Mjolner Workshop[1]

Denne note indeholder 5 øvelser, der dækker de vigtigste egenskaber ved mjolner-værktøjet. Derudover er der en oversigt over nogle meget brugte kommandoer – tag evt. et kig på dem nu.

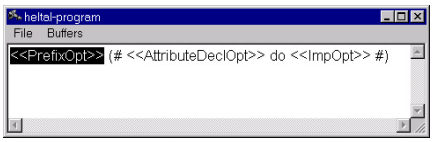
Hent filerne 'syntakseditering.bet', 'useloops.bet' og 'loops.bet' over til dig selv.

Start mjolner-værktøjet med "mjolner" (ikke "mjolner &").

Øvelse 1: Basal brug – editering og kompilering

I denne øvelse skal vi lave et program, der indlæser en række heltal og udskriver min og max (opgaven fra ugeseddel 1).

Vælg File->New BETA Program... Kald det nye program 'heltal'.

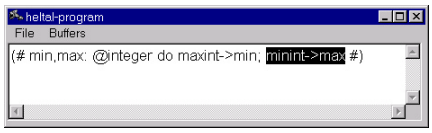


Når der står `<<AttributeDeclOpt>>` betyder det, at hér kan man skrive attributter. Vores program skal bruge to integers: `min` og `max`. Tryk på `<<AttributeDeclOpt>>` og skriv "`min,max: @integer`". Vælg Edit->Parse Text[2] for at afslutte tekst-editering.

Ved `<<PrefixOpt>>` kan man vælge at skrive et superpattern. Det ønsker vi ikke – tryk på `<<PrefixOpt>>` og Delete.

Ved `<<ImpOpt>>` kan der stå imperativer. Vi skal have initialiseret `min` og `max`. Tryk på `<<ImpOpt>>` og skriv "`maxint->min ; minint->max`". Gå ud af tekst-editering.

Nu skal vi have vores løkke. Markér `minint->max` ved at trykke museknappen ned midt i `minint` og slippe igen midt i `max`.

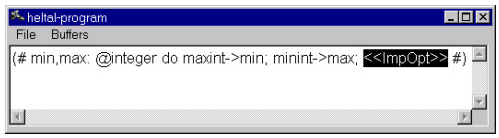


Indsæt noget "tom kode" efter det markerede ved at vælge Edit->Insert After (eller tryk ctrl-a eller return).

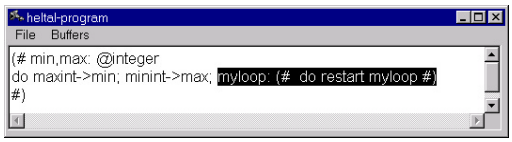
---

[1] Denne note er oprindelig forfattet er Christian Heide Damm (damm@daimi.au.dk) i oktober 1999, men er opdateret i september 2000 (af Elmer Sandvad) i forhold til release 5.2.

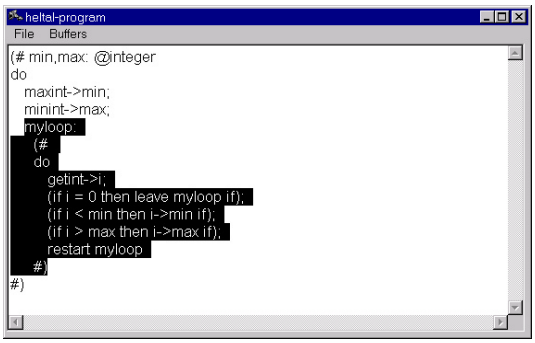
[2] For at starte/slutte tekst-editering kan man også bruge ctrl-t eller ctrl-space.



Indsæt løkken ved at skrive "`myloop: (# do restart myloop #)`". Gå ud af tekst-editering.



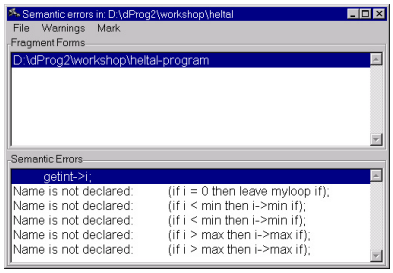
Gå i tekst-editering igen, mens hele `myloop`-løkken er markeret som på billedet (Markér løkken ved at trykke museknappen ned midt i `myloop` og slippe den igen inden for `(#...#)`. Gå i tekst-editering ligesom du tidligere har gået *ud* af tekst-editering). Udfyld løkken med kode, så den ser ud som på billedet nedenfor.



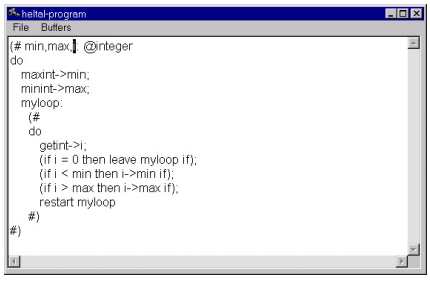
Virker koden?

Find ud af det ved at checke den: vælg Compile/Run->Check 'heltal' eller tryk på Check-knappen  i værktøjslinien.

Hvis der er fejl i programmet, kommer der et vindue frem, der viser fejlene:

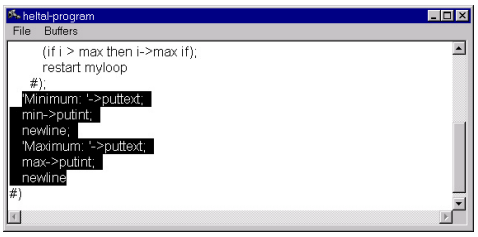


Du kan klikke i den nederste liste for at gennemse fejlene. Nedenunder listen og nederst i det store vindue står, hvad der er galt: "Info: Name is not declared". Det er selvfølgelig `i`, der ikke er erklæret. Indsæt `i` ved at trykke på `max` og indsætte "tom kode" derefter (det har du prøvet før). Skriv `i` og gå ud af tekst-editering.



Check programmet igen. Nu burde det virke.

Der skal indsættes et par udskrifter til sidst i programmet. Markér hele `myloop`-løkken (tryk museknappen ned midt i `myloop` og slip den igen inden for `(#...#)`). Indsæt "tom kode" efter løkken og tilføj udskrifterne af `min` og `max` som på billedet nedenfor.



Kompilér og kør programmet fra mjolner (du må *ikke* skrive "beta heltal" nogen steder!). Man kan interagere med programmet i den shell, hvor mjolner blev startet.

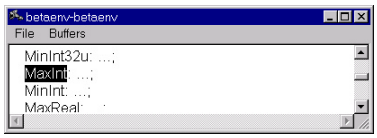
Øvelse 2: Browsing

I denne øvelse skal vi bruge mjolner-værktøjet til at browse rundt i koden.

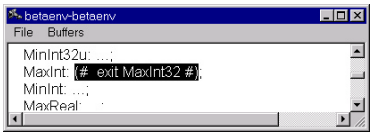
Vi skal bruge programmet fra øvelse 1.

Hvad er maxint og minint?

I 'heltal' bruger vi `maxint` og `minint`, men hvad er de defineret til? Find ud af det ved at markere f.eks. `maxint og` trykke på Definition-knappen i værktøjslinien (eller dobbelt-klik på `maxint eller` markér `maxint` og tryk ctrl-d).



Den er altså defineret i filen 'betaenv'. Dobbelt-klik på de tre prikker ud for `maxint`.

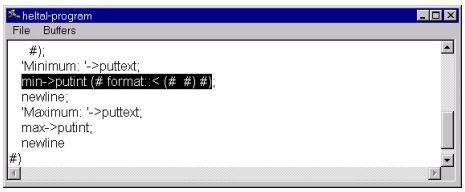


Fortsæt på samme måde med `maxint32`. Hvad er værdien?

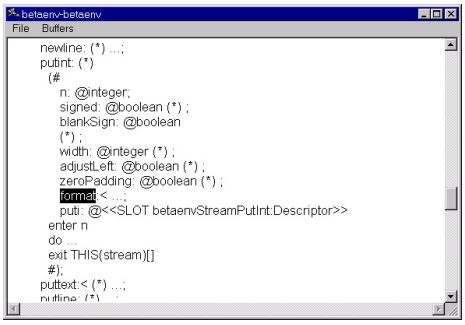
Nu skal vi tilbage til 'heltal' igen. Tryk på Back-knappen i værktøjslinien (eller på højre muse-knap og vælg Back eller tryk ctrl-b). Gentag, indtil du er i 'heltal' igen.

Minimum skal udskrives højrestillet med foranstillede nuller

Markér `min->putint` og gå i tekst-editering. Tilføj `"(# format::< (##) #)"` til putint, som på billedet:



Dobbelt-klik på `format`. Da compileren ikke har checket den tekst, som du lige har skrevet, ved den ikke hvad det betyder: svar ja til at den skal checke programmet.



Nu kan du se egenskaberne for `putint`, og du kan sikkert regne ud, hvad de forskellige ting betyder. Viderebind `putint.format` i 'heltal', således udskriver `min` højrestillet, med bredde 10 (width) og med foranstillede nuller (zeropadding).

Hvilke slags containere findes?

Du har hørt om lister og arraycontainere, men findes der andre slags containere?

I øverste linie af Projects-vinduet i mjolner-værktøjet står der "Std. Libraries/". Dobbelt-klik på denne linie eller klik på + for at åbne den. Åbn derefter "containers".

Herved vises nu de fragmentgrupper/filer , der findes i ~beta/containers/...

Prøv at gå filerne igennem og se de forskellige containere – der er 17 i alt!

Hvis der står **(*)** i et program, er der indsat en kommentar på det pågældende sted. Ved at dobbeltklikke på stjernen, kan man få kommentaren at se. Dobbelt-klik igen for at skjule kommentaren[3].

Hvis der er for meget på skærmen, til at man kan overskue det, kan man trykke ctrl-o (overview). Derved abstraheres en masse detaljer væk. Dobbelt-klik for at få flere detaljer (eller markér og tryk ctrl-d, eller på Definition-knappen i værktøjslinien).

Hvad findes der ellers i beta-systemet?

Nedenstående filer er gode at kende – tag et hurtigt kig på dem.

- basiclib
  - /directory (interface til directories på disk)
  - /file (filbehandling)
  - /math (matematiske beregninger)
  - /random (tilfældige tal)
- guienv
  - /controls (diverse brugergrænsefladeelementer, såsom knapper, tekst-felter, check-boxes)
  - /stddialogs (diverse standarddialoger, såsom fildialoger)
- persistentstore (facilitet til at gemme objekter på disk)
- sysutils

[3] Visning og skjulning af en kommentar kan også udføres med Definition-knappen  i værktøjslinien eller vha. ctrl-d.

- /envstring (få adgang til environmentvariable, f.eks. $USER)
- /time (få bl.a. fat i tiden lige nu)

Øvelse 3: Editering vha. syntaktiske kodestumper

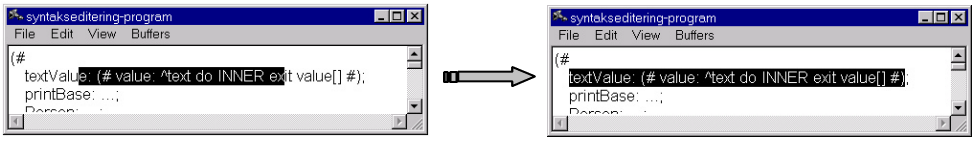
En meget vigtig egenskab ved værktøjet er, at man kan flytte rundt på og kopiere store kodestumper i en håndevending – uden at der nogensinde opstår syntaktiske fejl, såsom en manglende parentes e.l.

Det er desværre svært at illustrere i en lille opgave, hvor nyttigt ovenstående i virkeligheden er. Øvelsen vil derfor bestå i en gennemgang af de teknikker, der skal til for at kunne forstå og bruge syntaks-editering.

Hent filen 'syntakseditering' ind i værktøjet.

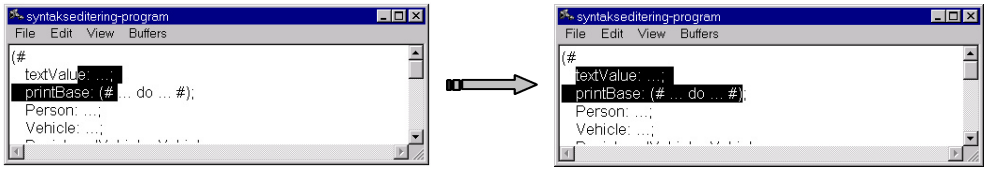
Markering af kodestumper

For det første skal man vide, hvordan man markerer en stump kode. Det er *ikke* muligt at markere noget, som ikke er en "helhed", såsom halvdelen af en descriptor



Værktøjet vil altid udvide det markerede, indtil det det passer med en syntaktisk kategori. I ovenstående eksempel vil selektionen blive hele det omsluttende pattern.

Endnu et eksempel. Du markerer fra midt i et pattern til midt i et andet pattern.

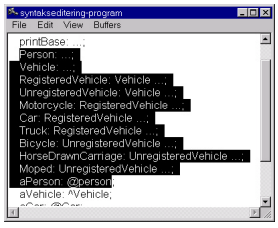


Værktøjet vil selektere begge patterns.

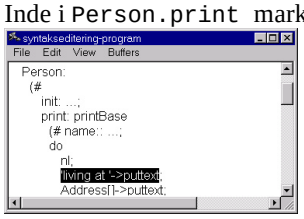
Prøv at markere forskellige ting i værktøjet og observér, hvad der bliver selekteret. Prøv f.eks. at markere følgende:

- `UnregisteredVehicle: Vehicle ...;`

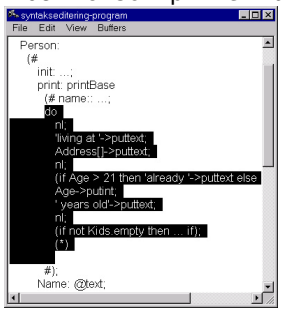
- `Person: ...` indtil `aPerson: @person`



- Inde i `Person.print` markér `'living at'->puttext`



- Inde i `Person.print` markér hele do-delen



Eksperimentér også med brugen af ctrl-o (overview) og ctrl-d (detail).

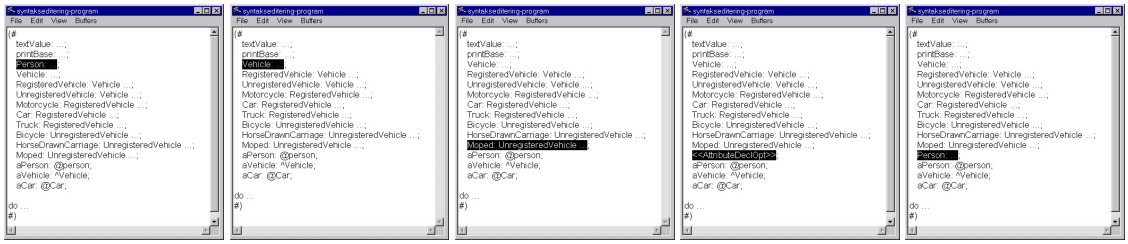
Cut, copy, paste af kodestumper

Værktøjet understøtter de sædvanlige operationer cut, copy, paste, undo og redo med de sædvanlige tastegenveje:

Operation	Tastegenvej
cut	ctrl-x
copy	ctrl-c
paste	ctrl-v
undo	ctrl-z
redo	ctrl-y

Nu skal vi flytte `Person`-patternet ned til `aPerson: @person`.

1. Markér `Person`-patternet.
2. Tryk ctrl-x for cut.
3. Markér `Moped: UnregisteredVehicle`.
4. Tryk ctrl-a for indsæt kode herunder.
5. Tryk ctrl-v for paste.



Hvor stor er `Person`-patternet egentlig? Det kunne faktisk have været flere tusinde linier stort, og det ville stadig være meget enkelt at flytte rundt på det eller kopiere det.

Prøv at undo'e det hele med ctrl-z.

Man kan gøre det samme med alle mulige andre kodestumper: imperativer, descriptorer, do-dele, enter-dele, exit-dele, variabelnavne, superpattern-prefix'er. Men det kræver en lille smule øvelse at bruge det effektivt.

Øvelse 4: Fragmentsystemet

Denne øvelse handler om fragmentsystemet og understøttelsen af dette i mjolner-værktøjet.

At skjule implementationsdetaljer

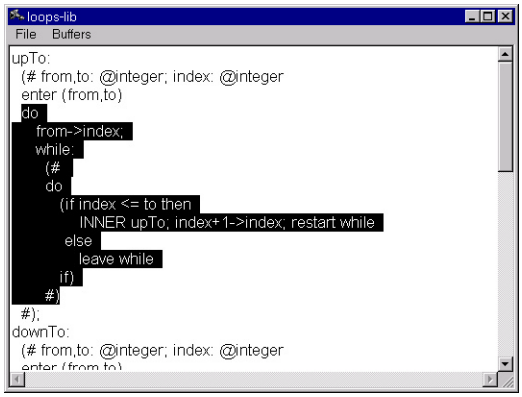
Hent programmet 'useloops' ind i mjolner-værktøjet.

Ved at klikke på + ved 'useloops'-ikonet i Projects-vinduet kan du se at programmet inkluderer (INCLUDE) filen 'loops' (angivet ved en enkelt-op-pil). Klik på linien med 'loops'.

Vi vil gerne gemme do-delene i en anden fil, vores implementationsfil. Mens du er i 'loops', vælg SLOTs->Create Implementation File... og skriv "loopsbody". Herved skabes en ny fil 'loopsbody', som har ORIGIN i 'loops' (angivet ved en dobbelt-op-pil)., samtidig med at 'loops' har BODY til 'loopsbody', vises ved: 'loopsbody'.

Gå tilbage til 'loops' ved at dobbelt-klikke på 'loops' i Projects-vinduet.

Markér do-delen i `upTo` (tryk museknappen ned midt i `do` og slip den et sted inde i selve do-delen).



Vælg SLOTs->Make DoPart SLOT, og giv det navnet "upToImplementation". Nu flyttes implementation af `upTo` automatisk ned i 'loopsbody'. Dobbelt-klik på `<<SLOT upToImplementation: DoPart>>`, hvorved du kommer direkte hen til koden i 'loopsbody' (eller tryk på Definition-knappen i værktøjslinien, eller brug ctrl-d).

Mjolner-værktøjet opererer hele tiden med en "current implementation file", og det er denne fil, implementation lægges ned i, når man vælger SLOTs->Make DoPart SLOT.[4]

Gentag sekvensen med `downTo` og `stepTo`[5].

---

[4] Når man bruger SLOTs->Create Implementation File... og derved skaber en ny fil, sættes current implementation file automatisk til den nye fil. Fra SLOTs-menuen kan man også vælge en anden current implementation file.

[5] Dette kan også udføres på én gang ved at markere både downTo og stepTo og benytte SLOTs->Hide implementation…

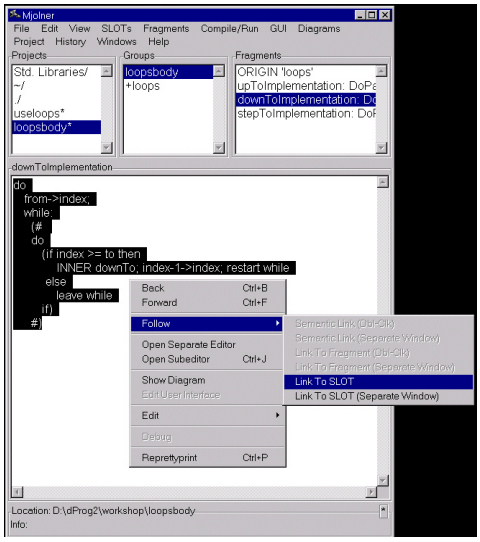
Browsing op og ned gennem SLOTs og fragmentformer

Et SLOT kan kendes på `<<SLOT blabla: DoPart>>` eller lignende. En fragmentform kan kendes på `---blabla: DoPart---`. Ved hjælp af værktøjet kan man gå fra et SLOT til den tilhørende fragmentform eller omvendt.

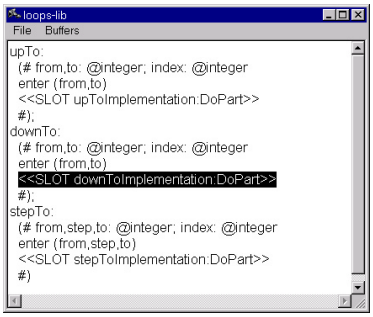
Gå til 'loops' og dobbelt-klik på `stepTo`'s SLOT, hvorved `stepTo`'s implementation vises.

I 'Fragment Forms'-vinduet kan man nu se, at der er tre do-dele i filen, og at stepToImplementation bliver vist i øjeblikket. Gå til `downTo`-implementationen ved at trykke på linien ovenover. Hvor er det tilhørende SLOT til `downTo`-koden?

Tryk på højre museknap i kodevinduet og vælg Follow->Link To SLOT eller tryk på 'To Slot'-knappen  i værktøjslinien[6].



Værktøjet leder så ORIGIN-kæden igennem (som reglerne nu siger man skal) og leder efter et SLOT med det rigtige navn – og det finder den i 'loops'.



I 'loops' er der defineret tre patterns: `upTo`, `downTo` og `stepTo`. De er mere præcist defineret i fragmentformen, der hedder `lib: Attributes`, som det kan ses i øverste, højre hjørne. Men hvilket SLOT bliver de tre patterns så lagt ind i, dvs. hvor er `lib: Attributes` defineret? Find ud af det på samme måde som ovenfor. Det rigtige SLOT ligger selvfølgelig i 'betaenv'.

[6] Dette screen dump er fra r5.1 versionen af Mjølner-tool.

Øvelse 5: Debugging

I denne øvelse ser vi kort på, hvordan mjolner kan hjælpe dig med at finde fejl i dine programmer.

Indlæs 'syntaksediting'.

Kompilér  og kør  programmet. Der er en "Reference is none"-fejl.

Højreklik i koden og vælg Compile/Run->Debug 'syntaksediting' eller tryk på Debug-knappen  i værktøjslinien. Du får nu et nyt vindue, hvor debug-informationen findes.

Reference is none

Vælg "Go" fra det nye vindue. Når programmet stopper denne gang, fortæller debug-vinduet nederst, at der er en "Reference is none"-fejl. Og kode-vinduet har markeret den kommando, der førte til fejlen (dvs. `'kg'->value`, der burde være `'kg'->value[]`). Ret fejlen og kør igen. For at rette fejlen skal man stoppe debuggeren. Det kan gøres ved at lukke debug-vinduet eller ved at højre-klikke i editor-vinduet og vælge Stop debugging.

Så let findes en "Reference is none"-fejl!

Breakpoints

Vi vil for øvelsens skyld indsætte et breakpoint ved `'The car is:'->putline`. Markér denne linie og højreklik. Vælg "Set Break". Markér noget andet (lige meget hvad).

Vælg "Go". Nu kører programmet lidt og stopper ved breakpointet.

Gå til debug-vinduet. Tryk på Object-knappen. Herved vises det aktive objekt, dvs. det objekt, hvis do-del er ved at blive udført. Det aktive objekt er naturligvis hele program-objektet.

Man kan se, at objektet har en `aPerson`-komponent, som er en instans af `program.Person`-patternet. Det vidste vi jo godt i forvejen. Dobbeltklik på denne linie. Dobbeltklik på `Name`, og dobbeltklik på `T`. Personen hedder "Santa Claus".

Sådan kan objekt-strukturen inspiceres på runtime. Det er nogle gange bedre end at indsætte debug-udskrifter.

Klik på breakpointet `>>1>>` og højreklik. Vælg "Erase Break". Vælg "Rerun" fra enten højrekliksmenuen eller det nye vindue. Så er programmet klart til at starte forfra.

Vigtige kommandoer og tastegenveje

Følgende operationer virker ikke i tekst-mode.

Operation	Tast	Mus/ Knap	Effekt	Menuindgang
Reprettyprint	ctrl-p		Gentegner koden, hvis der er gået ged i den...	View -> Reprettyprint
Overview	ctrl-o		Abstraherer koden omkring det markerede	View -> Overview
Detail **Vis definition** **Vis kode**	ctrl-d	Dbl-click  	Denne operation har en del forskellige betydninger afhængig af den markerede kode. Viser lidt flere detaljer om den markerede kode, finder definitionen af et navn, åbner eller lukker en kommentar.	View -> Detail View -> Follow Semantic Link View->Follow Link to SLOT
Abstract	ctrl-k		Abstraherer det markerede	View -> Abstract
To Slot	F3	 	Leder ad ORIGIN-kæden efter et SLOT med samme navd som den fragment form man står i	View-> Follow Link to SLOT
Back	ctrl-b	 	Finder tilbage til der hvor man kom fra ved f.eks. Vis definition.	History->Back
Forward	ctrl-f	 	Det modsatte af Back.	History ->Forward

Tom kode herover	ctrl-a return		Indsætter "tom kode" lige efter det markerede (hvadenten det er en erklæring, et funktionskald eller lignende).	Edit -> Insert After
Tom kode herunder	ctrl-u		Som ovenfor, blot indsættes den "tomme kode" *før* det markerede.	Edit -> Insert Before

Check current	F4		Checker koden (ligesom ved kompilering – den nøjes bare med at se, om programmet er lovligt).	Compile/Run -> Check Current
Check program	ctrl-F4	 	Som ovenfor: den genchecker det program, der blev checket sidst.	Compile/Run -> Check Program
Kompilér Current	F5		Kompilerer koden og genererer eksekvérbar fil (hvis current er et program).	Compile/Run -> Compile Current
Kompilér Program	Ctrl-F5	 	Kompilerer koden og genererer eksekvérbar fil for det sidst valgte program.	Compile/Run -> Compile Program
Kør programmet	F6	 	Kører det sidst valgte program (hvis det er blevet kompileret)	Compile/Run -> Run Program
Genkompilér og kør koden	Shift-F5		Kompilér og kør det sidst valgte program.	Compile/Run -> Recompile and Run
Debug programmet	F7		Debug det sidst valgte program (hvis det er blevet kompileret)	Compile/Run -> Debug Program

Operation	Tastegenvej
cut	ctrl-x
copy	ctrl-c
paste	ctrl-v
undo	ctrl-z
redo	ctrl-y

Tekst-mode

Operation	Tast	Mus	Effekt	Menuindgang
Gå til tekst-mode	ctrl-t ctrl-space		Starter tekst-editering af det markerede	Edit -> Edit Text
Gå ud af tekst-mode	ctrl-t ctrl-space	klik udenfor	Gen-fortolker den ændrede kode. Koden skal være 100% lovlig på det pågældende sted.	Edit -> Parse Text
Afbryd tekst-editering	escape		Glemmer de foretagne ændringer i teksten.	Edit -> Cancel Textediting