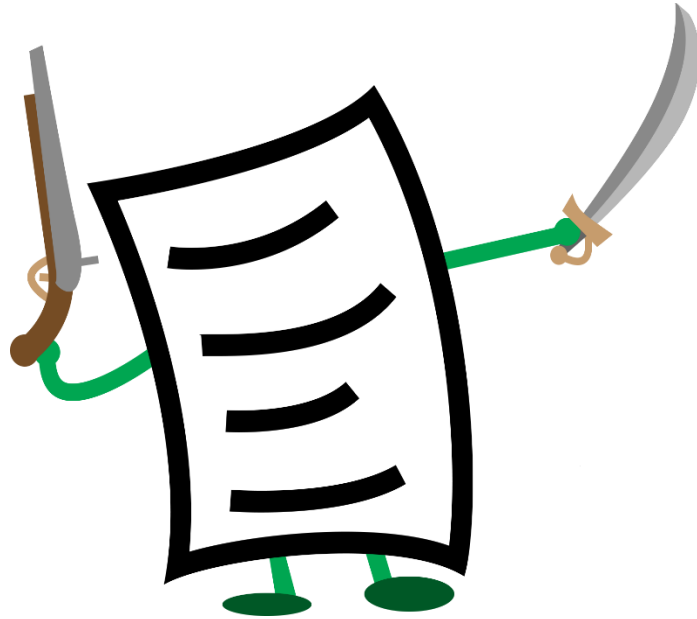


SPELMOTOR TXT

Program för att köra speläventyr från textfiler.



Eget projekt för kursen
Strukturerad Programmering C

Johan Kämpe

UIS16

2016-10-25

Sammanfattning

Programmet SPELMOTOR TXT används för att spela textäventyr som är lagrade i externa textfiler.

Programmet har stöd för val av spel, omstart av spel, och diverse extra-funktioner, så som formatering av spelets text.

Programmet navigerar i spelen med "identifikationsnummer" som skrivs i textfilerna.

En egen inkluderingsfil används med diverse funktioner för text- och textfilshantering. I programmets huvudfil finns tre stycken funktioner samt main-funktion.

Innehåll

1 Inledning	4
1.1 Syfte	4
1.2 Noteringar	4
1.3 Länkar	4
1.4 Filer	5
2 Genomförande och resultat	6
2.1 Använd programvara och litteratur	6
2.2 Avgränsningar och krav	6
2.3 Metod	7
2.3.1 Skrivning av källkod och kompilering	7
2.3.2 Utformning av textfiler för hållande av textäventyrspel	7
2.4 Programmens funktion	8
2.5 Textfilernas utförande	10
2.6 Debug-läge	12
2.7 Programmens kod	13
2.7.2 Variabler	15
2.7.3 Inkluderade filer och bibliotek	16
2.7.4 Funktioner	17
2.7.4 Main-funktionen	18
2.7.5 Funktionen listaVal()	20
3 Diskussion	21
4 Bilagor	23
Ändringslogg	23

1 Inledning

1.1 Syfte

Syftet med projektet är att skapa ett eget program för inlämning i kursen *Strukturerad Programmering C*. Projektet ska ses som träning och en sorts utmaning.

Hädanefter kommer projektet ibland att hänvisas till som "programmet".

Syftet med programmet *SPELMOTOR TXT* är att kunna spela olika textäventyr som är lagrade i externa textfiler. Programmet ska automatiskt leta upp de val som spelaren kan göra i de olika spelmomenten, och sedan gå till nästa steg i textäventyret beroende av valinmatning.

I de externa textfilerna ska det finnas möjlighet välja när spelet ska avslutas, det ska finnas funktionalitet för radbrytning i texten. Det ska också finnas möjlighet att utföra enklare kommandon genom att använda sifferkoder i textfilen, till exempel färgbyte av kommandofönstrets text.

Inspirationen för att starta projektet var de gruppuppgifter som utförts under kursen *Strukturerad Programmering C*, där grupper om cirka fem personer samarbetade för att skapa textäventyr i programspråket C. I dessa övningar skrevs all spel-text direkt i källkoden.

1.2 Noteringar

Rapporten är tänkt att läsas av personer med en grundläggande förståelse inom programspråket C. Ingen ordlista finns i rapporten. Flera standardfunktioner förklaras inte.

Visst innehåll i filerna *libtxt.c* & *libtxt.h* kommer ej behandlas i rapporten, detta bibliotek uppdaterades parallellt med utveckling av projektet *SPELMOTOR TXT*, och somliga funktioner i *libtxt.c* används ej i projektet.

1.3 Länkar

Programmet SPELMOTOR TXTs GitHub-sida

https://github.com/GoblinDynamiteer/spelmotor_txt

Funktionsbiblioteket libtxts GitHub-sida

<https://github.com/GoblinDynamiteer/libs>

1.4 Filer

Följande filer medföljer programmet:

- **spelmotortxt.c**
- **libtxt.c**
- **libtxt.h**
- **spel_default.txt**
- **spel_grupp2.txt**
- **spel_pirat.txt**

Filerna *libtxt.c* och *libtxt.h* innehåller diverse funktioner för text och textfiler, deklarationer, macron och standardbiblioteksinkluderingar. Textfilerna (med filändelse *txt*) är äventyrsspel som är tänkta att användas med programmet.

spel_default.txt är programmets standardspel, i vilket spelaren ikläder sig rollen som en brunbjörn i en skog.

spel_grupp2.txt är ett zombie-äventyr som påbörjades som ett grupprojekt av grupp 2 i kursen Strukturerad Programmering C. Spelet är delvis omskrivet och modifierat för att fungera med programmet.

spel_pirat.txt är ett spännande äventyr där spelaren får vara en piratkapten på 1700-talet.

För att programmet ska kunna hitta textfilerna att spela med vid körning, så måste de finnas i samma katalog som programfilen.

Vid kompilering av källkoden antas också att filerna *libtxt.h* och *libtxt.c* finns i samma katalog som *spelmotortxt.c*.

2 Genomförande och resultat

2.1 Använd programvara och litteratur

Programvaror

- Notepad++, texteditor
- Microsoft Word 2016
- Microsoft Notepad för Windows 10, texteditor
- Adobe Photoshop CC 2015, bildredigering
- GCC, GNU Compiler Collection, kompilator
- Git, versionshanteringsverktyg
- GitKraken, grafiskt gränssnitt för Git
- Snipping Tool (skärmsklippverktyget), Windows-program för selektiva skärmsklipp

Litteratur

- *C från början*, Jan Skansholm, 2016. – Boken kommer hänvisas till i texten som ”*boken*”

2.2 Avgränsningar och krav

Enligt specifikation från utbildningen ska följande krav uppfyllas:

- Programmets källkod ska ha en omfattning om minst 50 rader kod, och vara fördelat på minst två filer.
- Om redan färdig kod används från Internet, ska detta markeras i källkoden och kodens upphovsrättsman ska krediteras.
- Enklare dokumentation ska skapas, som beskriver projektets syfte och programmets funktionalitet.
- Inlämning ska ske på Moodle. Källkod, dokumentation och körbar exe-fil ska lämnas in.
- Deadline för projektet är den 30 oktober 2016.

Andra/egna avgränsningar:

- Programmet ska skrivas i programspråket C.
- Koden behöver enbart kunna kompileras och köras i en Windows-miljö.
- Programmet behöver enbart kunna användas med korrekt skrivna textfiler. Ingen kontroll ska utföras för att bekräfta att textfilen innehåller ett korrekt skrivet text-äventyr.

2.3 Metod

2.3.1 Skrivning av källkod och kompilering

Programmets källkod skrevs uteslutande i texteditorn *Notepad++*. Notepad++ har färgmallar för olika programspråk, vilket underlättar kodskrivandet och felsökning.

Notepad++ saknar inbyggt stöd för kompilering av kod, för att underlätta programmerandet används texteditorns "run"-funktion med följande inmatning:

```
cmd -cmd /K d: & cd "$(CURRENT_DIRECTORY)" & gcc "$(FULL_CURRENT_PATH)" libtxt.c -o  
"$(CURRENT_DIRECTORY)\$(NAME_PART)".exe & chcp 1252 & "$(CURRENT_DIRECTORY)\$(NAME_PART)".exe
```

Med detta kommando ges möjligheten att direkt kompilera den aktuella c-filen, tillsammans med *libtxt.c*, som innehåller funktioner för text- och filhantering. Efter kompilering startas exe-filen automatiskt.

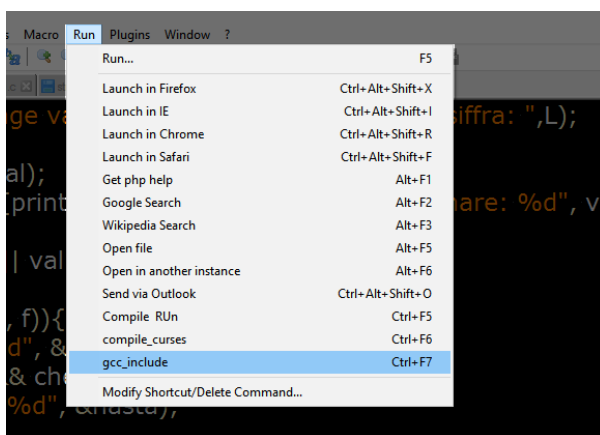


Bild 1 Run-menyn i programmet Notepad++

2.3.2 Utformning av textfiler för hållande av textäventyrspel

Innan arbetet med programmets källkod kunde påbörjas, skapades en första version av ett textäventyrsspel. Detta för att kunna anpassa programmet efter textfilens utformning.

Flera olika utföranden av textfilen testades parallellt med kodningen, innan det nuvarande utförandet fastställdes. Hur textfilerna med spel behöver utformas beskrivs senare i rapporten.

Windows-verktyget Notepad användes i första hand för att skriva textfilen.

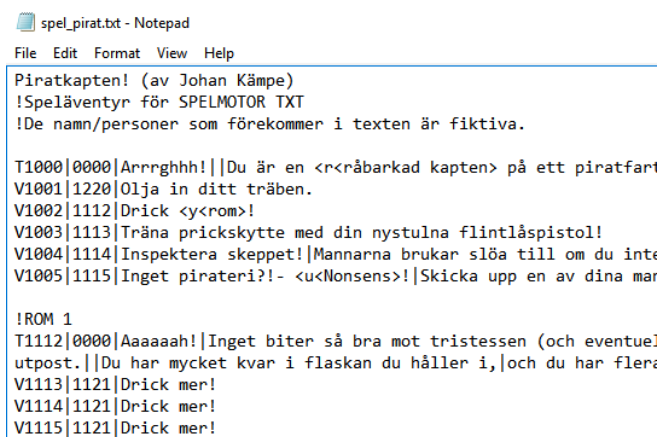
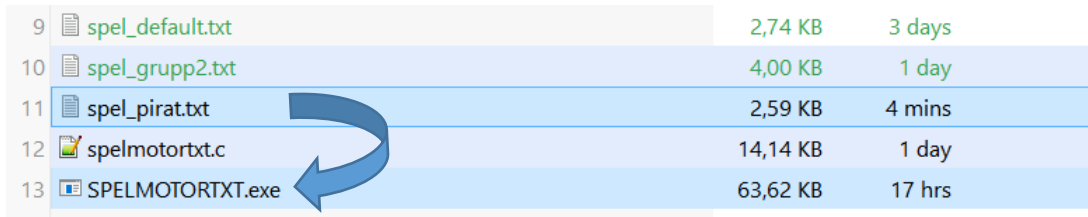


Bild 2 Texteditering med Notepad

2.4 Programmets funktion

Vid programmets start undersöks först om en fil har givits som ett argument. Detta görs genom att skriva *spelmotortxt.exe textfil.txt* i kommandotolken, där *textfil.txt* är den textfil som vill användas.

Det går också utföra detta genom att dra textfilen till programmets ikon, då startas programmet med den släppta textfilen som argument.



9	spel_default.txt	2,74 KB	3 days
10	spel_grupp2.txt	4,00 KB	1 day
11	spel_pirat.txt	2,59 KB	4 mins
12	spelmotortxt.c	14,14 KB	1 day
13	SPELMOTORTXT.exe	63,62 KB	17 hrs

Bild 3 Dra och släpp textfil på programikon för att starta det spelet.

Om inget argument har givits uppmanas användaren att skriva in namnet på den textfil denne vill använda att spela med. Om denna fil inte kan hittas kommer programmet att läsa in den fördefinierade textfilen *spel_default.txt*, som innehåller "Björnspelet". Om denna fil saknas eller inte kan läsas in avslutas programmet.

```
-----
SPELMOTOR TXT
-----
Ange filnamn (med filändelse) för textfil.
Mata in EOF (Ctrl+Z) för att avbryta
Om ingen fil anges laddas spel_default.txt
Textfil: spel_pirat.txt
```

Bild 4 Inmatning av textfil att spela med

Vid användarinmatning av textfil kan användaren välja att avsluta programmet med kommandot EOF (End Of File) vilket skrivs in med tangentkombinationen *CTRL+Z* i en Windows-miljö. Om *spel_default.txt* vill spelas kan användaren välja att inte skriva in något filnamn, eller skriva ett medvetet inkorrekt filnamn.

Om programmet lyckas läsa in en korrekt utformad textfil kommer dess innehållande spel att startas.

```
-----
Spel som kommer köras är:
Piratkapten! (av Johan Kämpe)
-----
Press any key to continue . . .
```

Bild 5 Korrekt inläst och startat spel

Textäventyrspelen fungerar på följande sätt:

- Initialt visas spelets titel, och användaren uppmanas att trycka på en valfri tangent för att starta spelet.
- En text visas på skärmen som beskriver spelarens situation, två eller fler val listas upp för spelaren, för att symbolisera hur denne vill agera i den givna situationen.
- Varje val som listas representeras av en siffra, från 1 och uppåt. Spelaren uppmanas att skriva in en siffra för att välja hur spelet ska fortskrida.

```
-----  
Du är en stor brunbjörn i en djup mörk skog, vad gör du?  
-----  
[1] - Ät blåbär!  
[2] - Gå i ide...  
[3] - Leta efter myror att kaka.  
[4] - Ryt!  
-----  
Ange val genom att slå in en siffra: 2
```

Bild 6 Textspel med val och användarinmatning, från spel_default.txt

- Om användaren slår in en siffra som inte har ett korresponderande val, eller slår in ett tecken eller text som inte börjar med en siffra, uppmanar programmet användaren att försöka igen, tills denne skriver in en korrekt siffra.
- Spelet går sedan vidare till en annan situation/text, beroende på vilket val som har utförts. Nya val listas för användaren.
- Denna process repeteras tills det att spelaren har antingen vunnit eller förlorat äventyret, det kan finnas flera vinst- och förlustsituationer.
- När äventyret är färdigspelat ges spelaren möjlighet att spela om textäventyret från början genom att mata in tecknet J eller j på tangentbordet. Om något annat skrivs in avslutas programmet.

```
-----  
Spelet är slut  
Tack!  
-----  
Tryck J för att spela igen:
```

Bild 7 Färdigt spel

2.5 Textfilernas utförande

För att programmet ska fungera korrekt behöver textfilerna som innehåller textäventyrspelen vara utformade på ett visst sätt. Programmet innehåller inga kontroller för att bestämma om textfilen är korrekt.

Nedan visas ett exempel, från *spel_default.txt*. Här visas de första raderna i textfilen.

Björnspelet (Default-äventyr)

!INITIAL TEXT

T1000|0000|Du är en stor <u><r>brunbjörn> i en djup mörk skog, vad gör du?

V1001|2100|Ät <c>blåbär!>

V1002|2200|Gå i ide...

V1003|2300|Leta efter <y>myror> att käka.

V1004|2400|<u>Ryt!>

Den första raden, här *Björnspelet (Default-äventyr)*, i textfilen ska vara namnet på spelet.

Den rad som börjar med *T1000* innehåller text som kommer visas i början av spelet. Bokstaven *T* innebär att det är en text som ska skrivas ut, och *1000* är dess identifikationsnummer. Just *T1000* är den textrad om alltid visas först när spelet startas, förutom spelets titel.

Under *T1000* listas de val som tillhör texten. Val ska börja med bokstaven *V* och sedan ha samma identifikationsnummer som texten de tillhör, fast ökat med 1 för varje val. Alltså innebär texten på rad *V1002* val nummer två för text *T1000*.

Efter de olika valens identifikationsnummer visas identifikationsnummer för den text som ska skrivas ut ifall det valet väljs av spelaren. Så om spelaren väljer val 3, *Leta efter myror att käka*, så kommer programmet leta upp den rad i texten som börjar med *T2300*, och lista de eventuella val som tillhör denna text.

Exempel från *spel_default.txt*, om användaren skulle välja valet 3 ovan.

!MYROR

T2300|0000|Myror är inget vidare, smakar <y>surt>.|Men fungerar i krig.|Var vill du leta efter myror?

V2301|3100|Leta under det stora stenblocket du ser framför dig.

V2302|3200|I myrstacken, duh!|Det borde ju finnas en <u>myriad> av myror där.

I exemplen visas *!INITIAL TEXT* och *!MYROR*, dessa är kommentarer och kommer inte användas av programmet. Programmet behandlar enbart rader som börjar med ett T eller ett V, och textfilens första rad med spelets titel.

Tecknet *|* i texterna skrivs ut som radbrytningar. Detta kan användas i både text och för val. Dock inte för spelets titel på den första raden. Flera (ex: *//*) tecken kan användas för att få flera radbrytningar efter varandra.

Den sifferkod som visas efter texternas identifikationsnummer används för att ge kommandon till programmet. För *0000* händer ingenting förutom att texten visas.

För sifferkoden *9999* som visas nedan i ett exempel från *spel_default.txt*, kommer äventyret att avslutas.

!GÅ I IDE – VINST

T2200|9999|Du går i ide: ZzzzzZzzzzz.|Bra björn, du vann!

Nedan listas de sifferkoder som är tillgängliga och vad de har för funktion i programmet:

- 0000 – Texten skrivs ut, ingen annan funktion.
- 9992 – Texten skrivs ut och byter färg till vit, standardfärg
- 9993 – Texten skrivs ut och byter färg till blå
- 9994 – Texten skrivs ut och byter färg till grön
- 9995 – Texten skrivs ut och byter färg till röd
- 9996 – Texten skrivs ut och byter färg till blå, spelet avslutas.
- 9997 – Texten skrivs ut och byter färg till grön, spelet avslutas.
- 9998 – Texten skrivs ut och byter färg till röd, spelet avslutas.
- 9999 – Texten skrivs ut och spelet avslutas.

Text för valen, som inleds med "V", behöver nödvändigtvis inte ligga direkt på raden under texten de tillhör. De behöver inte heller ligga i ordning efter varandra. Programmet letar igenom alla textfilens rader för att hitta korrekt rad att skriva ut, och spolar tillbaka textfilen efter varje upphittat val.

Exempel:

!Kommentar

V1001|1110|Olja in ditt träben.

V1003|1130|Träna prickskytte med din nystulna flintlåspistol!.

T1000|0000|Arrrghhh!|Du är en <r<råbarkad kapten> på ett piratfartyg, någon gång på 1700-talet. |Vad vill du göra?

V1002|1120|Drick rom!.

Kommer ge utskriften nedan

```
-----
Arrrghhh!
Du är en råbarkad kapten på ett piratfartyg, någon gång på 1700-talet.
Vad vill du göra?
-----
[1] - Olja in ditt träben.
[2] - Drick rom!.
[3] - Träna prickskytte med din nystulna flintlåspistol!.
-----
Ange val genom att slå in en siffra:
```

Bild 8 Utskrift av val som inte är ordnade efter identifikationsnummer i textfilen

Programmet har stöd för textformatering av enskilda tecken och textsträngar i spelet. Detta utförs med de taggar som har visats i exemplen ovan. Texten som ska formateras kapslas in på följande sätt:

Du är en <r<råbarkad kapten> på ett piratfartyg

I detta fall kommer texten *råbarkad kapten* att skrivas ut med röd text i programmet. Den text som ska ändras föregås av <x< och avslutas med >, där x är den typ av formatering som ska utföras

Nedan listas de textformateringar som är tillgängliga och vad de har för funktion i programmet:

- b – Blå text
- c – Cyan text
- g – Grön text
- l – lila text
- r – Röd text
- u – Understruken text, går att kombinera med en färg. Ex: `<u<<r<brunbjörn>`
- y – Gul text

2.6 Debug-läge

Programmet har stöd för ett enklare debug-läge. Läget aktiveras genom att initiera variabeln `_Bool debugMode = 1;`. För att avaktivera läget, sätts den till 0. Som standard är variabeln initierad till 0.

Variabeln är initierad i källkodens början, direkt efter funktionsdeklarationerna, utanför main-funktionen. Detta för att variabeln ska vara synlig för samtliga funktioner i filen.

Om debug-läget är aktiverat skrivs extra information ut till användaren, som kan användas för felsökning. Till exempel skriver vissa av funktionerna ut sina returvärden innan de ges. Debug-texter omges av hakparenteser.

```
[Funktion listaVal - valräknare värde: 4]
-----
Ange val genom at slå in en siffra: 1
val Input do-sats: 1, räknare: 4
[Funktion listaVal returnerar: 2100]
-----
Du hittar inga blåbär, vafalls?
Har någon annan björn ätit dem?
Vad gör du?
-----
[Switch-funktion: Nummer: 0]
[1] Leta upp blåbärtjuvsbjörnen!
[2] Meh, inte värt besväret. Kaka bark istället.
[Funktion listaVal - valräknare värde: 2]
-----
Ange val genom at slå in en siffra:
```

Bild 9 Extra debug-information från funktioner

2.7 Programmets kod

2.7.1 Macron

Macron är definierade i huvudfilen *spelmotortxt.c* och i *libtxt.h*

#define N 1000	Arraylängd för char-variabler, och för argument till vissa funktioner.
#define TEXTFIL "spel_default.txt"	Namn på den fördefinierade textfil som ska laddas, om inget argument ges till programmet vid start.
#define ADD_LINE "\n-----\n"	För utskrift av "linjer" med printf. (här nedkortad längd).
#define LT 11	Antal tecken i textfilen som ska hoppas över på början av raden för att skriva ut text.
#define TEXTHASTIGHET 15	Väntetid i millisekunder mellan att varje enskilt tecken skrivs ut. Endast för visuell effekt, sätt till 0 för inte använda.

Följande macron används för att formatera texten som skrivs ut, med ANSI-escape-kod.

ESC + [+ kod används för att mata in ANSI-escape-koder. I C är `\033` (oktalt) koden för tecknet escape.

För: ANSI-escape + n m

- där n är 31 - 37: Sätter olika färger på texten
- där n är 91 - 97: Sätter olika färger på texten, högintensivt
- där n är 40 - 47: Sätter olika färger på textens bakgrund
- där n är 0: Återställer formatering

m används för att indikera slut på formateringskoder

Exempel: `\033[91;43m TEXT` ger en intensiv röd teckenfärg med gul bakgrund

Macron för ljusare färg på text:

#define FORM_GREY "\033[90m"	Ljus svart (grå) text
#define FORM_RED "\033[91m"	Röd text.
#define FORM_GREEN "\033[92m"	Grön text.
#define FORM_YELLOW "\033[93m"	Gul text.
#define FORM_BLUE "\033[94m"	Blå text.
#define FORM_MAGENTA "\033[95m"	Lila text.
#define FORM_CYAN "\033[96m"	Cyan text.
#define FORM_WHITE "\033[97m"	Vit text.

Macron för mörkare färg på text:

<code>#define FORM_BLACK "\033[30m"</code>	Svart text.
<code>#define FORM_RED_DARK "\033[31m"</code>	Röd text.
<code>#define FORM_GREEN_DARK "\033[32m"</code>	Grön text.
<code>#define FORM_YELLOW_DARK "\033[33m"</code>	Gul text.
<code>#define FORM_BLUE_DARK "\033[34m"</code>	Blå text.
<code>#define FORM_MAGENTA_DARK "\033[35m"</code>	Lila text.
<code>#define FORM_CYAN_DARK "\033[36m"</code>	Cyan text.

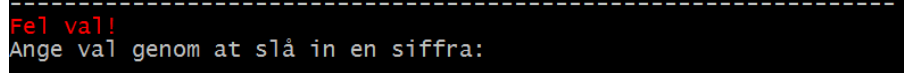
Macron för bakgrundsfärg på text:

<code>#define FORM_BLACK_BG "\033[40m"</code>	Svart bakgrund.
<code>#define FORM_RED_BG "\033[41m"</code>	Röd bakgrund.
<code>#define FORM_GREEN_BG "\033[42m"</code>	Grön bakgrund.
<code>#define FORM_YELLOW_BG "\033[43m"</code>	Gul bakgrund.
<code>#define FORM_BLUE_BG "\033[44m"</code>	Blå bakgrund.
<code>#define FORM_MAGENTA_BG "\033[45m"</code>	Lila bakgrund.
<code>#define FORM_CYAN_BG "\033[46m"</code>	Cyan bakgrund.
<code>#define FORM_WHITE_BG "\033[47m"</code>	Vit bakgrund.

Macron för annan textformatering:

<code>#define FORM_UNDER "\033[4m"</code>	Understruken text.
<code>#define FORM_INTENSIVE "\033[1m"</code>	Intensiv/fet-stilad text
<code>#define FORM_END "\033[0m"</code>	Återställer/avslutar textformatering

Dessa macron, samt `ADD_LINE` för utskrift av linje, kan användas direkt i *printf*-satser, och tillsammans med annan text. Exempel: `printf(ADD_LINE FORM_RED "Fel val!" FORM_END "\nAnge val genom at slå in en siffra: ");` ger utskriften i bilden nedan:



```
-----
Fel val!
Ange val genom at slå in en siffra:
```

Bild 10 Utskrift av *printf*-sats med macron och text

2.7.2 Variabler

FILE * textfil	Filvariabel för textäventyr i textform.
char s[N]	Håller rader från textfil. En rad i taget, eller N st. tecken.
char filnamn[N]	Inmatning av filnamn från användaren för att starta spel, om argument saknas vid start av program.
int idNum	För uppletning av textrad att skriva ut från textfilen. Sätts till 1000 för varje gång ett spel startas, för att skriva ut spelets första text som ska ha just identifikationsnummer 1000.
int idCheck	Läser in den aktuella textradens identifikationsnummer, för att jämföra mot värdet i variabeln idNum.
int switchCheck	Läser in den aktuella textradens sifferkod, ex. 9999 för att avsluta spelet.
_Bool debugMode = 0	Variabel för att sätta debug-läge på eller av. Om den sätts till 1 skrivs diverse extra information ut från funktioner.
_Bool korrektVal = 1	Sätts till 0 så fort texten för uppmanande av användarinmatning visas första gången för varje val-serie. Om användaren skriver in ett inkorrekt val, så kommer en annan uppmanande text att visas.
int valRaknare = 1	Räknar hur många val som hittas för varje nytt valscenario. Används för att testa om spelaren slår in en korrekt valsiffra.
int nastaldNum	Returneras från funktionen <i>listaVal</i> och sätts till variabeln <i>idNum</i> i main-funktionen, för att leta upp nästa textrad som ska skrivas ut.
char restart = 'j'	Variabel för omstart av spel, när spelet är avslutat ombeds användaren att mata in ett tecken. Om tecknet är "j" eller "J" startar spelet om från början.
int checkVal	Används för att hålla identifikationsnumret på en inläst rad från textfilen. Värdet i checkVal jämförs med det identifikationsnummer som ska letas upp, för att skriva nästa text som ska visas i spelet.
char valInput[2]	Håller inmatning från användaren när ett val ska utföras i spelet. valInput[0] kontrolleras för att bestämma om användaren har matat in ett korrekt val.

2.7.3 Inkluderade filer och bibliotek

Standardbibliotek

stdio.h

För diverse input/output-funktionalitet.

stdlib.h

För *system()*, som används för att köra systemkommandon. I detta program används följande systemkommandon:

- *cls* "Clear Screen" - blankar skärmen. Behövs också för fungerande textfärg.
- *pause* Pauserar programmet tills användaren trycker på en valfri tangent.
- *color* För färgbyte av kommandotolkens text och/eller bakgrund.
- *chcp* "Change Code Page", används för att byta teckenkodning.

string.h

För *strlen()*, som returnerar teckenlängden på en sträng.

ctype.h

För funktionen *tolower()* som används för att kontrollera om användaren har skrivit in j/J om denne vill spela om ett avslutat spel. Inmatningen konverteras till en gemen och jämförs med "j".

windows.h

För funktionen *Sleep()*. Funktionen används i programmet för att få en kort fördröjning mellan varje teckenutskrift i funktionen *skrivUtText*, som beskrivs i nästa kapitel.

Enligt Wikipedia innehåller *windows.h* egna inkluderingar av *string.h* och *ctype.h* och dessa hade eventuellt inte behövt inkluderats i källkodens text. Dock behålls inkluderingarna för garanterad funktionalitet.

Egna filer

libtxt.c

Innehåller funktioner för text och textfil-hantering. Funktionerna beskrivs i källkoden och i nästa kapitel i rapporten. Många av funktionerna är lika de som finns i boken, fast med annan namngivning.

Filen innehåller också andra funktioner, som inte används i projektet. Dessa funktioner beskrivs inte i rapporten.

libtxt.h

Innehåller deklarationer av funktioner i filen *libtxt.c*, samt inkluderingar av standardbibliotek.

Denna header-fil definierar också flertalet macron för textformatering.

Dessa har beskrivits i kapitel 2.7.1 *Macron*

2.7.4 Funktioner

Funktioner som ligger i huvud-filen spelmotortxt.c

Main-funktionen och funktionen ListaVal beskrivs utförligare i senare kapitel.

int listaVal(int a, FILE *f);

Funktionen skriver ut de val som finns för det aktuella scenariot i spelet. Som parametrar får funktionen textens identifikationsnummer som en int-variabel, samt den inlästa textfilen.

Funktionen kontrollerar att användaren matar in en korrekt siffra för val, och returnerar sedan det identifikationsnummer som tillhör nästa text som ska skrivas ut.

void skrivUtText(char *string, int n, _Bool linjer, _Bool tabb);

Funktionen skriver ut text till skärmen, ett tecken i taget. Mellan varje teckenutskrift sker en fördröjning på lika många millisekunder som macro *TEXTHASTIGHET* är definierat till. Som argument får funktionen textsträngen som ska skrivas ut, en int-variabel med antal tecken som ska skrivas ut. *_Bool*-variablerna bestämmer om funktionen ska skriva ut "rader" i början och slutet av textblocken, och om texten ska innehålla tabbslag efter radbrytning.

_Bool textSwitchar(int s);

Funktionen används för att köra vissa kommandon som kan användas i spelet. Som t.ex. att avsluta spelet eller att byta färg på texten. Som parametrar får funktionen den sifferkod som finns efter den aktuella textradens identifikationsnummer. Funktionen returnerar en etta eller nolla beroende på om spelet ska avslutas eller inte.

Funktioner som finns i filen libtxt.c:

Utförligare kommentering finns i källkodsfilen

_Bool removeNewLine(char[]);

Funktionen kontrollerar om det sista tecknet i en textsträng är ett nyradstecken, och raderar den om så är fallet. Denna funktion är identisk med bokens funktion *remove_nl* på sida 199.

void clearBuffer(void);

Tömmer textbufferten, denna funktion är identisk med bokens funktion *skip_line* på sida 190.

void clearBufferFil(FILE *f);

Tömmer textfilinläsningsbufferten, denna funktion är identisk med bokens funktion *fskip_line* på sida 226.

_Bool radInput(char[], int);

Skriver textinmatning från användare till char-variabel, använder sig av funktionerna *clearBuffer* och *removeNewLine*. Funktionen är identisk med bokens funktion *read_line* på sida 200.

_Bool textfilTillString(char[], int, FILE *f);

Funktion som läser in rader från textfil till char-variabel. Funktionen returnerar 0 när inläsning misslyckas eller är klar. Vid korrekt inläsning returneras 1. Funktionen är identisk med bokens funktion *fread_line* på sida 226

_Bool TTS(char *a, int n, FILE *f);

Kan användas i stället för funktionen *textfilTillString*. TTS anropar funktionen *textfilTillString* och returnerar dess värde.

2.7.4 Main-funktionen

I detta kapitel beskrivs grundläggande vad som händer i programmets kod, för ytterligare kommentering hänvisas läsaren till källkodsfilerna.

- Initialt deklareraras och initieras variabler och funktioner som finns i main-filen. Med funktionen `system()` sätts kommandotolkens teckenkodning till 1252 och skärmens innehåll blankas. Sedan skrivs programmets namn ut med den egna funktionen `skrivUtText()`. Med bool-argumentet 1 för funktionen skrivs "linjer" ut före och efter texten.
- Pekarvariabeln `textfil` sätts till det argument som givits till programmet vid start, med funktionen `fopen()` och argument "r" som innebär endast läsning av textfil. `textfil` kommer ge returvärdet **NULL** om textfilen inte kan öppnas. Detta nyttjas i tre **if**-satser, som triggar om en textfil inte kunnat öppnas. Om `textfil` är **NULL** kommer programmet först att be användaren om att skriva in namnet på den textfil som denne vill använda. Inmatning från användaren görs med den egna funktionen `radInput()` och sätts till variabeln `filnamn`.

Om `textfil` inte kan öppna den användarinmatade filen i `filnamn`, så triggar nästa **if**-villkor där programmets standardfil för spel `spel_default.txt`, läses in. Om denna fil inte kan läsas avslutas programmet.

- En **while**-loop kapslar in resterande programkod i main-funktionen, villkoret för att loopen ska köra är att det gemena tecknet i variabeln `restart` är lika med "j". Variabeln initieras till "j" innan **while**-loopen, för att den ska köra första gången. För att testa det gemena tecknet används funktionen `tolower()`. Varje gång ett spel är färdigt uppmanas användaren att välja om denne vill spela om det aktuella spelet från början. Om användaren skriver in något annat än J eller j kommer programmet att avslutas.
- Variabeln `idNum` sätts till **1000**, `idNum` innehåller identifikationsnumret till den textrad som ska skrivas ut på skärmen. Just **1000** är spelets första text att skriva ut.
- Den egna funktionen `TTS()` anropas för att läsa in textfilens första rad (eller `N` st. tecken, det som sker först) till variabeln `s`. `N` är ett macro definierat till **1000**. Om raden i textfilen innehåller fler än 1000 tecken blir inte inläsningen korrekt. Sedan skrivs spelets titel ut med `printf()`, som ska vara textfilens första rad. Programmet pauserar med `system("pause")`, tills det att användaren trycker på en valfri tangent.
- En **while**-loop körs med villkoret "`TTS(s, N, textfil)`". `TTS()` ger ett sanningsvärde i retur så länge inläsning sker korrekt. När textfilen är slut kommer sanningsvärdet att bli 0. Detta innebär att för varje varv i **while**-loopen så kommer en textrad från textfilen att behandlas, så länge den inte har fler än `N` st. tecken.
- Identifikationsnumret på textraden i variabeln `s` läses in till variabeln `idCheck` med standardfunktionen `sscanf()`. `sscanf()` fungerar likt `scanf()`, men får input från en teckenström (`s`) istället för från tangentbordet. Då identifikationsnumret börjar på radens andra tecken, används `s+1` som argument. `s` pekar på strängens första värde, `s+1` på dess andra osv.
- En **if**-sats triggar när rätt rad att skriva ut hittas i textfilen. Villkoret är att värdet i `idCheck`, som sätts till ett nytt värde för varje rad som läses från textfilen, är lika med värdet i `idNum`. Samt att radens första tecken är "T". När denna rad hittas skrivs texten ut med funktionen `skrivUtText()`.

- Den funna radens sifferkod läses in till variabeln *switchCheck*. Funktionen *textSwitcher* anropas med värdet i *switchCheck* som argument. I funktionen används en **switch**-sats för att utföra olika moment beroende på vad värdet i *switchCheck* är. Om spelet ska avslutas ger funktionen sanningsvärdet **0**, annars **1**. Då funktionen är relativt enkel kommer den inte beskrivas utförligare i denna rapport.

Om värdet är **0** triggas en **if**-sats i main-funktionen, som skriver ut att spelet är avslutat, med funktionen *skrivUtText()*. Sedan används **break** för att bryta den inre **while**-loopen.

När spelet är avslutat anropas *rewind()* för att "spola tillbaka" textfilen. Därefter får användaren möjlighet att spela om det aktuella spelet genom att skriva in ett tecken, som sätts till variabeln *restart* med funktionen *getchar()*.

- Om spelet inte ska avslutas anropas funktionen *listaVal()* med värdet i *idNum* som argument, samt textfil-variabeln *textfil*. Funktionen *listaVal()* beskrivs i nästa kapitel. Funktionen listar de val användaren kan göra och ger i returvärde ett identifikationsnummer som skrivs till *idNum*, beroende på vilket val användaren gjorde. Textfilen spolas tillbaka med funktionen *rewind()*, och den inre **while**-loopen börjar om, för att leta upp nästa rad att skriva ut.

2.7.5 Funktionen listaVal()

Funktionen listaVal() skriver ut de val som finns för det aktuella scenariot i spelet och får en användarinmatning. Sedan returnerar funktionen det identifikationsnummer som tillhör nästa text i spelet.

- Initialt deklarerar variabler som används i funktionen. *valRaknare* används för att kontrollera och räkna antal val som tillhör den senast visade textraden i spelet, variabeln initieras till 1, för att kunna hitta textens första val. Textfilen spolas tillbaka med *rewind()*, för att valsökning ska börja på första raden.
- En **while**-loop körs med villkoret "*TTS(s, N, f)*". *f* är filvariabeln som skickats som argument till funktionen. *TTS()* ger ett sanningsvärde i retur så länge inläsning sker korrekt. När textfilen är slut kommer sanningsvärdet att bli 0. Detta innebär att för varje varv i **while**-loopen kommer en textrad i textfilen att behandlas, så länge den inte har fler än *N* st. tecken.

För varje varv i loopens läses radens identifikationsnummer till variabeln *checkVal*. En if-sats triggar om *checkVal* är lika med *a+valRaknare*, samt om radens första tecken är ett *V*. *a* är den senaste visade textens identifikationsnummer som har skickats som argument till funktionen. Om if-satsen triggar innebär det att ett val som tillhör texten har visats. Det aktuella valets text skrivs ut med den egna funktionen *skrivUtText()*, med siffran i valräknare som prefix, *valRaknare* ökar sedan med 1, för att kontrollera om fler val finns. I varje varv spolas också textfilen tillbaka med *rewind()*, detta görs för att kunna läsa in val som inte ligger sorterade efter identifikationsnummer i filen.

- När **while**-loopen har kört färdigt har alla valen hittats och *valRaknare* minskar med 1, för att detta val inte hittades. En **do/while**-loop startar och användaren uppmanas att slå in en siffra, för att välja hur denne vill fortsätta i spelet. För att undvika programfel, i fall användaren skriver in något annat än en siffra, används först den egna funktionen *radInput()* för att skriva inmatningen till char-variabeln *valInput*. Det första tecknet i variabeln testas med *isdigit()*, *isdigit* ger returvärdet 1 om det testade tecknet är en siffra. Om så är fallet läses siffran in till variabeln *anvandareVal* med *sscanf()*. Annars sätts *anvandareVal* till -1.

Om användaren skriver in en korrekt siffra uppfylls inte villkoret i **do/while**-satsen och programmet går vidare. Annars, om användaren har gjort en felaktig inmatning uppmanas användaren att återigen slå in en siffra.

- Textfilen spolas tillbaka med *rewind()*. En **while**-loop körs med villkoret "*TTS(s, N, f)*" tills raden med det val användaren har gjort hittas. Denna rad innehåller det identifikationsnummer som ska visas i spelets nästa steg. Numret läses in till *nastaldNum* med *sscanf(s+6, "%d", &nastaldNum)*. *s+6* innebär att inläsning ska ske från det sjunde tecknet i variabeln *s*.
Exempel på rad: *V1003|1130|Träna pricksskytte med din nystulna flintlåspistol!*
I exemplet kommer den blåmarkerade texten att läsas in till *nastaldNum*, detta värde ges i retur och funktionen avslutas.

3 Diskussion

I detta kapitel diskuteras genomförandet och resultatet av projektet, samt ges förslag på ändringar och andra utföranden.

Användandet av `rewind()`

I programmet utförs "tillbakaspolning" av textfilen flertalet gånger, av olika anledningar. Detta kan upplevas som en dålig lösning för att söka upp spelets textrader som ska visas. En alternativ och eventuellt bättre lösning skulle vara att läsa in textfilen en gång och räkna antalet rader. Sedan skapa en struktur (struct)-array med antalet hittade rader som arraystorlek, och sedan läsa in hela textfilen till strukturerna. Ett exempel på hur en sådan struktur skulle kunna se ut visas nedan:

```
struct aventyr{  
    int idNum, sifferKod, nastaldNum;  
    char text[N];  
    _Bool AvslutaSpel, typ;  
};
```

Arrayen hade sedan kunnat sorteras efter identifikationsnummer och genomsökning skulle eventuellt bli snabbare.

Spel hade då också gått att spara genom att skriva ut hela strukturen till en binär datafil (.dat), dock skulle detta försvåra uppdatering och ändring av spelet.

Denna version av programmet har påbörjats och återfinns på programmets GitHub-sida, under branch-namnet `struct-load`.

Textfilerna

Programmet utför ingen kontroll för om textfilen som laddas är ett äventyrsspel i rätt format. Textfilerna måste också vara skrivna i ett mycket specifikt format för att fungera. En lösning på båda problemen är att skriva ett nytt program för skapande av speläventyr, där programmet håller reda på val och identifikationsnummer.

Spelen skulle kunna sparas och läsas in till det programmet i binärt datafilsformat (.dat) och hanteras med strukturer.

En speciell textsträng skulle kunna läggas till i filen för att visa att det är ett spel som kan spelas i SPELMOTOR TXT.

Kompabilitet

Koden går inte att kompilera i en icke-Windowsmiljö på grund av inkluderingen av biblioteket <windows.h>. Dock, med ett fåtal ändringar i koden finns möjlighet att köra programmet i t.ex. en Linux-miljö, se bild nedan.

En framtida uppdatering skulle eventuellt vara att skapa kompabilitet för flera plattformar.

```
SPELMOTOR TXT
-----
Ange filnamn (med filändelse) för textfil.
Mata in EOF (Ctrl+Z) för att avbryta.
Om ingen fil anges laddas spel_default.txt

Textfil:
Inmatad fil: '' kunde inte lösas.
Läser in standard-spel spel_default.txt
-----
Spel som kommer köras är: Björnspelet (Default-öventyr)
-----
sh: 1: pause: not found
sh: 1: cls: not found

-----
Du är en stor brunbjörn i en djup mörk skog, vad gör du?
-----
[1] -  Öt blöbör!
[2] -  Gå i ide...
[3] -  Leta efter myror att köka.
[4] -  Ryt!
-----
Ange val genom att slå in en siffra: █
```

Bild 11 Programmet i Debian, kompilerat med gcc. Dock ej med rätt teckenkodning.

4 Bilagor

Ändringslogg

Återfinns också i källkoden

2016-10-10

- Initial release
- Testspel fungerar att köras från textfil.
- Spelet kan avslutas
- Kontroll tillagt för användarinmatning vid val
- Debug-funktionalitet tillagd

2016-10-11

- Radbrytningar i text, dessa triggas med tecknet | i textfilen
- Funktion för switchar
- Test för om textfilen existerar
- Tagit bort onödig variabel "bokstav", behövs ej för att fånga upp första bokstaven i textraderna
- Det går att starta spelet med en textfil som argument, då laddas denna fil ex: 'text_dyn.exe spel_pirat.txt'
- Namngivning av program: SPELMOTOR TXT
- Borttagning av onödiga deklarerade variabler som inte användes.

2016-10-12

- Stöd för titel av spel, som ska/måste skrivas på rad 1 i textfilen.
- Stöd för omspel, användaren kan trycka på J/j efter avslutat spel för att spela om.
- Fler sifferkoder tillagda för färgbyte vid avslut av spel.

2016-10-13

- Ändrat ordning på laddning av textfiler
- Lagt in system("chcp 1252") för att byta teckenkodning, så att svenska tecken visas korrekt.
- Uppdaterat debug-texter
- Korrigerat grammatik och stavfel i text.
- Lagt till tre nya kommandon för att byta textfärg, utan att avsluta spelet
- Flertalet namnbyten av variabler

2016-10-16

- Ökade radinläsning (macro N) till 1000st tecken.
- Bytt parameter -> argument i kommentarer
- Ny _Bool-parameter för funktion skrivUtText, används för tabbslag i början av text, och efter radbrytning
- Ändrade hur utskrift av val ser ut, för läsbarheten skull

2016-10-19

- Fixade kontroll för inmatning av användarval, programmet slutar nu inte att fungera om användaren matar in något annat än en siffra
- Lade till funktionalitet för att läsa in val från textfil som inte ligger i ordning, eller efter texten de tillhör
- Lade till sifferkod 9992, för färgbyte av text till vit
- Lagt till stöd för enskild teckenfärg och formatering

2016-10-21

- Ändrat vissa teckenkoder för textformatering, så färgen blir ljusare vid utskrift
- Namnbyte av fil strings_text_v1.c till funk.c
- Namnbyte av fil strings_text_v1.h till funk.h

2016-10-22

- Namnbyte av funk.c till libtxt.c
- Namnbyte av funk.h till libtxt.h
- Flyttade macron för textformatering till libtxt.h
- Flyttade macro för "linje" till libtxt.h
- Ny namngivning av macron i libtxt.h
- Ändring av macro-anrop i koden

2016-10-23

- Ändrade felaktig text i kommentarer

2016-10-27

- Ytterligare kommentering av källkoden