Velkommen til INF2100

Jeg er *Dag Langmyhr* (dag@ifi.uio.no).

Dagens tema:

- Hva går kurset ut på?
  - Bakgrunn for kurset
  - Hvordan gjennomføres kurset?
  - Hvordan får man det godkjent?
- Pause (med registrering av fremmøte)
- Programmeringsspråket RusC
  - En kort oversikt
  - Syntaks
  - Fire eksempler

Bakgrunnen for INF2100

I INF1000/INF1100 og INF1010 har dere lært å programmere, men bare små programmer (< 1000 linjer).

Hensikten med INF2100 er

- å gi mer programmeringstrening
- forstå mekanismene man trenger i større programmer (som objektorientering og moduler)
- gi trening i å lese og forstå andres programkode

Prosjektet

Prosjektet er valgt fordi det har en nytteverdi i seg selv:

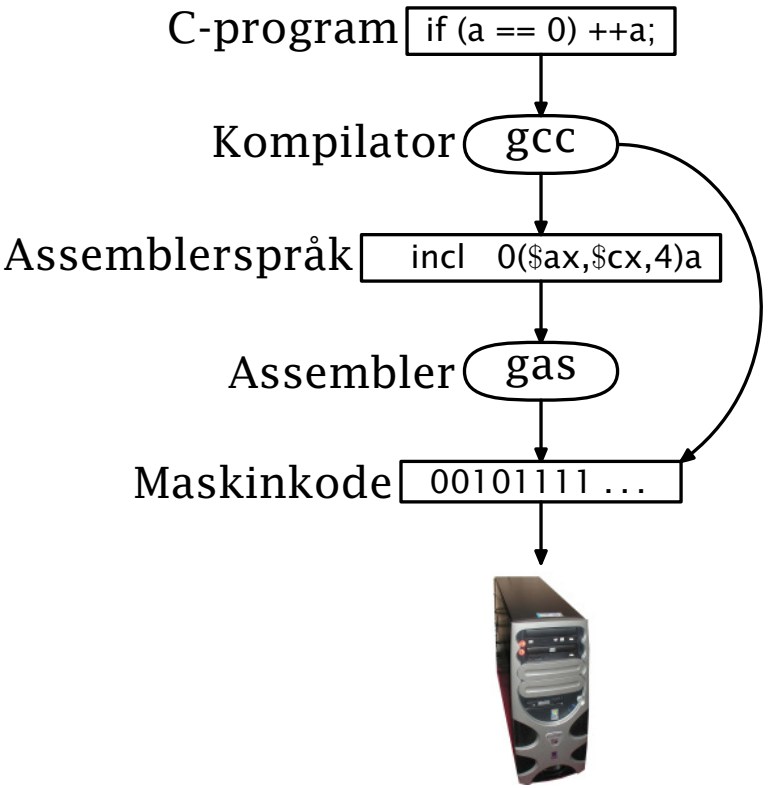
> *Skriv en kompilator for programmeringsspråket RusC.*

Dette gir

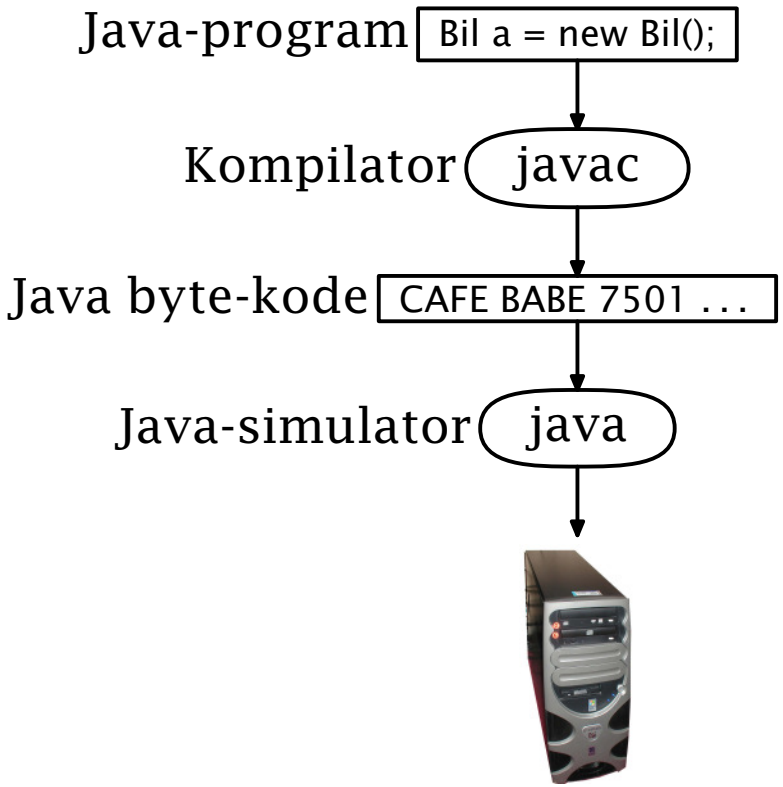
- forståelse for hvordan en kompilator fungerer
- kjennskap til maskin- og assemblerspråk
- eksempel på hvorledes et programmeringsspråk er definert og bygget opp

Hva gjør en kompilator?

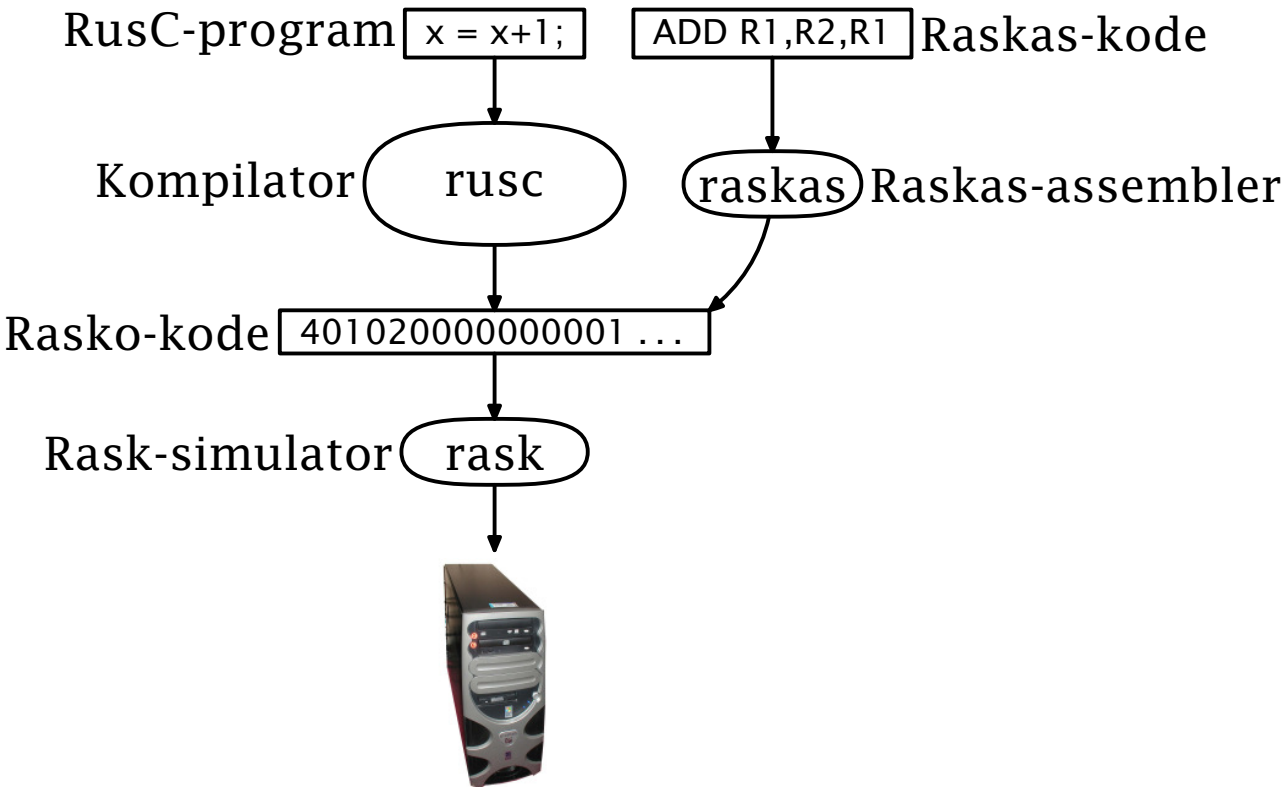
En kompilator oversetter et program til en annen kode, oftest maskin- eller assemblerkode.



Noen kompilatorer oversetter til en intern kode som *tolkes* av en interpret. (Det finnes også Java-kompilatorer som lager maskinkode, men de er ikke så vanlige.)



Opplegget for RusC ligner mye på Java-systemet.



Oppgaven deres er å skrive denne kompilatoren.

Ulike programmeringsspråk

I dette kurset skal vi innom flere språk:

Java benyttes til all implementasjon.

RusC er målet for prosjektet.

Rasko er maskinkoden til datamaskinen vår *Rask*.

Raskas er assemblerspråket til Rasko.

Oppbyggingen av kurset

Kurset har seks hovedkomponenter:

- **Forelesningnen** holdes i utgangspunktet nesten hver uke.
- **Gruppeøvelsene** er 2 timer hver uke.
- **Gruppearbeid** for å løse oppgavene. Man jobber to og to (eller alene om man vil). Man kan velge partner selv, eller la gruppelæreren plukke ut par.
- **Kompendiet** gir en grundig innføring i RusC og prosjektet.
- **Basiskoden** er utgangspunktet for programmeringen deres. Den skal bare utvides, ikke endres!
- **Nettsidene** er også en viktig informasjonskanal.

Godkjenning av kurset

Kurset har ikke karakterer, bare bestått/ikke bestått.

Det er tre obligatoriske oppgaver. Når de er godkjent, er kurset godkjent.

Men . . .

Mot slutten av semesteret vil noen bli plukket ut til en samtale om programmet de har laget. Dette kan man stryke på.

Siden man normalt jobber i lag, forventes at

- begge har kjennskap til hovedstrukturen i programmet
- begge kan identifisere sin del av programmet (som skal være rundt halvparten) der de kan forklare nøye hvorfor koden er blitt slik den er.

Samarbeid og fusk

Samarbeid og utveksling av ideer er bra!

Kopiering og fusk er ikke bra!

Gode råd for samarbeid

- Snakk gjerne med andre om ideer.
- Kopier aldri andres kode.

Ulike programmeringshjelpemidler

Når man skal programmere et større prosjekt, trenger man hjelpemidler:

- et spesialisert redigeringsprogram (à la Eclipse)
- et dokumentasjonsopplegg (à la JavaDoc)
- et versjonskontrollsystem (à la Subversion)
- ...

Dette kommer vi til mot slutten av kurset (når dere er motivert for å høre om dem ☺).

Programmeringsspråket RusC

Dette språket («Rudimentary simple C») er en meget forenklet utgave av C spesiallaget til dette kurset.

p0.rusc

```
int main ()
{
   putchar('!');   putchar(10);
}
```

p0.rusc

Programmet kompileres og kjøres med kommandoene

```
> rusc p0.rusc
> rask p0.rask
!
```

eller

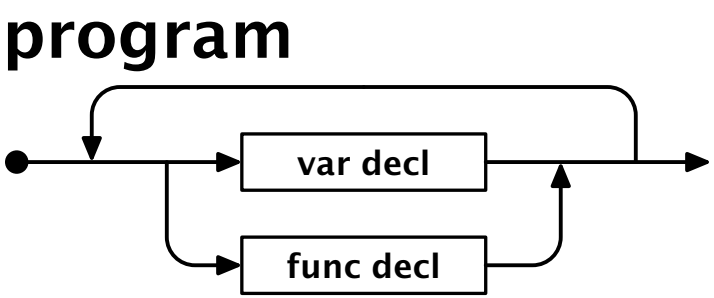
```
> java -jar /local/opt/inf2100/Rusc.jar p0.rusc
> java -jar /local/opt/inf2100/Rask.jar p0.rask
!
```

Siden RusC er forenklet C, kan vi også bruke C-kompilatoren til å kompilere koden (om vi tar med en definisjonsfil):

```
> more p1.rusc
#include "/local/opt/inf2100/include/rusc.h"
int main ()
{
  putchar('!');   putchar(10);
}
> gcc -x c -o p1 p1.rusc
> ./p1
!
```

Grammatikk

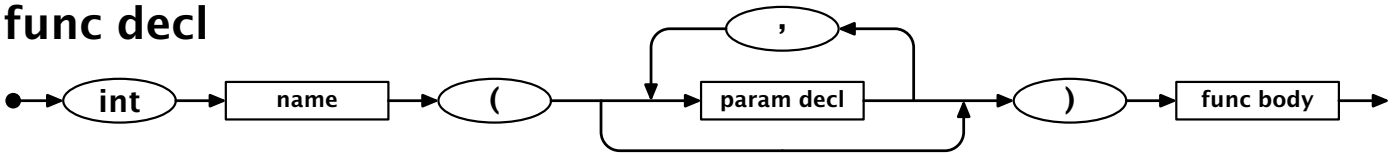
Vi kan beskrive grammatikken til RusC med *jernbanediagrammer*:



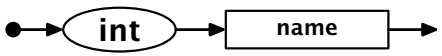
Et program er bare (som i C) en samling deklarasjoner av variable og funksjoner.

Funksjoner deklareres som i C og Java (men kan ha maksimalt fire parametre):

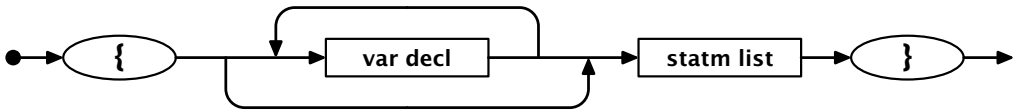
func decl



param decl



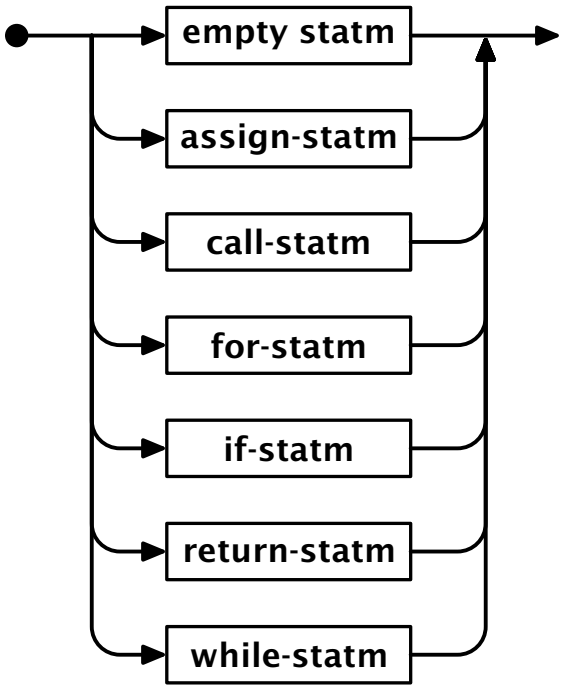
func body



Funksjonen main er hovedprogrammet.

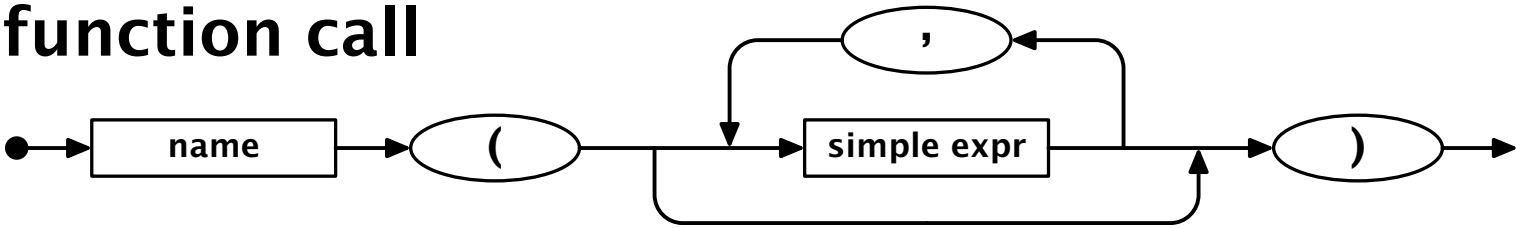
RusC har færre setninger enn C og Java:

statement



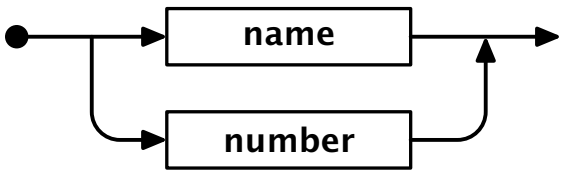
Funksjonskall er som forventet

function call



men parametrene kan bare være ganske enkle:

simple expr



Grammatikken forteller om et program er korrekt når det gjelder oppsettet (men det kan allikevel ha logiske feil).

Dette programmet er ikke korrekt:

```
int main ()
{
   int x;   x = 5;
   putint(-x);   putchar(10);
}
```

Kjøring gir

```
> rusc p1b.rusc
RusC error in line 4: A simple expression expected,
                      but found a subtractToken!
```

p2.rusc

```
int main ()
{
   int c, kind;

   putchar('?');  c = getchar();  /* Read a letter. */
   if (c >= 'a') {
      /* Convert to uppercase (if required) */
      c = c - 32;
   }

   kind = 'C';  /* Initially, assume a consonant. */
   if (c == 'A') { kind = 'V'; }
   else { if (c == 'E') { kind = 'V'; }
      else { if (c == 'I') { kind = 'V'; }
         else { if (c == 'O') { kind = 'V'; }
              else { if (c == 'U') { kind = 'V'; }}}}}

   putchar(kind);   putchar(10);   exit(0);
}
```

Eksempel 2

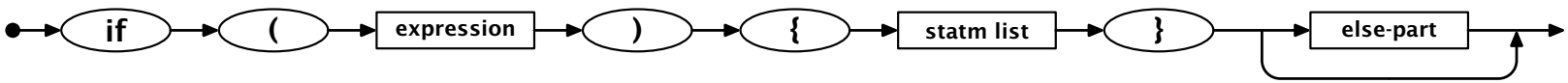
```
> rusc p2.rusc
> rask p2.rask
?f
C
```

Programmet leser en bokstav og avgjør om det en vokal eller konsonant.

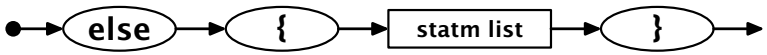
If-setninger

If-setninger tester på 0 (*usant*) og ikke-0 (*sant*).

if-statm

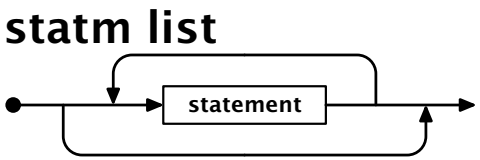


else-part



Legg merke til at det alltid skal være krøllparenteser.

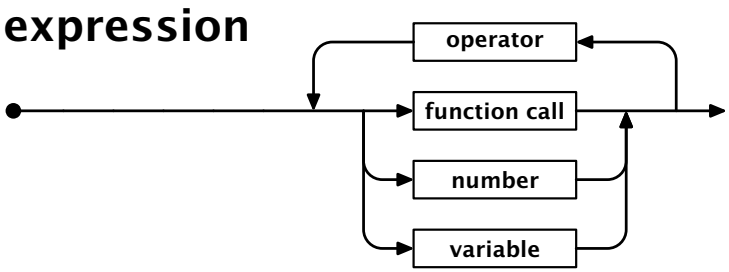
Setningslister inneholder 0 eller flere setninger:



Det finnes to former for kommentarer:

- /* ... */
  kan gå over flere linjer.
- # ...
  er en ren kommentarlinje (om # står helt til venstre)

Eksempel 2

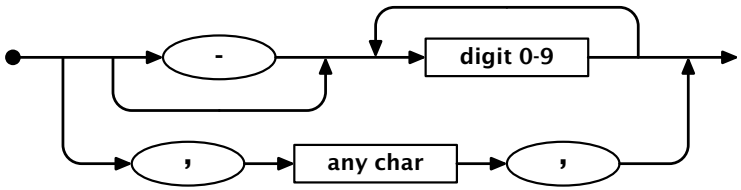


- Lovlige operatorer er de fire regneartene (+, –, *, /) og seks sammenligningene (==, !=, <, <=, >, >=).
- De har samme *presedens*.
- Vi har ikke parenteser i uttrykkene.

Derfor må vi bruke flere variable enn vi er vant til.

Et tegn (som 'a') er bare en annen notasjon for et tall (97).

number



ISO 8859–1

0	000 00	32		040 20	64	@	100 40	96	‘	140 60	128		200 80	160		240 A0	192	À	300 C0	224	à	340 E0
1	001 01	33	!	041 21	65	A	101 41	97	a	141 61	129		201 81	161	¡	241 A1	193	Á	301 C1	225	á	341 E1
2	002 02	34	"	042 22	66	B	102 42	98	b	142 62	130		202 82	162	¢	242 A2	194	Â	302 C2	226	â	342 E2
3	003 03	35	#	043 23	67	C	103 43	99	c	143 63	131		203 83	163	£	243 A3	195	Ã	303 C3	227	ã	343 E3
4	004 04	36	$	044 24	68	D	104 44	100	d	144 64	132		204 84	164	¤	244 A4	196	Ä	304 C4	228	ä	344 E4
5	005 05	37	%	045 25	69	E	105 45	101	e	145 65	133		205 85	165	¥	245 A5	197	Å	305 C5	229	å	345 E5
6	006 06	38	&	046 26	70	F	106 46	102	f	146 66	134		206 86	166	¦	246 A6	198	Æ	306 C6	230	æ	346 E6
7	007 07	39	’	047 27	71	G	107 47	103	g	147 67	135		207 87	167	§	247 A7	199	Ç	307 C7	231	ç	347 E7
8	010 08	40	(	050 28	72	H	110 48	104	h	150 68	136		210 88	168	¨	250 A8	200	È	310 C8	232	è	350 E8
9	011 09	41	)	051 29	73	I	111 49	105	i	151 69	137		211 89	169	©	251 A9	201	É	311 C9	233	é	351 E9
10	012 0A	42	*	052 2A	74	J	112 4A	106	j	152 6A	138		212 8A	170	ª	252 AA	202	Ê	312 CA	234	ê	352 EA
11	013 0B	43	+	053 2B	75	K	113 4B	107	k	153 6B	139		213 8B	171	«	253 AB	203	Ë	313 CB	235	ë	353 EB
12	014 0C	44	,	054 2C	76	L	114 4C	108	l	154 6C	140		214 8C	172	¬	254 AC	204	Ì	314 CC	236	ì	354 EC
13	015 0D	45	–	055 2D	77	M	115 4D	109	m	155 6D	141		215 8D	173	-	255 AD	205	Í	315 CD	237	í	355 ED
14	016 0E	46	.	056 2E	78	N	116 4E	110	n	156 6E	142		216 8E	174	®	256 AE	206	Î	316 CE	238	î	356 EE
15	017 0F	47	/	057 2F	79	O	117 4F	111	o	157 6F	143		217 8F	175	¯	257 AF	207	Ï	317 CF	239	ï	357 EF
16	020 10	48	0	060 30	80	P	120 50	112	p	160 70	144		220 90	176	°	260 B0	208	Đ	320 D0	240	ð	360 F0
17	021 11	49	1	061 31	81	Q	121 51	113	q	161 71	145		221 91	177	±	261 B1	209	Ñ	321 D1	241	ñ	361 F1
18	022 12	50	2	062 32	82	R	122 52	114	r	162 72	146		222 92	178	²	262 B2	210	Ò	322 D2	242	ò	362 F2
19	023 13	51	3	063 33	83	S	123 53	115	s	163 73	147		223 93	179	³	263 B3	211	Ó	323 D3	243	ó	363 F3
20	024 14	52	4	064 34	84	T	124 54	116	t	164 74	148		224 94	180	´	264 B4	212	Ô	324 D4	244	ô	364 F4
21	025 15	53	5	065 35	85	U	125 55	117	u	165 75	149		225 95	181	µ	265 B5	213	Õ	325 D5	245	õ	365 F5
22	026 16	54	6	066 36	86	V	126 56	118	v	166 76	150		226 96	182	¶	266 B6	214	Ö	326 D6	246	ö	366 F6
23	027 17	55	7	067 37	87	W	127 57	119	w	167 77	151		227 97	183	·	267 B7	215	×	327 D7	247	÷	367 F7
24	030 18	56	8	070 38	88	X	130 58	120	x	170 78	152		230 98	184	¸	270 B8	216	Ø	330 D8	248	ø	370 F8
25	031 19	57	9	071 39	89	Y	131 59	121	y	171 79	153		231 99	185	¹	271 B9	217	Ù	331 D9	249	ù	371 F9
26	032 1A	58	:	072 3A	90	Z	132 5A	122	z	172 7A	154		232 9A	186	º	272 BA	218	Ú	332 DA	250	ú	372 FA
27	033 1B	59	;	073 3B	91	[	133 5B	123	{	173 7B	155		233 9B	187	»	273 BB	219	Û	333 DB	251	û	373 FB
28	034 1C	60	<	074 3C	92	\	134 5C	124	|	174 7C	156		234 9C	188	¼	274 BC	220	Ü	334 DC	252	ü	374 FC
29	035 1D	61	=	075 3D	93	]	135 5D	125	}	175 7D	157		235 9D	189	½	275 BD	221	Ý	335 DD	253	ý	375 FD
30	036 1E	62	>	076 3E	94	^	136 5E	126	~	176 7E	158		236 9E	190	¾	276 BE	222	Þ	336 DE	254	þ	376 FE
31	037 1F	63	?	077 3F	95	_	137 5F	127		177 7F	159		237 9F	191	¿	277 BF	223	ß	337 DF	255	ÿ	377 FF

p3.rusc

```
int LF;

int gcd (int a, int b)
{
   while (a != b) {
      if (a < b) {
         b = b-a;
      } else {
         a = a-b;
      }
   }
   return a;
}
```

```
int main ()
{
   int v1, v2, res;

   LF = 10;
   putchar('?');
   v1 = getint();   v2 = getint();
   res = gcd(v1,v2);
   putint(res);  putchar(LF);
}
```

Programmet finner *største felles divisor*:

```
> rusc p3.rusc
> rask p3.rask
52  221
13
```

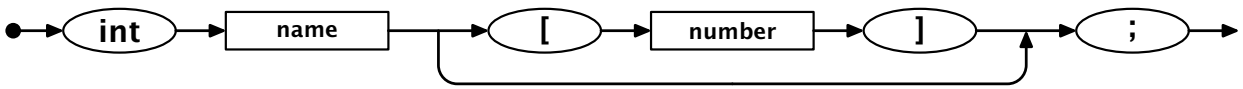
Biblioteket

RusC kjenner til disse fem funksjonene:

exit(*status*)	Avslutter programmet
getchar()	Leser et tegn fra tastaturet
getint()	Leser et heltall fra tastaturet
putchar(*n*)	Skriver ut et tegn på skjermen
putint(*n*)	Skriver ut et heltall på skjermen

Vektorer

var decl




```
int a [3];
```

deklarerer a med elementene a[0], a[1] og a[2]. Det er ingen sjekk på grensene.

primes.rusc, del 1

```
/* Program 'primes'
   ----------------
   Finds all prime numbers up to 1000 (using the technique called
   "the sieve of Eratosthenes") and prints them nicely formatted.
*/

#include "/local/opt/inf2100/include/rusc.h"

int prime[1001];  /* The sieve */
int LF;           /* LF */
```

primes.rusc, del 2

```
int find_primes ()
{
   /* Remove all non-primes from the sieve: */

   int i1;  int i2;

   for (i1 = 2;  i1 <= 1000;  i1 = i1+1) {
      for (i2 = 2*i1;  i2 <= 1000;  i2 = i2+i1) {
         prime[i2] = 0;
      }
   }
}
```

primes.rusc, del 3

```
int mod (int a, int b)
{
   /* Computes a%b. */

   int ax;

   ax = a/b;  ax = ax*b;
   return a - ax;
}
```

primes.rusc, del 4

```
int p3 (int v)
{
   /* Does a ‘printf("%3d", v)´;
      assumes 0<=v<=999. */

   if (v <= 9) {
      putchar(’ ’);   putchar(’ ’);
   } else {
      if (v <= 99) {
         putchar(’ ’);
      };
   }
   putint(v);
}
```

primes.rusc, del 5

```
int print_primes ()
{
   /* Print the primes, 10 on each line. */

   int n_printed;  int i;

   n_printed = 0;
   for (i = 1;  i <= 1000;  i = i + 1) {
      if (prime[i]) {
         if (mod(n_printed,10) == 0 * n_printed) {
            putchar(LF);
         }
         putchar(' ');   p3(i);   n_printed = n_printed+1;
      }
   }
   putchar(LF);
}
```

primes.rusc, del 6

```
int main ()
{
   int i;

   LF = 10;
   /* Initialize the sieve by assuming all numbers >1 to be primes: */
   prime[1] = 0;
   for (i=2; i<=1000; i=i+1) { prime[i] = 1; }

   /* Find and print the primes: */
   find_primes();  print_primes();
}
```

Eksempel 4

```
> rusc primes.rusc
> rask primes.rask
  2   3   5   7  11  13  17  19  23  29
 31  37  41  43  47  53  59  61  67  71
 73  79  83  89  97 101 103 107 109 113
127 131 137 139 149 151 157 163 167 173
179 181 191 193 197 199 211 223 227 229
233 239 241 251 257 263 269 271 277 281
283 293 307 311 313 317 331 337 347 349
353 359 367 373 379 383 389 397 401 409
419 421 431 433 439 443 449 457 461 463
467 479 487 491 499 503 509 521 523 541
547 557 563 569 571 577 587 593 599 601
607 613 617 619 631 641 643 647 653 659
661 673 677 683 691 701 709 719 727 733
739 743 751 757 761 769 773 787 797 809
811 821 823 827 829 839 853 857 859 863
877 881 883 887 907 911 919 929 937 941
947 953 967 971 977 983 991 997
```

Oppsummering

RusC er et meget enkelt programmeringsspråk.

- Det bør være enkelt å lære.
- Ikke alt man forventer, er lov. Sjekk grammatikken!