptions (eg. one exception = one id that should not be chosen as a dish) */
1346     exceptions_count = count_lines_in_file(file_name);
1347     load_database(NutDBList);
1348     load_time_data(file_name, except_time_list, &exceptions_count);
1349     make_exception_array(exception_list, file_name, except_time_list, &exceptions_count);
1350
1351     for(i = 0; i < MAX_MEALPLAN_LGT; i++)
1352     {
1353         meal_type = (i / 5) + 1;
1354         meal_array[i] = (int)determine_dish(&scale_array[i], exception_list, &exceptions_count,
1355                                           meal_type, nutrition_meal, NutDBList, except_time_list);
1356         check_add_exception(exception_list, meal_array[i], &exceptions_count, file_name, except_time_list);
1357     }
1358 }

```

Figure 3.31: create meal composition

`check_add_exception` vist i figur 3.32 tjekker om et givent ID, allerede er i listen over retter der allerede er blevet valgt indenfor 14 dage. Hvis retten ikke er der, bliver den tilføjet til listen, og funktionen kan dermed genkende den ved en anden gennemgang af filen. Hvis IDet allerede er på listen, returnerer funktionen 0. Dette er et forsøg på at sikre at retter ikke bliver valgt indenfor et interval af 14 dage. Denne funktion er dog ikke færdigudviklet.

```

1369 int check_add_exception(ndouble exception_list[], double id, double *exception_list_count, char *file_name, ntime *t)
1370 {
1371     int return_value = 0;
1372
1373     /* If current id is not in exception list - add it */
1374     load_time_data(file_name, t, exception_list_count);
1375
1376     /* If the current dish is not already in the exception list
1377     And if the current id is above or greater to 0 (not -1)
1378     -> add the dish id to exception list, so it wont be picked another time */
1379     if((get_id_index_in_list(id, t, exception_list_count) == -1) && (id >= 0))
1380     {
1381         write_timeline_to_file(file_name, id); /* Write a line of current date and time to the fat file "Food_Variations.txt" */
1382         *exception_list_count += 1; /* Add 1 to exception list count to keep track of number of exceptions */
1383         exception_list[(int)*exception_list_count - 1] = id; /* Add the id at the last slot of exception list */
1384         load_time_data(file_name, t, exception_list_count); /* Update the "ntime *t" list (list of dishes already picked at a certain time) */
1385         return_value = 1;
1386     }
1387     /* If a new exception were made, return 1, else return 0 */
1388     return return_value;
1389 }

```

Figure 3.32: Check add exception funktionen der tjekker ID

### 3.3 Program udførsel

Programmet starter med at brugeren vælger om han har en profil eller ikke. Hvis brugeren allerede har en profil, indlæser programmet automatisk profilen og viser brugeren alle personlige data. I tilfælde af at brugeren ikke har en profil, skal brugeren indtaste data som navn, køn, alder, højde, vægt og hvor aktiv brugeren er i hverdagen. Derefter spørger programmet hvilke ernæringsmål programmet skal følge. Alt efter om brugeren vælger at bevare vægten, tabe vægt eller øge sin muskelmasse, tilpasser programmet brugerens daglige behov. Programmet viser brugeren de forskellige informationer om hvor mange kcal, proteiner, fedt og kulhydrater brugeren har behov for, for at opnå sit mål. Hermed er brugerens profil opstillet og brugeren bliver sendt videre til hovedmenuen.

```

Hey there! and welcome to the new and improved meal planner 1.0!
Now.. do you already have a profile set up and are ready to go? <Y/N> n
Welcome to the one time account / user setup
Please type in your name
Marco
Please type in your surname
Polo
What is your gender? <M/F>
M
How old are you? age between 0-99
30
How tall are you? in centimeters 30-272
180
What's your weight in kilograms? 30-500
60
How active would you say you are during the day from 1 to 14
1: Literally not moving, at all during the day
7: Average daily exercise
14: Heavy daily exercise
7
Now, what is your goal with your diet going forward?
Maintain weight: 11
Increase weight: 12
Lose weight: 13
3
What would you like to do?

Logout type:          logout
create the weeks shopping list: shop
change your profile type: profilechange
create the weeks's menu type: mealplan
view your profile type: viewprofile
if you need help please type: help
quit type:            quit

```

Figure 3.33: Profil

I hovedmenuen kan brugeren vælge mellem syv funktioner (se figur 3.33): logout, shop, profilechange, mealplan, viewprofile, help og quit. Ved brug af logout bliver brugeren sendt tilbage til login hvor brugeren kan lave en ny profil eller lade en profil der allerede eksistere. Hvis brugeren har brug for en madplan der følger ernæringsmålet kan brugeren vælge madplansfunktionen, hvori brugeren får opstillet en madplan for en uge. Til hver dag i ugen, bliver der opstillet tre retter, en til morgen, middag og aftensmad (se figur 3.34).

Madplanen viser hvor stort et indhold af næringsstoffer retten har og hvor store portioner brugeren bør spise, for at opnå brugerens daglige behov. Med shop funktionen kan brugeren opstille en indkøbsliste med alle ingredienser for madplanen. Det er muligt at ændre en del af profilen med profilechange. Hvis brugeren vælger at ændre profilen, bliver der vist alle de data der er gemt i profilen. Hver data bliver repræsenteret af et tal som brugeren kan vælge et af,

Meal	Thursday
Breakfast	0.4 × 038 med ægeli
Lunch	2.0 × tomat ragout
Dinner	0.4 × Moerbrad med pesto mozzarella og soltørrede tomater
Meal	Friday
Breakfast	3.0 × Frugt morgenmad
Lunch	0.4 × valdorfsalat
Dinner	1.3 × Minestrone suppe med kylling

13.5 stk	loeg
225.2 ml	mælk
25.2 g	nudler
6.2 g	mayonnaise
4.0 g	muskatnød
380.7 g	nudler
446.2 g	oksesnaknød
41.4 ml	olie
3.3 g	oregano
4.2 g	paprika
33.2 g	parmesanost
6.5 g	peber
7.0 stk	peberfrugt
60.3 g	peberød
89.4 g	persille
2.3 stk	porre
15.1 stk	purloeg

Figure 3.34: Madplan og Indkøbsliste

for at ændre en del af profilen. Derefter bliver brugeren sendt til hovedmenuen hvor det er muligt at se alle ændringerne med viewprofile. I det tilfælde af at brugeren har nogle spørgsmål, kan brugeren gøre brug af help funktionen. I denne funktion bliver brugeren sendt til FAQ siden hvor ofte stillede spørgsmål bliver besvaret. Den sidste funktion er quit funktionen, der kan bruges til at afslutte programmet.

## 3.4 Vurdering

### 3.4.1 Fejl og mangler i programmet

Ikke alle programmets succeskriterier er blevet implementeret med ønsket effekt. I 2. kriterie ønskes det at individet skal kunne vælge blandt en liste eller database af retter. Dette er dog ikke tilfældet da brugeren blot bliver givet en plan for ugen. Ønskes en anden kan programmet køres igen, hvorved programmet vil sørge for at brugeren ikke får de samme ting, dog kan dette med vores begrænsede database resultere i at programmet udelukker flere eller alle retter.

Missing feature mis-tanke	In function	Fixable with
1. Unødvendig if efter else if i kæde	New user basic input	else if med korrekte assignments således at den ekstra if kunne fjernes



2. Programmet kan ikke bruges af folk over 99 år gammel	New user basic input	Øge den øvre grænse for alderen
3. Programmet kan ikke bruges af folk over 272 eller under 32 cm I højde	New user basic input	Forøgelse eller formindskelse af henholdsvis øvre og nedre grænse
4. Programmet kan ikke bruges af individer der vejer mere end 500 eller mindre end 32 kilogram	New user basic input	Forøgelse eller formindskelse af henholdsvis øvre og nedre grænse
5. Returnere til hoved menu i stedet for at kalde funktionen igen således at brugeren har endnu en chance	user profile changer	Uvist, forsøgt flere gange dog uden held eller bedre resultater
6. At program kun gemmer den ændrede variable I filen frem for hele filen med 1 ændret variabel	user profile changer	Uvist eller ikke forsøgt
7. Kun udprintning af brugerens profil I stedet for at load hele brugerens profil på ny	check for which function	En printf af brugerens data
8. Flere funktioner der går ud af programmet	check for which function	Korrekt brug af kæden eller assignments (se 1.)
9. Unødvendigt brug af et flertal af print og fprintf	Meal plan printer	Enkelt men større printf og fprintf kunne spare på ressourcerne
10. Ægte eller ikke prøve spørgsmål	General FAQ	Skrive korrekte eller meningsfulde spørgsmål
11. Ægte eller ikke prøve spørgsmål	User FAQ	Skrive korrekte eller meningsfulde spørgsmål
12. Ingen grund til at printe en fejlbesked, foregår I en separat funktion	General help / error-generalhelp	Have beskeden I den originale funktion giver både mere logisk mening og er mere besparende eller korrekt
13. Ikke brug af den dertil lavede error print funktion	errorgeneralhelp	Slette den gamle printf og tage print error funktionen i brug

14. overkompliceret næsten kopieret switch	Calorie need calculator	En enkelt switch, der i hver case havde en if der tjekker køn kunne være mere elegant og nemt at navigere for en potentiel læser
15. fil pointer og deklaration af fopen 2 forskellige steder	Count file line	Sammen trækning af disse
16. assignment af 1 til en variabel andet steds end deklarationen af variablen	Sort shp lst	Sammentrækning af disse
17. Shop lst size erklæret som 1 for blot at blive fratrasket med 1 igen I slutningen af funktionen for at den virker korrekt undervejs	Shop lst size	Uvist eller ikke forsøgt
18. identifier er lagt op med 1 for at den virker	Recipe data handling	Anderledes handling af funktionen eller deklaration af identifier
19. Ingen fejl tjek eller fejlbesked	Print func dish	Tilføjelse af førnævnte
20. Ingen fejl tjek eller fejlbesked	Print func ing	Tilføjelse af førnævnte
21. Over kommentering	check add exception	Gennemlæsning af kommentarer og skære unødvendig info eller detaljer fra
22. enkeltstående printf af en fejlbesked	load database	Brug af print error funktionen
23. Over kommentering	struct to array	Fjernelse, for mange unødvendige detaljer
24. Det er ikke nødvendigt med tjek for time, minut og sekund, og kan lede til retter der burde kunne vælges af programmet der ikke bliver valgt, skulle programmet blive kørt tidligere på dagen end for 14 dage siden	write timeline to file	Kun brug af år, måned og dag
25. Unødvendig at sætte en variabel og så lukke filen senere for så at lukke den uden for løkken	load time data	Luk filen og returner den ønskede værdi inde i løkken ved det ønskede sted ved de ønskede krav

26. Unødvendigt tjek for om filen blev fundet ved hjælp af fgetc og ungetc if's condition	count lines in file	fjernelse af disse
27. Brugervenlig layout af opskrifter til retter	recipe data handling	Formatering og bedre brug eller kontrol af længere ord således der ikke er utilsigtede linjeskift
28. ikke en funktionel funktion	delete line from file	Mange fejler gennem programmet hvor det ikke lykkedes at finde en løsning
29. Funktion der tillader brugeren at gennemse databasen efter en enkelt ret personen kunne ønske sig at spise eller tilføje il sin madplan	NaN	En ny funktion ville være nødvendig for dette, den skulle have adgang til både madplanen og alle retters ID og næringsindhold, kunne potentielt prompte brugeren hvis retten sætter dem over deres anbefalede energi mængde
30. For lille databse	NaN	Udvidelse af database, forhåbentlig via en API eller lign som kan automatisere processen og blive evt. eksternt ved lige holdt.
31. Grafiks brugerflade	NaN	Ville kræve enorm meget arbejde, og ville have været nemmere at implementere i et andet sprog med vores nuværende evner, f.eks. c-sharp eller c++

Table 3.1: Programvurdering og forbedringer.

Noget som der kunne tilføjes til denne liste, men som af praktiske årsager ikke står der på. Er at der ved 46 steder med scanf igennem programmet, ikke bliver lavet et tjek for om de returnerer den forventede værdi efter de er blevet udført.

I problemformuleringen fremgår det at programmet skal være en kombination af en madplanlægger og en kalorieberegner. Uden at tage nødvendig tid i form af bogføring af data. Kombinationen af de to typer ses i udførelsen og implementeringen af programmet. Kombinationen af de to typer har dertil forår-

saget at programmet nu kun behøver en oprettelse af individets profil. Dette har medført at programmet behøver brugerinput i form af aktivitetsniveau, vægt, højde og alder for at programmet kan beregne individets næringsbehov og sammensætte brugerens diæt. Ved fjernelsen af bogføringsproblemet fra kalorieberegnerne, er tiden der skal bruges på programmet forkortet kraftigt. Dertil er der ved kombinationen af de to typer, sket en formindskelse af den viden om ernæring og beregning af kulhydrater, brugeren førhen var nødsaget til at have. Programmet beregner nemlig individets kaloriebehov og giver brugeren alle de informationer der er nødvendige for at forstå de informationerne programmet printer. Motivationen vil med dette, derfor med alt sandsynlighed bevares for de fleste brugere. Motivations faktoren kunne dog hæves en del i form af en bedre GUI og interaktion med brugeren. Dette er ikke blevet implementeret i programmet grundet tidsmæssige årsager. Med alt dette opfylder programmet altså problemformuleringen. Der kunne dertil være blevet lavet nogle forbedringer i programmet som bliver forklaret nærmere i kritik og vurdering af programmet.

### 3.5 Konklusion

Overvægtige og deres situation kan kun betragtes som meget svær i dagens Danmark. Mange forstår ikke deres situation, og der forefindes ikke videnskabelig evidens på hvorfor individer bliver overvægtige. Dette skaber en lang række sociale represalier, hvor overvægtige kæmper med deres vægt, og samfundet omkring dem. Samtidig øger de 47.5% de statslige omkostninger til sundhedssektoren, med mange millioner på årlig basis. Hele denne situation skaber kravet om programmer, der kan gøre det lettere for både normal- og overvægtige at henholdsvis bibeholde og tabe vægt, da dette både vil give anledning til bedre helbred hos befolkningen og skabe en bedre samfundsøkonomi, hvor disse økonomiske midler kan anvendes til andet en behandling af syge.

De tilstedeværende teknologier som Madlog.dk og Fitness Meal Planner, danner begge gode rammer om hvordan sådanne programmer kan udføre og implimenteres i samfundet. De nyder stor popularitet blandt den danske befolkning, og hjælper mange individer med at bibeholde og tabe vægt. Dog kunne vi konstatere at hovedargumenterne fra interessant analysen lød på at tidsskravene på brugen af sådanne programmer, til tider er en medfaktor i faldende motivation, pga. programmerne kræver for meget tid at anvende effektivt i dagligdagen.

Ud fra disse oplysninger, er der blevet designet et program, der forsøger at minimere tidsforbruget ved anvendelse af denne type programmer, men stadig bibeholder funktionaliteten fra Madlog.dk og Fitness Meal Planner. Pga. manglende tid, har det ikke været muligt at teste programmet op imod en repræsentativ gruppe af brugere, og derfor kan det ikke konkluderes om programmet bibeholder samme funktionalitet som Madlog.dk og Fitness Meal Planner. Dog fungerer 80% af programmet, og denne del tager betydeligt mindre tid at anvende i forhold til Madlog.dk, som var gruppens benchmark.

### 3.6 Perspektivering

Programmet tager udgangspunkt i allerede implementerede ideer fra Kalorieberegner og Madplanlæggere. Programmets kalorieberegner funktion kan i sine stadier sammenlignes med kalorieberegneren fra Madlog, hvor programmet beregner individets daglige behov, men i stedet for at gengive informationerne til brugeren, som med Madlog, sender programmet informationerne forbi brugeren og anvender dem til udregning af en Madplan. Derved behøver brugeren ikke at tænke på næringsindholdet i den mad brugeren spiser, men blot lave maden efter den specificerede opskrift.

Samtidig kan Programmets madplans funktion også sammenlignes med Fitness Meal Planners madplan, hvor Fitness Meal Planner sammensætter en madplan baseret på brugerens input, med de givne informationer kan Fitness Meal Planner, i modsætningen til Madlog.dk, også opstille en indkøbsliste. Dog skalere og tilpasser programmet ikke dets opskrifter til brugerens behov, ved at gøre dette, kan programmet tilbyde en bredere vifte af opskrifter under forskellige omstændigheder. Samtidig genereres indkøbslisten automatisk efter madplanlægnings algoritmen har kørt, og ligger klar til at sende informationerne videre til brugeren.

Disse forskellige dele, reducere den totale tid brugeren benytter på at anvende programmet, og optimere dermed tiden brugeren har til andre ting i dagligdagen. Set i relation til vores problemformulering, kan dette potentielt betyde at motivationen for vægttab bevares. Dette kan ses i bredere forstand, hvor det også kan have en positiv impakt på samfundet, hvor programmet i sine dele, kan være med til at hjælpe på fedmeepidemien, og dermed lette trykket på udgifterne til behandling af overvægtige.

## Bibliography

- [1] 8fit. <http://8fit.com/>, 2015. Accessed: 16-11-2015.
- [2] Fitnessmealplanner. <http://www.fitnessmealplanner.com/>, 2015. Accessed: 16-11-2015.
- [3] Energy metabolism during the postexercise recovery in man. <http://ajcn.nutrition.org/content/42/1/69.abstract>, 1985. Accessed: 16-11-2015.
- [4] Organization WH. [http://apps.who.int/bmi/index.jsp?introPage=intro\\_3.html](http://apps.who.int/bmi/index.jsp?introPage=intro_3.html), ??? Accessed: 07-12-2015.
- [5] 3 Keys To Dialing In Your Macronutrient Ratios. <http://www.bodybuilding.com/fun/macro-math-3-keys-to-dialing-in-your-macro-ratios.html>, 2015. Accessed: 16-11-2015.
- [6] Bulking for Ectomorphs. [http://www.bodybuilding.com/fun/ectomorph\\_diet\\_bulking\\_plan.htm](http://www.bodybuilding.com/fun/ectomorph_diet_bulking_plan.htm), 2015. Accessed: 16-11-2015.
- [7] Effects of exercise on appetite control: loose coupling between energy expenditure and energy intake. <http://europepmc.org/abstract/med/9778093>, 1998. Accessed: 16-11-2015.
- [8] Sociologiens bidrag til forståelse af ernæring og kroniske sygdomme. [http://www.frederiksborg-gymhf.dk/Files/elever/vejledninger/gym/AT/AT14/files/ressourcerummet/files/130096\\_sociologi\\_ernaering.pdf](http://www.frederiksborg-gymhf.dk/Files/elever/vejledninger/gym/AT/AT14/files/ressourcerummet/files/130096_sociologi_ernaering.pdf), 2009. Accessed: 16-11-2015.
- [9] iform.dk forum. <http://iform.dk/forum>, 2015. Accessed: 16-11-2015.
- [10] LIISA LÄTHEENMÄKI OG KLAUS G. GRUNERT INGE HUMMELSHØJ HANSEN, KRISTINA AACHMANN. Danskernes forståelse af "de otte kostråd". Technical Report 022, Aarhus University, May 2013.
- [11] O'Doherty KoLH. *Hvad er "rigtig mad"?* Munksgaard, 1. edition, 2003.

- [12] Madlog. <http://www.madlog.dk/da/>, 2015. Accessed: 16-11-2015.
- [13] Madlogdatabase. <http://www.madlog.dk/index.php?id=281>, 2015. Accessed: 16-11-2015.
- [14] motion-online forum. <http://www.motion-online.dk/fora/index.php?act=idx>, 2015. Accessed: 16-11-2015.
- [15] Long-term weight loss after diet and exercise: a systematic review. <http://www.nature.com/ijo/journal/v29/n10/full/0803015a.html>, 2005. Accessed: 16-11-2015.
- [16] Følger af fedme. <http://www.netdoktor.dk/overvaegt/artikler/fedefolger.htm>, 2008. Accessed: 16-11-2015.
- [17] World Health Organization. Obesity, preventing and managing the global epidemic. Technical Report 894, World Health Organization, March 2000.
- [18] Opgaveløsning: Skudårskfunktionen. <http://people.cs.aau.dk/~normark/impr-c/functions-ekstr-opg-solution-1.html>, 2015. Accessed: 18-12-2015.
- [19] Noom Coach. <https://play.google.com/store/apps/details?id=com.wsl.noom>, 2015. Accessed: 16-11-2015.
- [20] Fakta om sundhedsvæsenet - sundhedsvæsenet i tal. <http://www.regioner.dk/aktuelt/temaer/fakta+om+regionernes+effektivitet+og+%C3%B8konomi/kopi+af+fakta+om+sundhedsv%C3%A6senet>, 2011. Accessed: 09-12-2015.
- [21] Evidence-Based Strategies in Weight-Loss Mobile Apps. <http://www.sciencedirect.com/science/article/pii/S0749379713004261>, 2015. Accessed: 13-12-2015.
- [22] Overvægtige danskeres lades i stikken. <http://www.science.ku.dk/presse/nyhedsarkiv/2012/overvaegtige-danskere/>, 2012. Accessed: 16-11-2015.
- [23] Overvægt og fedme. [http://www.si-folkesundhed.dk/upload/kap\\_21\\_overv%C3%A6gt\\_og\\_fedme.pdf](http://www.si-folkesundhed.dk/upload/kap_21_overv%C3%A6gt_og_fedme.pdf), 2007. Accessed: 09-12-2015.
- [24] Strava App. [www.strava.com](http://www.strava.com), 2015. Accessed: 03-11-2015.
- [25] De Ti Kostråd. <https://sundhedsstyrelsen.dk/da/nyheder/2013/de-ti-kostraad>, 2011. Accessed: 07-12-2015.
- [26] 10 veje til vægttab. <http://sundhedsstyrelsen.dk/da/sundhed/-/media/20D00E6CC2284992AE2C3650FE9CF911.ashx>, 2014. Accessed: 16-11-2015.
- [27] Supertracker. <https://www.supertracker.usda.gov/default.aspx>, 2015. Accessed: 16-11-2015.
- [28] Supertracker contact page. <https://www.supertracker.usda.gov/contactus.aspx>, 2015. Accessed: 16-11-2015.

- 
- [29] Building muscle: nutrition to maximize bulk and strength adaptations to resistance exercise training. <http://www.tandfonline.com/doi/full/10.1080/17461390801919128>, 2008. Accessed: 16-11-2015.
- [30] Myths, Presumptions, and Facts about Obesity. <http://www.nejm.org/doi/full/10.1056/NEJMsa1208051#t=article>, 2013. Accessed: 16-11-2015.



## Appendix

WHO klassifikation	Alternativ benævnelse	BMI (kg/m <sup>2</sup> )	Helbredsrisiko
Undervægt		< 18,5	Afhænger af årsagen til undervægten
Normalvægt		18,5 - 24,9	Normal
Overvægt		≥ 25,0	
Moderat overvægt		25,0 - 29,9	Let øget
Svær overvægt	Fedme	≥ 30,0	Middel til meget øget
Klasse 1	Fedme	30,0 - 34,9	Middel øget
Klasse 2	Svær fedme	35,0 - 39,9	Kraftigt øget
Klasse 3	Ekstrem svær fedme	≥ 40,0	Ekstremt øget

Table A.1: Bilag 1.

Rangordning af måltider	Rangordning og kombination af fødevarer
1. Middag/aftensmad	Kødet navngiver hovedretten Grøntsager Kartofler (erstatte kornprodukter)
2. Frokost	Pålægsprodukter navngiver <b>maden</b> Grøntsags- oste, ægge-, fiske- eller kødprodukter Mindre synlig del: rugbrød
3. Morgenmad	Kornprodukter navngiver retten og udgør den dominerende bestandel

Table A.2: Bilag 2.

*Hvordan kan madplaner og kalorieberegneres kombineres, så de ikke tager tid ud af dagligdagen og bibeholder motivationen for et vægttab?*

Figure A.1: Bilag 8.

Programtype	Specifikke valgte programmer
Kalorieberegner	Madlog Supertracker
Madplanlæggere	Fitness Meal Planner 8Fit
AI Coaches	Noom Strava
Fora	Motion-online Iform

Table A.3: Bilag 3.

Program	Mad-database	Motionsplanlægger	Coach function	Informationsforum	Madplanlægger	Dagligt tidsforbrug
Madlog	X	X				20 -30 min
SuperTracker	X			X		20 -30 min

Table A.4: Bilag 4.

Program	Mad-database	Motionsplanlægger	Coach function	Informationsforum	Madplanlægger	Dagligt tidsforbrug
Fitness Meal Planner		X			X	5 - 10 min hver uge
8Fit		X			X	2 - 7 min pr. dag

Table A.5: Bilag 5.

Program	Mad-database	Motionsplanlægger	Coach function	Informationsforum	Madplanlægger	Dagligt tidsforbrug
Motion-Online				X		Ikke relevant
Iform				X		Ikke relevant

Table A.6: Bilag 6.

Program	Mad-database	Motionsplanlægger	Coach function	Informationsforum	Madplanlægger	Dagligt tidsforbrug
Noom	X	X	X	X		10 min setup
Strava		X	X			10 min setup

Table A.7: Bilag 7.

```

23 struct profile_info
24 {
25     char first_name[MAX_STRING_LGT];
26     char last_name[MAX_STRING_LGT];
27     char gender[MAX_STRING_LGT];
28     int age;
29     int height;
30     double weight;
31     double calorie_need;
32     int protein_need;
33     int carbon_hydrates_need;
34     int fat_need;
35     int activity_level;
36     int nutrition_choice;
37 }user_profile;

```

Figure A.2: Bilag 9.

```

142 void new_or_old_profile()
143 {
144     int new_or_old_profile_answer;
145     printf("My friend, let me know if you want to create a new profile or if you want to use an old one!\n");
146     printf("New: 0; y: already have a profile and want to use it; old: already have a profile and want to use it\n");
147     scanf("%i", &new_or_old_profile_answer);
148     while(new_or_old_profile_answer != 0 || new_or_old_profile_answer != 1 || new_or_old_profile_answer != 2)
149     {
150         printf("Invalid input!\n");
151     }
152     else
153     {
154         switch(new_or_old_profile_answer)
155         {
156             case 'y': case '1':
157                 user_profile_loader();
158                 break;
159             case '0': case '2':
160                 new_user_basic_input();
161                 break;
162         }
163     }
164 }

```

Figure A.3: Bilag 10.

```

165 void new_user_basic_input()
166 {
167     int i = 0;
168     printf("Please enter your name and surname!\n");
169     printf("Last name: ");
170     scanf("%s", user_profile.first_name);
171     printf("First name: ");
172     scanf("%s", user_profile.last_name);
173     printf("Gender: ");
174     scanf("%s", user_profile.gender);
175     printf("Age: ");
176     scanf("%i", &user_profile.age);
177     while(i != 0)
178     {
179         if(strcmp(user_profile.gender, "m") != 0 || strcmp(user_profile.gender, "f") != 0 || strcmp(user_profile.gender, "o") != 0)
180         {
181             i = 1;
182         }
183         else if(strcmp(user_profile.gender, "m") != 0 || strcmp(user_profile.gender, "f") != 0 || strcmp(user_profile.gender, "o") != 0)
184         {
185             printf("Invalid input!\n");
186             printf("Please enter your gender again!\n");
187             scanf("%s", user_profile.gender);
188         }
189     }
190 }

```

Figure A.4: Bilag 11.

```

270 void profile_saver()
271 {
272     FILE *file_pointer;
273
274     file_pointer = fopen("test.txt", "w");
275     if(file_pointer == NULL)
276     {
277         print_error("FILE NOT FOUND", "FILE_SAVER", "FATAL");
278     }
279
280     fprintf( file_pointer, "%s\n%s\n%s\n%d\n%d\n%d\n%d\n%d\n",
281             user_profile.first_name, user_profile.last_name, user_profile.gender, user_profile.age, user_profile.height,
282             user_profile.weight, user_profile.calorie_need, user_profile.protein_need, user_profile.carbon_hydrates_need,
283             user_profile.fat_need, user_profile.activity_level, user_profile.nutrition_choice);
284
285     fclose(file_pointer);
286 }

```

Figure A.5: Bilag 12.

```

287 void user_profile_loader()
288 {
289     FILE *file_pointer;
290     file_pointer = fopen("test.txt", "r");
291
292     if(fscanf( file_pointer, "%s\n%s\n%s\n%d\n%d\n%d\n%d\n%d\n",
293             user_profile.first_name, user_profile.last_name, user_profile.gender, &(user_profile.age), &(user_profile.height),
294             &(user_profile.weight), &(user_profile.calorie_need), &(user_profile.protein_need), &(user_profile.carbon_hydrates_need),
295             &(user_profile.fat_need), &(user_profile.activity_level), &(user_profile.nutrition_choice)) != 11)
296     {
297         print_error("INVALID ASSEMBLED", "FILE_LOADER", "FATAL");
298     }
299
300     printf("\n\nLoading profile file\n\n");
301     "NAME: %s\n",
302     "NAME: %s\n",
303     "Gender: %s\n",
304     "Age: %d\n",
305     "Height: %d\n",
306     "Weight: %d\n",
307     "Calorie Need: %d\n",
308     "Protein Need: %d\n",
309     "Carbon Hydrates: %d\n",
310     "Fat: %d\n",
311     "Activity Level: %d\n",
312     "Nutrition Choice: %d\n",
313     user_profile.first_name, user_profile.last_name, user_profile.gender, user_profile.age, user_profile.height,
314     user_profile.weight, user_profile.calorie_need, user_profile.protein_need, user_profile.carbon_hydrates_need,
315     user_profile.fat_need, user_profile.activity_level, user_profile.nutrition_choice);
316
317     fclose(file_pointer);
318     user_profile.calorie_need = calorie_need_calculator();
319     nutrition_distribution();
320 }

```

Figure A.6: Bilag 13.

```
350     switch(user_change_choice)
351     {
352         case 1:
353             printf("Please enter your new first name ");
354             scanf("%s", user_profile.first_name);
355             break;
356         case 2:
357             printf("Please enter your new last name ");
358             scanf("%s", user_profile.last_name);
359             break;
360         case 3:
361             printf("Please enter your new gender ");
362             scanf("%s", user_profile.gender);
363             break;
364         case 4:
365             printf("Please enter your new age ");
366             scanf("%d", &(user_profile.age));
367             break;
368         case 5:
369             printf("Please enter your new height ");
370             scanf("%d", &(user_profile.height));
371             break;
372         case 6:
373             printf("Please enter your new weight ");
374             scanf("%lf", &(user_profile.weight));
375             break;
376         case 7:
377             printf("Please enter your new activity level");
378             scanf("%d", &(user_profile.activity_level));
379             break;
380         case 8:
381             nutrition_distribution(1);
382             break;
383         default:
384             print_error("INPUT ERROR!", "PROFILE CHANGER", "IRRELEVANT");
385     }
386     nutrition_distribution(0);
387     profile_saver();
388 }
389
```

Figure A.7: Bilag 14.

```

847 double calorie_need_calculator()
848 {
849     double tmp;
850
851     if(strcmp(user_profile.gender, "m") == 0 || strcmp(user_profile.gender, "M") == 0)
852     {
853         if(user_profile.age < 5)
854         {
855             tmp = ((1.16 * (user_profile.weight) + (1.5 * (user_profile.height / TO_PERCENT)) - 5.0) * KCAL_CONSTANT);
856         }
857         else if(user_profile.age >= 5 && user_profile.age < 10)
858         {
859             tmp = ((1.16 * (user_profile.weight) + (1.1 * (user_profile.height / TO_PERCENT)) + 0.7) * KCAL_CONSTANT);
860         }
861         else if(user_profile.age >= 10 && user_profile.age < 18)
862         {
863             tmp = ((1.16 * (user_profile.weight) + (1.1 * (user_profile.height / TO_PERCENT)) + 0.4) * KCAL_CONSTANT);
864         }
865         else if(user_profile.age >= 18 && user_profile.age < 30)
866         {
867             tmp = ((1.16 * (user_profile.weight) + (1.1 * (user_profile.height / TO_PERCENT)) + 0.4) * KCAL_CONSTANT);
868         }
869         else if(user_profile.age >= 30 && user_profile.age < 40)
870         {
871             tmp = ((1.16 * (user_profile.weight) + (1.1 * (user_profile.height / TO_PERCENT)) + 0.4) * KCAL_CONSTANT);
872         }
873         else if(user_profile.age >= 40)
874         {
875             tmp = ((1.16 * (user_profile.weight) + (1.1 * (user_profile.height / TO_PERCENT)) + 0.4) * KCAL_CONSTANT);
876         }
877         else
878         {
879             printf("ERROR: tmp is NULL with user age %d\n", user_profile.age);
880         }
881     }
882 }

```

Figure A.8: Bilag 15.

```

916 switch(user_profile.activity_level)
917 {
918     case 1:
919         tmp *= 1.1;
920         break;
921     case 2:
922         tmp *= 1.2;
923         break;
924     case 3:
925         tmp *= 1.3;
926         break;
927     case 4:
928         tmp *= 1.4;
929         break;
930     case 5:
931         tmp *= 1.5;
932         break;
933     case 6:
934         tmp *= 1.6;
935         break;
936     case 7:
937         tmp *= 1.7;
938         break;
939     case 8:
940         tmp *= 1.8;
941         break;
942     case 9:
943         tmp *= 1.9;
944         break;
945 }

```

Figure A.9: Bilag 16.



```

485 void main_menu_help_page()
486 {
487     FILE *fp = fopen("generalhelp.txt", "r");
488
489     help_options help;
490
491     answerswitch(help, fp);
492
493 }

```

Figure A.12: Bilag 19.

```

496 void answerswitch(help_options help, FILE *fp)
497 {
498     unsigned int i;
499     printf("FAQ and general Q&A.\n\n");
500     "Please make you selection\n";
501     "Access FAQ \t\t[FAQ]\n";
502     "Additional help \t[HELP]\n";
503     "Return to main menu \t[QUIT]\n";
504     ": ";
505     scanf("%s", help.answer);
506
507     for(i = 0; i < strlen(help.answer); i++)
508     {
509         help.answer[i] = toupper(help.answer[i]);
510     }
511
512     if(!(strcmp(help.answer, "faq") || !(strcmp(help.answer, "FAQ"))))
513     {
514         system("cls");
515         FAQ(help, fp);
516     }
517     else if(!(strcmp(help.answer, "help") || !(strcmp(help.answer, "HELP"))))
518     {
519         system("cls");
520         general_help(help, fp);
521     }
522     else if(!(strcmp(help.answer, "quit") || !(strcmp(help.answer, "QUIT"))))
523     {
524         system("cls");
525         return;
526     }
527     else
528     {
529         system("cls");
530         print_error("INPUT ERROR", "HELP OPTIONS", "IRRELEVANT");
531         answerswitch(help, fp);
532     }
533     return;
534 }

```

Figure A.13: Bilag 20.



```
540 void FAQ(help_options help, FILE *fp)
541 {
542     unsigned int i = 0;
543     printf("\nFrequently Asked Questions (FAQ)\n"
544           "please make your selection\n\n"
545           "User profile \t [USER]\n"
546           "General use \t [GENERAL]\n");
547     scanf("%s", help.choice);
548
549     for(i = 0; i < strlen(help.choice); i++)
550     {
551         help.choice[i] = toupper(help.choice[i]);
552     }
553
554     if(!(strcmp(help.choice, "GENERAL")))
555     {
556         general_FAQ(help, fp);
557     }
558     else if(!(strcmp(help.choice, "USER")))
559     {
560         user_FAQ(help, fp);
561     }
562     else
563     {
564         print_error("INPUT ERROR!", "FAQ", "IRRELEVANT");
565
566         FAQ(help, fp);
567     }
568 }
```

Figure A.14: Bilag 21.

```

571 void general_FAQ( help_options help, FILE *fp)
572 {
573     system("cls");
574     printf("General Sample Question 1\n\n");
575     printf("General Sample Question 2\n\n");
576     printf("General Sample Question 3\n\n");
577     printf("General Sample Question 4\n\n");
578     printf("General Sample Question 5\n\n");
579     printf("Quit          0\n\n");
580     printf("Please make your selection\n: ");
581     scanf("%d", &help.question);
582     if(! (help.question))
583     {
584         return;
585     }
586     else
587     {
588         switch(help.question)
589         {
590             case 1:
591                 system("cls");
592                 printf("\nGeneral Sample Answer 1\n\n");
593                 printf("Please make your selection\n");
594                 printf("Return to FAQ \t\t[FAQ]\n");
595                 printf("Return to General \t[QUIT]\n");
596                 scanf("%s", help.answer);
597                 if(! (strcmp(help.answer, "yes") || ! (strcmp(help.answer, "YES"))))
598                 {
599                     system("cls");
600                     FAQ(help, fp);
601                 }
602                 else
603                 {
604                     system("cls");
605                     answerswitch(help, fp);
606                 }
607                 break;
608             case 2:
609                 system("cls");
610                 printf("\nGeneral Sample Answer 2\n\n");

```

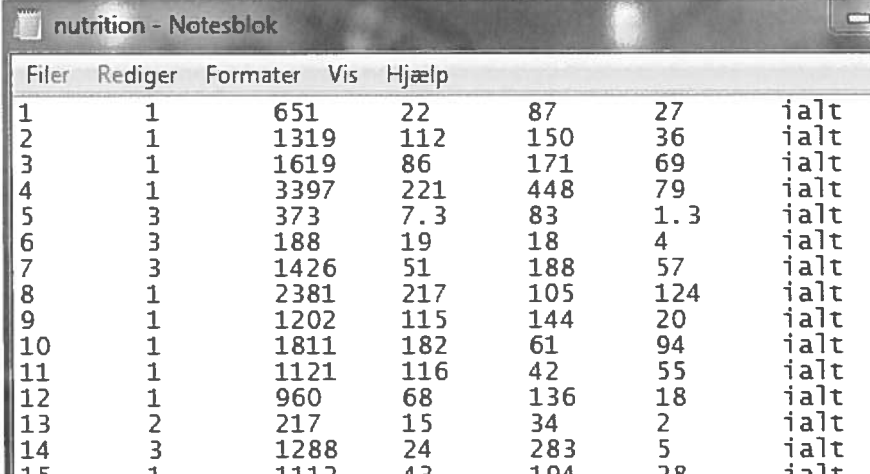
Figure A.15: Bilag 22.

```

819 void general_help( help_options help, FILE *fp)
820 {
821     char helptextfile;
822     errorgeneralhelp(help, fp);
823     while((helptextfile = fgetc(fp)) != EOF)
824     {
825         printf("%c", helptextfile);
826     }
827     fclose(fp);
828     printf("\nReturning to main menu. \n\n");
829     answerswitch(help, fp);
830 }

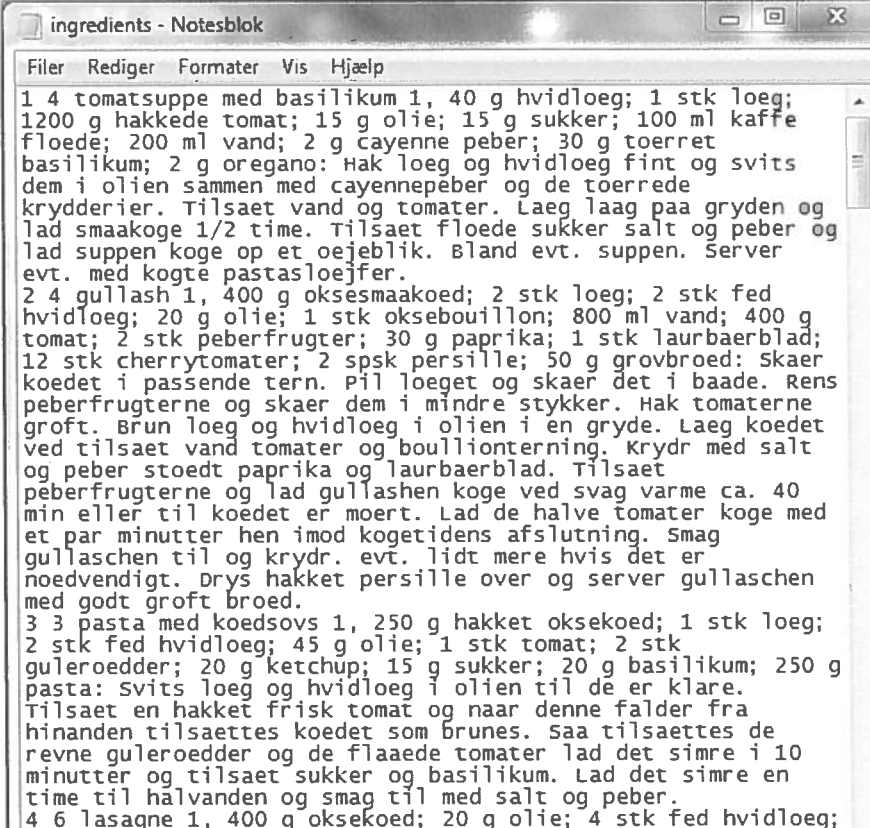
```

Figure A.16: Bilag 23.



Filer	Rediger	Formater	Vis	Hjælp		
1	1	651	22	87	27	ialt
2	1	1319	112	150	36	ialt
3	1	1619	86	171	69	ialt
4	1	3397	221	448	79	ialt
5	3	373	7.3	83	1.3	ialt
6	3	188	19	18	4	ialt
7	3	1426	51	188	57	ialt
8	1	2381	217	105	124	ialt
9	1	1202	115	144	20	ialt
10	1	1811	182	61	94	ialt
11	1	1121	116	42	55	ialt
12	1	960	68	136	18	ialt
13	2	217	15	34	2	ialt
14	3	1288	24	283	5	ialt
15	1	1117	47	104	70	ialt

Figure A.17: Bilag 24.



Filer	Rediger	Formater	Vis	Hjælp
<p>1 4 tomatuppe med basilikum 1, 40 g hvidloeg; 1 stk loeg; 1200 g hakkede tomat; 15 g olie; 15 g sukker; 100 ml kaffe floede; 200 ml vand; 2 g cayenne peber; 30 g toerret basilikum; 2 g oregano: Hak loeg og hvidloeg fint og svits dem i olien sammen med cayennepeber og de toerrede krydderier. Tilsaet vand og tomater. Laeg laag paa gryden og lad smaakoge 1/2 time. Tilsaet floede sukker salt og peber og lad suppen koge op et oejeblik. Bland evt. suppen. Server evt. med kogte pastasloejfer.</p> <p>2 4 gullash 1, 400 g oksesmaakoed; 2 stk loeg; 2 stk fed hvidloeg; 20 g olie; 1 stk oksebouillon; 800 ml vand; 400 g tomat; 2 stk peberfrugter; 30 g paprika; 1 stk laurbaerblad; 12 stk cherrytomater; 2 spsk persille; 50 g grovbroed: Skaer koedet i passende tern. Pil loeget og skaer det i baade. Rens peberfrugterne og skaer dem i mindre stykker. Hak tomaterne groft. Brun loeg og hvidloeg i olien i en gryde. Laeg koedet ved tilsaet vand tomater og boullionterning. Krydr med salt og peber stoedt paprika og laurbaerblad. Tilsaet peberfrugterne og lad gullashen koge ved svag varme ca. 40 min eller til koedet er moert. Lad de halve tomater koge med et par minutter hen imod kogetidens afslutning. Smag gullaschen til og krydr. evt. lidt mere hvis det er noedvendigt. Drys hakket persille over og server gullaschen med godt groft broed.</p> <p>3 3 pasta med koedsovs 1, 250 g hakket oksekoed; 1 stk loeg; 2 stk fed hvidloeg; 45 g olie; 1 stk tomat; 2 stk guleroedder; 20 g ketchup; 15 g sukker; 20 g basilikum; 250 g pasta: Svits loeg og hvidloeg i olien til de er klare. Tilsaet en hakket frisk tomat og naar denne falder fra hinanden tilsaettes koedet som brunes. Saa tilsaettes de revne guleroedder og de flaaede tomater lad det simre i 10 minutter og tilsaet sukker og basilikum. Lad det simre en time til halvanden og smag til med salt og peber.</p> <p>4 6 lasagne 1, 400 g oksekoed; 20 g olie; 4 stk fed hvidloeg;</p>				

Figure A.18: Bilag 25.

```
969 void count_file_lines(int *lines, int identifier[])
970 {
971     FILE *pfile;
972     char ch;
973     int i = 0;
974
975     pfile = fopen("ingredients.txt", "r");
976
977     if(pfile == NULL)
978     {
979         print_error("FILE NOT FOUND", "COUNT_FILE_LINES", "FATAL");
980         return;
981     }
982
983     while(!feof(pfile))
984     {
985         ch = fgetc(pfile);
986         if((ch == '\n'))
987         {
988             i++;
989         }
990         else if(ch == ';')
991         {
992             identifier[i]++;
993         }
994     }
995     *lines = i;
996     fclose(pfile);
997 }
```

Figure A.19: Bilag 26.

```

1003 int load_file(dish_data dish[], int *lines, int identifier[])
1004 {
1005     FILE *ptfile;
1006     int j, i;
1007
1008     ptfile = fopen("ingredients.txt", "r");
1009     if(ptfile == NULL)
1010     {
1011         print_error("FILE NOT FOUND", "LOAD_FILE", "FATAL");
1012         return(0);
1013     }
1014     else
1015     {
1016         for(i = 0; i < *lines + 1; i++)
1017         {
1018             fscanf(ptfile, "%d %d %[^,], %d ",
1019                 &dish[i].dish_id,
1020                 &dish[i].dish_amt,
1021                 dish[i].dish_name,
1022                 &dish[i].meal_time_id);
1023             for(j = 0; j < identifier[i] + 1; j++)
1024             {
1025                 fscanf(ptfile, "%lf %s %[^;];",
1026                     &dish[i].ingredients[j].ingamt,
1027                     dish[i].ingredients[j].ingvalue,
1028                     dish[i].ingredients[j].ingname);
1029             }
1030             fscanf(ptfile, ":%[^\n]", dish[i].recipe);
1031         }
1032     }
1033     fclose(ptfile);
1034     return(0);
1035 }
1036

```

Figure A.20: Bilag 27.

```

1044 void get_dish_info(dish_data dish[], int meal_id[], double meal_scaler[], int *lines, int identifier[])
1045 {
1046     int i, j, g, array_lgt = 0, k = 0;
1047     ingredients_data handler[MAX_STRUCT_LGT];
1048     ingredients_data shop_lst[MAX_STRUCT_LGT];
1049
1050     for(i = 0; i < MAX_MEALPLAN_LGT; i++)
1051     {
1052         if(meal_scaler[i] < 0)
1053         {
1054             print_error("INCORRECT SCALER", "CALC_INGREDIENTS_AMT", "FATAL");
1055         }
1056         else if(meal_id[i])
1057         {
1058             for(j = 0; j < *lines + 1; j++)
1059             {
1060                 if(meal_id[i] == dish[j].dish_id)
1061                 {
1062                     for(g = 0; g < identifier[j] + 1; g++, k++)
1063                     {
1064                         strcpy(handler[k].ingname, dish[j].ingredients[g].ingname);
1065                         strcpy(handler[k].ingvalue, dish[j].ingredients[g].ingvalue);
1066                         handler[k].ingamt = (meal_scaler[i] * dish[j].ingredients[g].ingamt);
1067                     }
1068                     break;
1069                 }
1070             }
1071         }
1072         else
1073         {
1074             print_error("INCORRECT ID", "CALC_INGREDIENTS_AMT", "NON_FATAL");
1075             break;
1076         }
1077     }
1078     array_lgt = k;
1079     sort_shp_lst(handler, shop_lst, array_lgt);
1080 }

```

Figure A.21: Bilag 28.

```

1086 void sort_shp_lst(ingredients_data handler[], ingredients_data shop_lst[], int array_lgt)
1087 {
1088     int i, j, g, identifier, shop_lst_size;
1089     shop_lst_size = 1;
1090
1091     for(j = 0, i = 0; j < array_lgt; j++)
1092     {
1093         if(handler[j].ingamt)
1094         {
1095             for(g = 0, identifier = 0; g < shop_lst_size; g++)
1096             {
1097                 if (!strcmp(shop_lst[g].ingname, handler[j].ingname))
1098                 {
1099                     identifier = 1;
1100                     shop_lst[g].ingamt += handler[j].ingamt;
1101                 }
1102             }
1103             if (!identifier)
1104             {
1105                 shop_lst[i++] = handler[j];
1106                 shop_lst_size++;
1107             }
1108         }
1109     }
1110     shop_lst_size--;
1111     qsort(shop_lst, shop_lst_size, sizeof(ingredients_data), compare_sort_shp_lst);
1112     print_func_ing(shop_lst, shop_lst_size, "sort_shp_lst");
1113 }

```

Figure A.22: Bilag 29.

```

1119 void recipe_data_handling(double meal_id[], dish_data dish[], int identifier[])
1120 {
1121     int i, j, g, id, recipe_list_size = 0;
1122     int ingredient_lgt[MAX_STRING_LGT];
1123     dish_data handler[MAX_MEALPLAN_LGT];
1124
1125     for(j = 0, i = 0; j < MAX_MEALPLAN_LGT; j++, i++)
1126     {
1127         if(meal_id[j])
1128         {
1129             for(g = 0, id = 0; g < MAX_MEALPLAN_LGT; g++)
1130             {
1131                 if(dish[g].dish_id == meal_id[j])
1132                 {
1133                     id = 1;
1134                     handler[i] = dish[g];
1135                     ingredient_lgt[i] = identifier[g];
1136                     break;
1137                 }
1138             }
1139             if(id)
1140             {
1141                 recipe_list_size++;
1142             }
1143         }
1144     }
1145     print_func_dish(handler, ingredient_lgt, recipe_list_size, "recipe_data_handling");
1146 }

```

Figure A.23: Bilag 30.

```

50  typedef struct
51  □{
52      double id, type, kcal;
53      double prot, carbs, fat;
54      char mass[MAX_STRING_LGT];
55  }meal;
56
57  typedef struct
58  □{
59      double id;
60      int ss, mm, hh, day, month, year;
61  }ntime;

```

Figure A.24: Bilag 31.

```

1506 void load_database(meal *m)
1507 □{
1508     /* Pick up data from the database... */
1509     FILE *input_file_pointer = fopen("Nutrition.txt", "r");
1510     int i;
1511     if(input_file_pointer != NULL)
1512     □{
1513         for(i=0; i < NUMBER_OF_DATABASE_ROWS; i++)
1514         □{
1515             fscanf(input_file_pointer,
1516                 " %lf %lf %lf %lf %lf %lf %s ",
1517                 &m[i].id, &m[i].type, &m[i].kcal, &m[i].prot,
1518                 &m[i].carbs, &m[i].fat, m[i].mass);
1519         }
1520         fclose(input_file_pointer);
1521     }
1522     else
1523     □{
1524         printf("Error loading file, bye\n");
1525         perror("Error");
1526     }
1527 }

```

Figure A.25: Bilag 32.

```

1432 func= determine_dish(mMeal, scale_factor, mMeal, exception_list(), mMeal, *exceptions_report, mMeal, meal_type, meal_out_mmeal, meal_m, mMeal, time_list)
1433 {
1434     // While proposed array (NUTRITION_CATEGORIES - NUT_CAT_EMB_SFF), tmp_scale.
1435     best_approximated_dish_id = -1, nutrition_array[NUTRITION_CATEGORIES - NUT_CAT_EMB_SFF],
1436     last_comparing_factor = 0, current_comparing_factor,
1437     // Adjusting last comparing factor with a temp value.
1438     // we can adjust our value as last comparing factor /
1439     if (i, j, exception = 0, deviation = 0)
1440     meal_proposed_mmeal;
1441     mMeal time_present = current_time();
1442
1443     // Initialize nutrition array, it's nutrition mMeal.
1444     struct_to_array(mMeal, nutrition_array);
1445
1446     for(i = 0; i <= NUMBER_OF_CATEGORIES; i++)
1447     {
1448         // let exception is a flag.
1449         // if exception = 0, the nutrition will pass for the current dish is 0.
1450         if(exception = 1, the nutrition "current" iteration is 1, will be skipped, as the 1 representing the current dish is,
1451         it is an "0" a dish that should not be chosen.
1452         exception = 0;
1453         if(*exceptions_report > 0)
1454         {
1455             for(j = 0; j < (first_exceptions_amount); j++)
1456             {
1457                 if(exception_list[j] == 1)
1458                 {
1459                     exception = 1;
1460                 }
1461             }
1462
1463             // This part doesn't work as intended.
1464             // If a dish is used more than 4 days ago, it can be
1465             // added multiple times.
1466             // Since the 4 days "maximum time" from food (or drink) let doesn't work.
1467             if((time_between(time_list[j], time_present) > 4) && (exception_list[j] == 1))
1468             {
1469                 exception = 1;
1470             }
1471         }
1472     }
1473
1474     // If the current dish should not be chosen, it of the "current dish" is if the "current dish"
1475 }

```

Figure A.26: Bilag 33.

```

1457 {
1458     // If we are looking for a breakfast meal, and current dish is of type dinner -> skip (current dish)
1459     if(exception == 0) { if (meal_type != m[i].type)
1460     {
1461         continue;
1462     }
1463
1464     // Get (current dish) struct mMeal from Database.
1465     // Fill out proposed dish array with information from one specific dish.
1466     // From Database.
1467     proposed_mmeal = m[i];
1468
1469     // Transfer our proposed array dish to store information from our proposed meal.
1470     struct_to_array(proposed_mmeal, proposed_array_dish);
1471
1472     // Divide the proposed dish array, to fit our nutrition needs (from nutrition_array).
1473     // The desired scale value is found by comparing the total kcal value of our nutrition_array and from our proposed_dish_array.
1474     tmp_scale = scale_array(nutrition_array, proposed_array_dish, (int)meal_type);
1475
1476     // Get the average comparing factor by comparing every value in the two arrays, with each other.
1477     // Get a get comparing factor function.
1478     current_comparing_factor = get_comparing_factor(nutrition_array, proposed_array_dish, deviation);
1479
1480     // Compare the two arrays.
1481     // If the current dish's comparing factor is lower than the last
1482     // one, the dish is a better match to our goal values of nutrition_array.
1483     // If the deviation is too big
1484     // (also, the definition of "too big deviation" is
1485     // defined as a global variable MAX_DEVIATION.
1486     if((current_comparing_factor < last_comparing_factor) && !(deviation))
1487     {
1488         last_comparing_factor = current_comparing_factor;
1489         // Give the id of the current dish.
1490         // If no better dish is found, this dish's id will be returned as the best approximated dish.
1491         best_approximated_dish_id = (double)i;
1492         // Store information about current scale factor of current dish, to the scale_factor pointer.
1493         // If this dish happens to be the final (best approximated) dish, the scale_factor will be
1494         // overwritten anymore, and the value is safely stored.
1495         *scale_factor = tmp_scale;
1496     }
1497     deviation = 0;
1498 }

```

Figure A.27: Bilag 34.

17	2	312	31	32	6.5	ialt
18	2	9806	345	1673	154	ialt
19	2	1368	40	139	76	ialt
20	2	301	39	10	12	ialt
21	2	1771	116	193	58	ialt

Figure A.28: Bilag 35.



```

1578 double get_comparing_factor(const double const_array[], double compare_array[], int *deviation)
1579 {
1580     int i;
1581     double comparing_factor = 0;
1582     /* Initializing i to nutrition category start skip + 1, to make it ok; kcal comparing */
1583     for(i = NUT_CAT_START_SKIP + 1; i < NUTRITION_CATEGORIES - NUT_CAT_END_SKIP; i++)
1584     {
1585         /* Get the percentage deviation */
1586         comparing_factor += (sqrt((const_array[i] - compare_array[i])*(const_array[i] - compare_array[i])) ) / const_array[i]);
1587         if (get_is_deviating((double)const_array[i], (double)compare_array[i]))
1588             *deviation = 1;
1589     }
1590     /* Find the average of all the percentage deviations of the two arrays */
1591     comparing_factor /= (NUTRITION_CATEGORIES - 2);
1592     return comparing_factor;
1593 }

```

Figure A.29: Bilag 36.

```

1602 int get_is_deviating(double static_value, double test_value)
1603 {
1604     double c;
1605     int result = 0;
1606
1607     /*Get the absolute value of the difference between static_value and test_value*/
1608     c = sqrt((static_value - test_value) * (static_value - test_value));
1609     c /= static_value;
1610
1611     /* If the deviation of the two values are greater than MAX_DEVIATION -> deviation = true. */
1612     if (c > MAX_DEVIATION)
1613     {
1614         result = 1;
1615     }
1616     return result;
1617 }

```

Figure A.30: Bilag 37.

```

1685 ntime current_time()
1686 {
1687     time_t timer;
1688     ntime t;
1689     char cur_time[26];
1690     struct tm* tm_info;
1691     time(&timer);
1692     tm_info = localtime(&timer);
1693
1694     strftime(cur_time, 26, "%H:%M:%S - %d:%m:%Y ", tm_info);
1695     sscanf(cur_time, "%d:%d:%d - %d:%d:%d ",
1696            &t.hh, &t.mm, &t.ss,
1697            &t.day, &t.month, &t.year);
1698     t.id = 0;
1699     return t;
1700 }

```

Figure A.31: Bilag 38.

```

1853 int is_leap_year(int year)
1854 {
1855     return (year % 400 == 0) || ((year % 4 == 0) && (year % 100 != 0));
1856 }

```

Figure A.32: Bilag 39.

```

1537 double scale_array(double const_array[], double target_array[], int meal_type)
1538 {
1539     int i;
1540     double scale_factor, new_target_value, modifier;
1541     switch(meal_type)
1542     {
1543     case 0:
1544         modifier = 1;
1545         break;
1546     case 1:
1547         modifier = MEAL_SHARE_DINNER;
1548         break;
1549     case 2:
1550         modifier = MEAL_SHARE_LUNCH;
1551         break;
1552     case 3:
1553         modifier = MEAL_SHARE_BREAKFAST;
1554         break;
1555     }
1556     /* Calculate how much we should scale target_array */
1557     /* Based on the const array total kcal */
1558
1559     scale_factor = (const_array[2] * modifier) / target_array[2];
1560     /* Scale every value in the target_array */
1561     for(i = NUT_CAT_START_SKIP; i < NUTRITION_CATEGORIES - NUT_CAT_END_SKIP; i++)
1562     {
1563         new_target_value = (double)target_array[i] * scale_factor;
1564         target_array[i] = new_target_value;
1565     }
1566     return scale_factor;
1567 }

```

Figure A.33: Bilag 40.

```

1375 void create_meal_composition(int meal_array[], double scale_array[], meal_nutrition_meal)
1376 {
1377     int i;
1378     double exceptions_count = 0, meal_type = 0, exception_list[50];
1379     char *file_name = "Food_Variations.txt";
1380     ntime except_time_list[50];
1381
1382     /* Nutrition Database List */
1383     meal_nut_db_list[NUTRITION_DATABASE_ROWS];
1384
1385     /* Find the amount of exceptions (eg. one exception = one id that should not be chosen as a dish) */
1386     exceptions_count = count_lines_in_file(file_name);
1387     load_database(NUTRITION_DATABASE_ROWS);
1388     load_time_data(file_name, except_time_list, exceptions_count);
1389     make_exception_array(exception_list, file_name, except_time_list, exceptions_count);
1390
1391     for(i = 0; i < MAX_MEALPLAN_LGT; i++)
1392     {
1393         meal_type = (i / 3) * 3;
1394         meal_array[i] = (int)determine_dish(scale_array[i], exception_list, exceptions_count,
1395                                         meal_type, nutrition_meal, NUTRITION_DATABASE_ROWS, except_time_list);
1396         check_add_exception(exception_list, meal_array[i], exceptions_count, file_name, except_time_list);
1397     }
1398 }

```

Figure A.34: Bilag 41.

```

1369 int check_add_exception(double exception_list[], double id, double *exception_list_count, char *file_name, ntime *t)
1370 {
1371     int return_value = 0;
1372
1373     /* If current id is not in exception list -> add it */
1374     load_time_data(file_name, t, exception_list_count);
1375
1376     /* If the current dish is not already in the exception list
1377     /> If the current id is above or greater to 0 (not -1)
1378     -> add the dish to the exception list, as it must be picked another time */
1379     if((get_id_index_in_list(id, t, exception_list_count) == -1) && (id >= 0))
1380     {
1381         write_timeline_to_file(file_name, id); /* Write a line of current date and time to the txt file "Food_Variations.txt" */
1382         *exception_list_count += 1; /* Add 1 to exception_list_count to keep track of number of exceptions */
1383         exception_list[(int)*exception_list_count - 1] = id; /* Add the id at the last slot of exception_list */
1384         load_time_data(file_name, t, exception_list_count); /* Update the "nlist.txt" list (list of dishes already picked as a variation) */
1385         return_value = 1;
1386     }
1387     /* If a new exception were added, return 1, else return 0 */
1388     return return_value;
1389 }

```

Figure A.35: Bilag 42.

```

Hey there! and welcome to the new and improved meal planner 1.0!
Now.. do you already have a profile set up and are ready to go? <Y/N> n
Welcome to the one time account / user setup
Please type in your name
Marco
Please type in your surname
Polo
What is your gender? <M/F>
M
How old are you? age between 0-99
30
How tall are you? in centimeters 30-272
180
What's your weight in kilograms? 30-500
60
How active would you say you are during the day from 1 to 14
1: Literally not moving, at all during the day
7: Average daily exercise
14: Heavy daily exercise
7
Now, what is your goal with your diet going forward?
Maintain weight:      [1]
Increase weight:      [2]
Lose weight:          [3]
3
What would you like to do?
Logout type:          logout
create the weeks shopping list: shop
change your profile type: profilechange
create the weeks's menu type: mealplan
view your profile type: viewprofile
if you need help please type: help
quit type:            quit

```

Figure A.36: Bilag 43.

Meal	Thursday
Breakfast	0.4 x ð30 ned ægslí
Lunch	2.0 x tonat ragout
Dinner	0.4 x Moerbrad ned pesto mozarella og saltkræfde tonster
Meal	Friday
Breakfast	3.0 x Frugtorgensmad
Lunch	0.4 x volderfsalat
Dinner	1.3 x Minestroneuppe ned kylling
13.5 stk	loeg
725.2 ml	naelk
25.2 g	oandler
6.2 g	mayonnaise
4.0 g	maskatnoed
300.2 g	nudler
446.2 g	oksesmaaknoed
41.4 ml	olie
3.3 g	oregano
4.2 g	paprika
33.2 g	parmasanost
6.5 g	peber
7.0 stk	peberfrugt
60.3 g	peberod
89.4 g	persille
2.3 stk	porre
15.1 stk	porloeg

Figure A.37: Bilag 44.