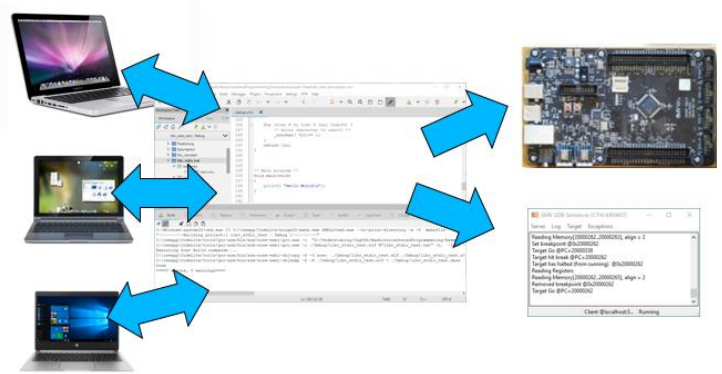
Introduktion till Maskinorienterad programmering

Målsättningar:

- ✓ Bli bekant med utvecklingsmiljön
- ✓ Kunna redigera, kompilera och testa ett enkelt C-program

Ur innehållet:

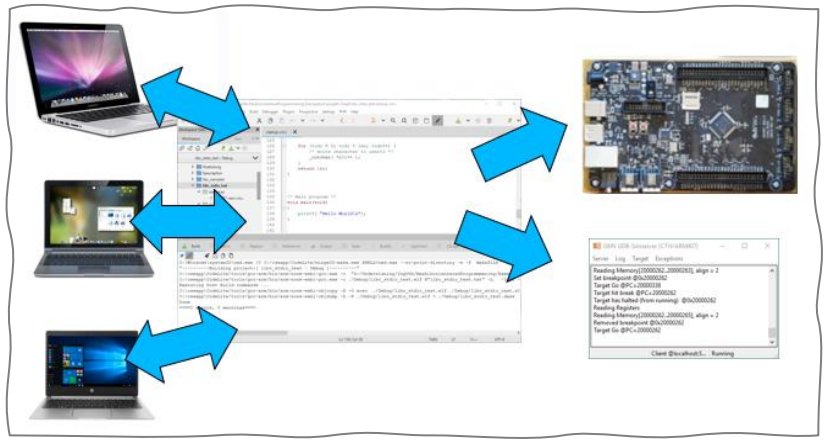
- ✓ Programutveckling i C och assembler
- ✓ Programutvecklingsmiljö
- ✓ Så översätts ett C-program



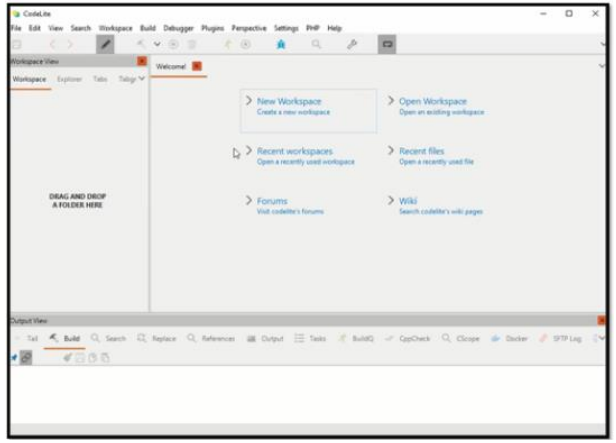
Introduktion till C

- ✓ Variabeldeklarationer
- ✓ Heltalstyper
- ✓ Funktioner
- ✓ Textsträngar

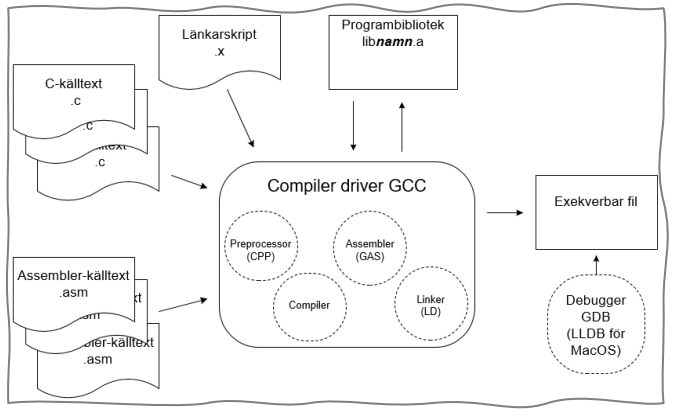
Programutveckling i C



Korsutveckling

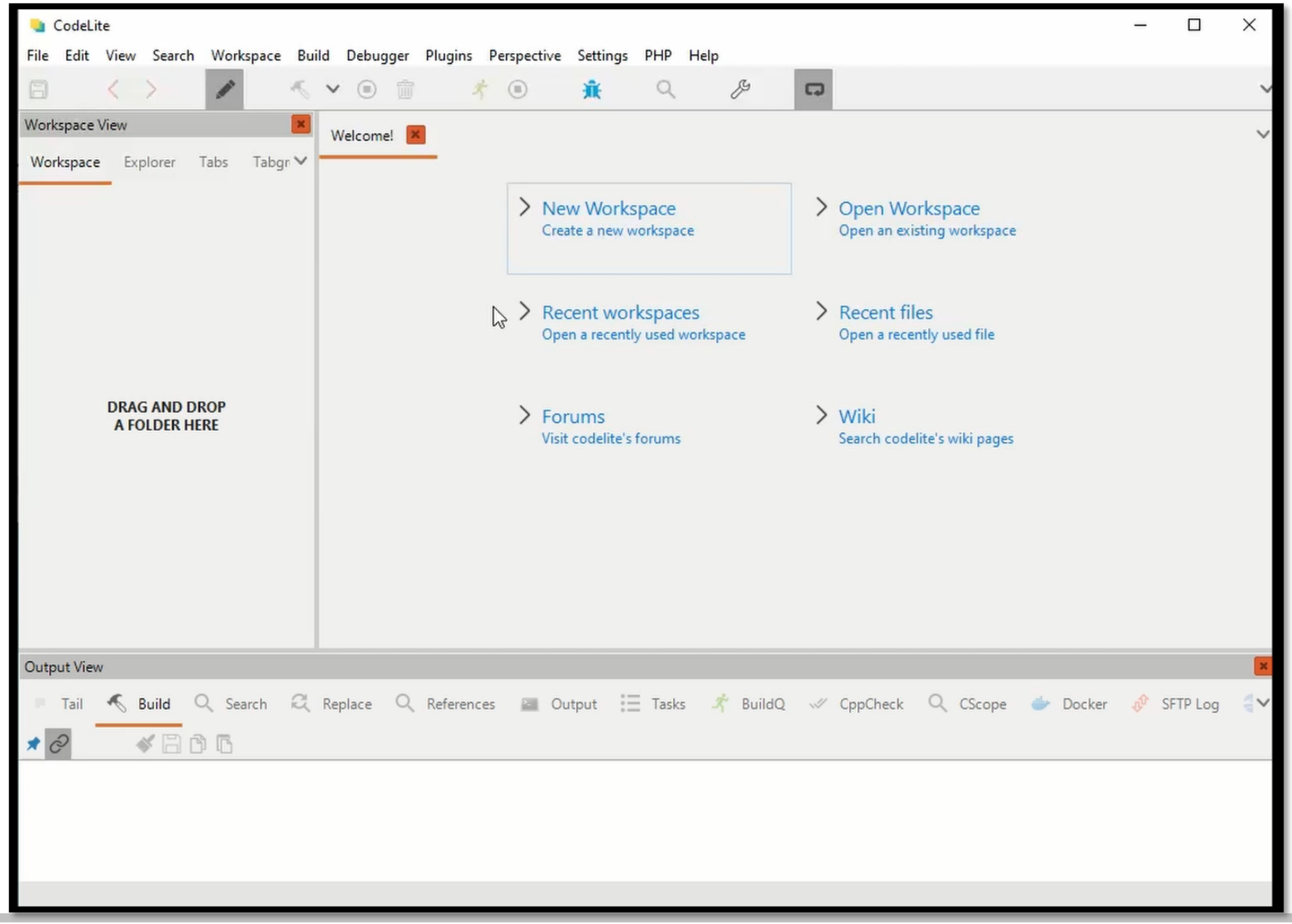


CodeLite

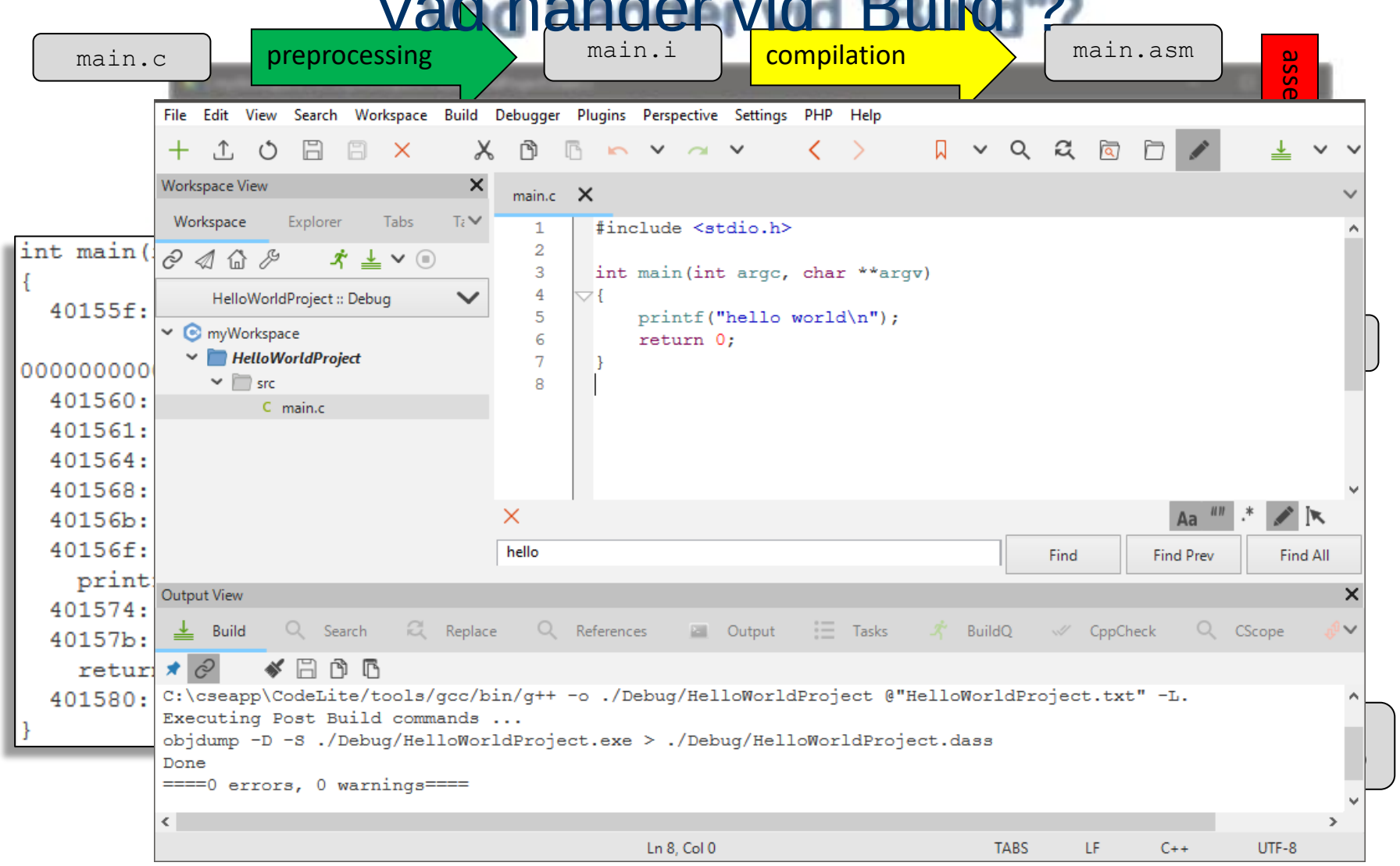


GCC – Gnu Compiler Collection

"hello world"


CodeLite
File Edit View Search Workspace Build Debugger Plugins Perspective Settings PHP Help
Workspace View
Workspace Explorer Tabs Tabgr
DRAG AND DROP
A FOLDER HERE
Welcome!
New Workspace
Create a new workspace
Open Workspace
Open an existing workspace
Recent workspaces
Open a recently used workspace
Recent files
Open a recently used file
Forums
Visit codelite's forums
Wiki
Search codelite's wiki pages
Output View
Tail Build Search Replace References Output Tasks BuildQ CppCheck CScope Docker SFTP Log

Vad händer vid "Build"?



Typsystemet i C – en första översikt

Vi använder färgad syntax för de reserverade orden i C

Heltalstyper

```
/* Exempel: */
char c;
short si;
long li;
int i;
```

heltalstyperna kan vidare delas upp i tal utan tecken (`unsigned`) respektive tal med tecken (`signed`).

Flyttal

```
/* Exempel: */
float f;
double d;
long double ld;
```

.... i kursen berör vi bara undantagsvis användning av flyttal

Dessutom har vi

- fält
- pekartyper
- sammansatta typer
- bitfält
- uppräkningstyp
- ”egendefinierad” typ

.... under kursens gång behandlar vi dessa efter hand i olika omfattning

Heltalstyper i C

OBS: Typen `char` kan vara synonym med `signed char` eller `unsigned char` beroende på kompilator
Exempel GCC/x86: `signed char`, men GCC/ARM: `unsigned char`

Deklaration	Förklaring
`char`	Minsta adresserbara enhet som kan rymma en basal teckenuppsättning. Det är dock en heltalstyp som kan vara signed eller unsigned, detta är implementationsberoende.
`signed char`	Tolkas som heltal med tecken. Måste **minst** kunna representera talområdet [−127, +127], dvs. minst 8 bitar.
`unsigned char`	Tolkas som heltal utan tecken. Måste **minst** kunna representera talområdet [0, 255], dvs. minst 8 bitar.

`short`
`short int`
`signed short`
`signed short int`
	Kort heltalstyp med tecken. Måste **minst** kunna representera talområdet [−32767, +32767], dvs. minst 16 bitar. Observera att talområdet [−32768, +32767] som fås vid 2-komplementsrepresentation, också är tillåtet.

`unsigned short` `unsigned short int`	Kort heltalstyp utan tecken. Måste **minst** kunna representera talområdet [0, +65535], dvs. minst 16 bitar.

`long`
`long int`
`signed long`
`signed long int`
	Lång heltalstyp med tecken. Måste **minst** kunna representera talområdet [−2147483647, +2147483647], dvs. minst 32 bitar.

`unsigned long` `unsigned long int`	Lång heltalstyp utan tecken. Måste **minst** kunna representera talområdet [0, +4294967295], dvs. minst 32 bitar.

`int`
`signed`
`signed int`
	Implementationsbestämd heltalstyp med tecken. Måste **minst** kunna representera talområdet [−32767, +32767], dvs. minst 16 bitar. Observera att talområdet [−32768, +32767] som fås vid 2-komplementsrepresentation, också är tillåtet.
	Observera speciellt att `int` kan vara synonym med `short` eller `long`.

`unsigned` `unsigned int`	Typen `int` utan tecken. Måste **minst** kunna representera talområdet [0, +65535], dvs. minst 16 bitar.

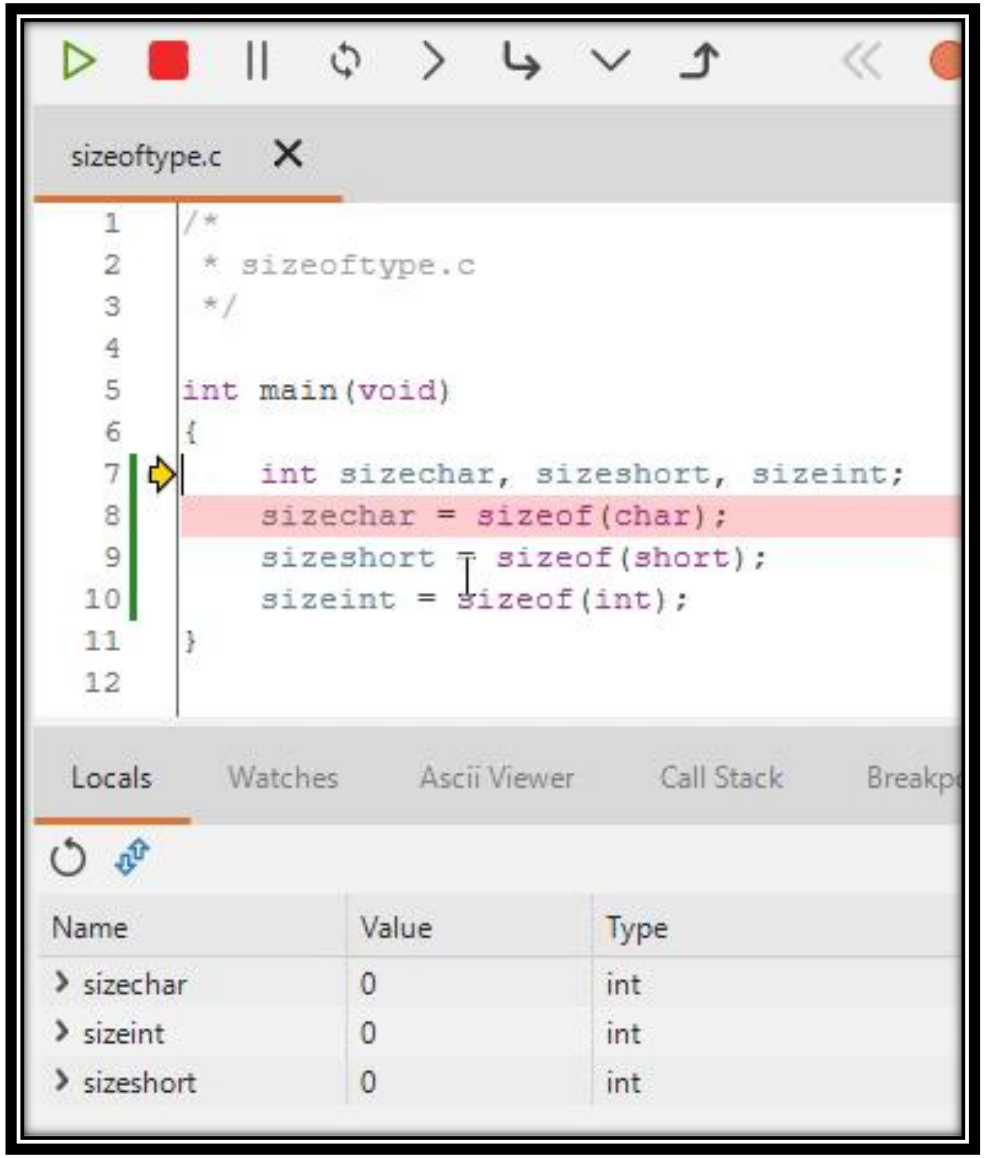
Hur stor är en typ?

Storleken av en typ, i antal bytes, ges av C-operatorn

```
size = sizeof(typ)
```

Exempel:
Vi söker *antalet bitar* hos en `unsigned long`:

```
= sizeof( unsigned long ) * 8;
```



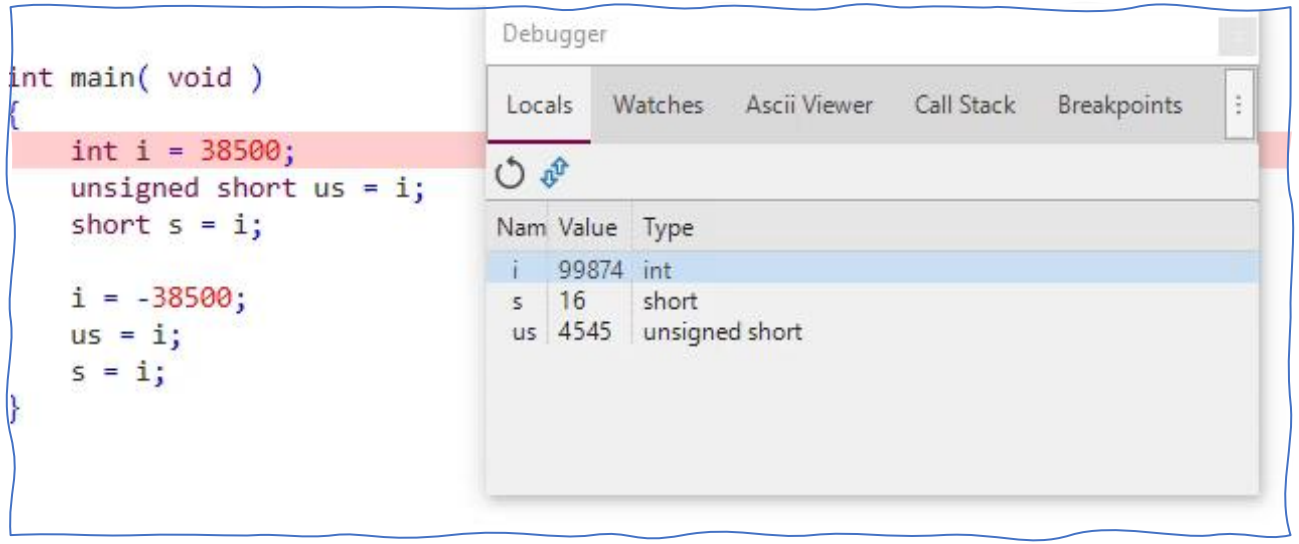
Varför skiljer vi på heltalstyperna?

Vi vill använda tillgängligt minne så effektivt som möjligt.
Vi har möjlighet att anpassa storlek, men även tecken

Exempel:

datatyp	antal bytes	unsigned	signed
char	1	0..255	-128..127
short	2	0..65535	-32768..32767
long	4	0..4294967295	-2147483648.. 2147483647

men att blanda typerna kan ha sina risker...



Typkonvertering (*casting*)

Konvertering mellan typer (eng: *cast)*, kallas allmänt för typkonvertering (casting).

```
/*
 * castings.c
 */

int main(void)
{
   char  c;
   short s;
   int   i;

   i = s + c;
   i = (int) s + (int) c;
   return 0;
}
```

Implicit typkonvertering, görs automatiskt av kompilator

Explicit typkonvertering, programmerare anger hur konverteringen ska utföras

Implicit typkonvertering görs efter regler bestämda i programspråket.

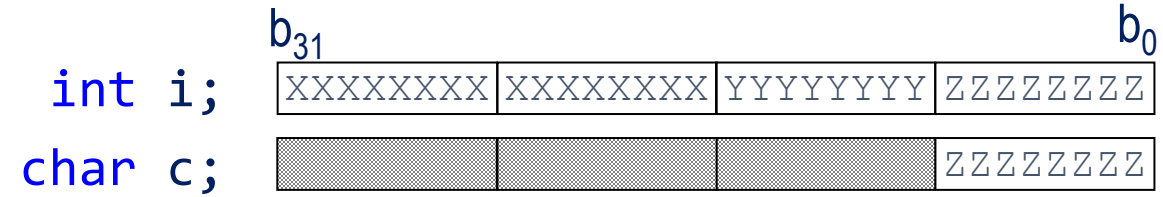
Vissa typkonverteringar kan generera varningsmeddelanden...
...medan andra (felaktiga konverteringar) genererar felmeddelanden.

Konvertering från *mindre* till *större* typ (*widening*) görs med *teckenutvidgning*.

Konvertering från *större* till *mindre* typ (*narrowing*) görs med *trunkering*.

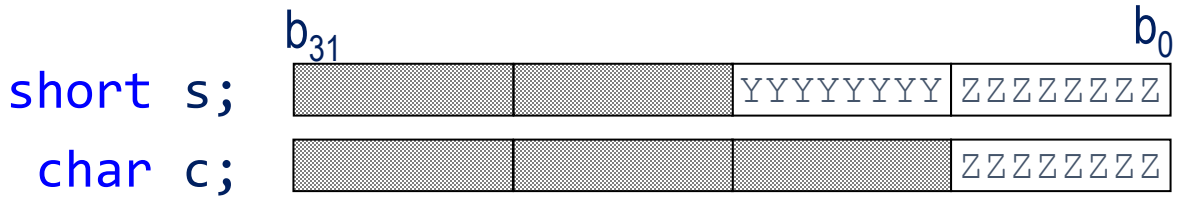
Exempel: *trunkering*

Antag: `sizeof(char)` är 1,
`sizeof(short)` är 2
och `sizeof(int)` är 4



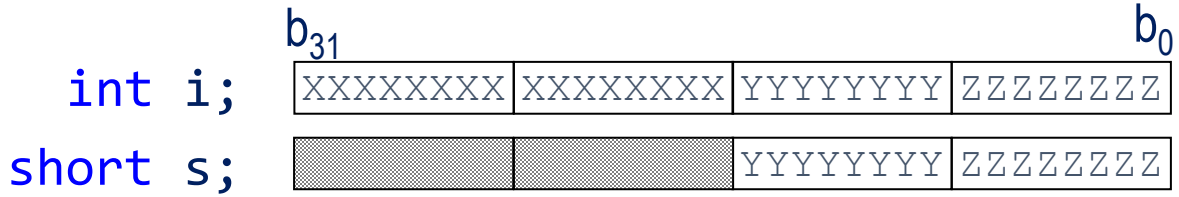
Bitar som förloras vid en tilldelning:

```
c = i;
```



Bitar som förloras vid en tilldelning:

```
c = s;
```



Bitar som förloras vid en tilldelning:

```
s = i;
```

Exempel: *teckenutvidgning*

Vi har deklarationerna: `signed char c; unsigned char uc;`

`sizeof(char)` är 1
`sizeof(long)` är 4

`signed char c;`

?	?	?	0XXXXXXX	→	00000000	00000000	00000000	0XXXXXXX
?	?	?	1XXXXXXX	→	11111111	11111111	11111111	1XXXXXXX

`(long) c;`

`unsigned char uc;`

?	?	?	0XXXXXXX	→	00000000	00000000	00000000	0XXXXXXX
?	?	?	1XXXXXXX	→	00000000	00000000	00000000	1XXXXXXX

`(long) uc;`

`sizeof(short)` är 2

Vi har deklarationerna: `signed short s; unsigned short us;`

`signed short s;`

?	?	0XXXXXXX	XXXXXXXX	→	00000000	00000000	0XXXXXXX	XXXXXXXX
?	?	1XXXXXXX	XXXXXXXX	→	11111111	11111111	1XXXXXXX	XXXXXXXX

`(long) s;`

`unsigned short s;`

?	?	0XXXXXXX	XXXXXXXX	→	00000000	00000000	0XXXXXXX	XXXXXXXX
?	?	1XXXXXXX	XXXXXXXX	→	00000000	00000000	1XXXXXXX	XXXXXXXX

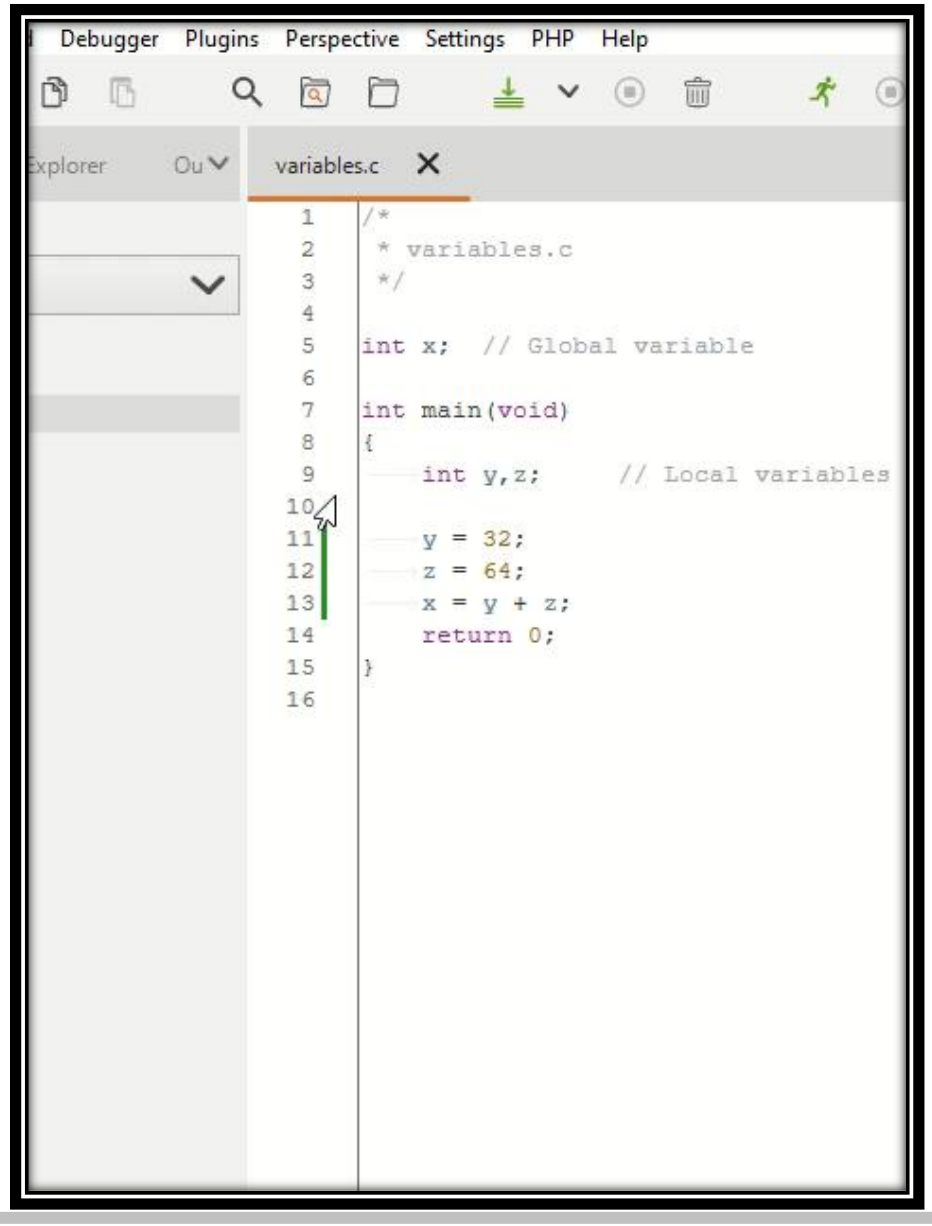
`(long) s;`

Variabler, deklarationer och tilldelningar

```
/*
 * variables.c
 */

int x;              /* Global variabel */

int main(void)
{
   int y,z;         /* Lokal variabel */

   y = 32;
   z = 64;
   x = y + z;
   return 0;
}
```

Deklarationer

Tilldelningar



C-funktioner, parametrar och returvärden

Inget returvärde

Parametrar ("argument")

```
void foo(int x, char y)
{
   int sum;

   /* Satser ... */

}
```

Parametrar skickas med värdeanrop (”pass-by value”)

Exempel:

```
char var2 = 7;
foo(5, var2);
```

`var2` har fortfarande värdet 7 efter funktionen

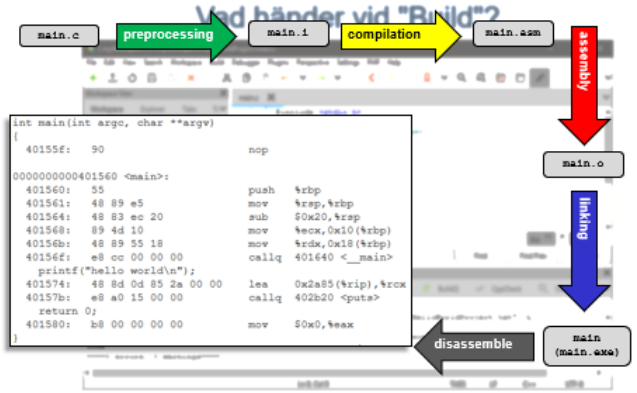
Returvärde har funktionens typ

```
int foo1(int x, char y)
{
   int sum;

   /* Satser ... */
   return sum;
}
```

Programstruktur

För att ge programmet en god struktur kan källtexterna delas upp i olika delar.

- Filer med ändelsen `.c`, innehållet i dessa genererar så småningom maskinkod.
- Filer med ändelsen `.h` (*header-filer*) används för *deklarationer*, *användardefinierade typer* och *makrodefinitioner*. De sistnämnda återkommer vi till




```c
/* main.c*/
#include <stdio.h>
#include "decl.h"

int main()
{
    int x;
    char c;
    ...
    printf("foo0=: %d\n", foo0(x));
    foo1();
    printf("foo2=: %c\n", foo2((short)x));
    return 0;
}
```

```c
/* decl.h */

/* user defined types */
...
/* macros */
...
/* function prototypes */
int foo0(int x);
void foo1(void);
char foo2(short x);
```

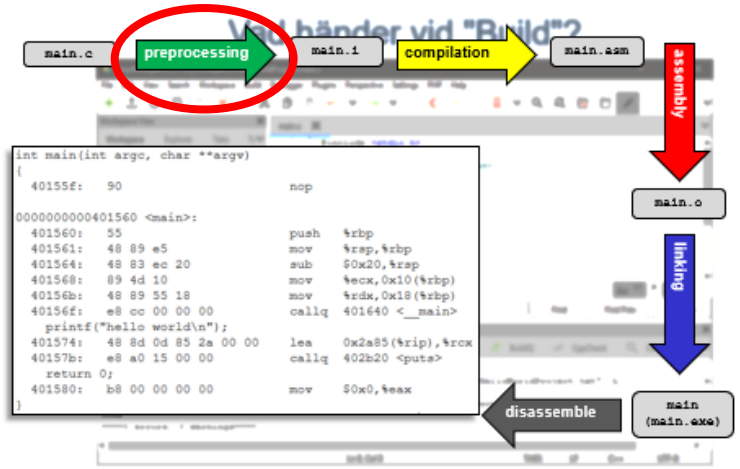
```c
/* foo0.c */
#include "decl.h"
```

```c
/* foo1.c */
#include "decl.h"
```

```c
/* foo2.c */
#include "decl.h"

/* Function definition */
char foo2(short x)
{
    ...
}
```

"Pre-processing"

Pre-processorn utför *textsubstitution*!




```
// main.c
#include <stdio.h>

#define MAX_SCORE 100
#define SQUARE(x) (x)*(x)

int main()
{
   printf(”Highest score is %d\n", MAX_SCORE);
   printf(”Square of 3 is %d\n", SQUARE(3));
   return 0;
}
```

infoga en annan fil

"macro" definitioner

preprocessing

main.i

```
...
    innehåll "stdio.h"
...

int main()
{
   printf(”Highest score is %d\n", 100);
   printf(”Square of 3 is %d\n", 3*3 );
   return 0;
}
```

Att styra villkorlig kompilering

Direktiven: `#define, #undef, #if, #elif, #else, #endif, #ifdef, #ifndef`

Preprocessor conditional inclusion

```
#define X  1              // syntax: #define [identifier name] [value], where [value] is optional
#define SIMULATOR         // symbol SIMULATOR is defined without a value

#if X == 0          // syntax: #if <value>, where 0=false and !0==true. Integer arithmetic OK.
    …         // any C-code
#elif X-1 == 1      // elif means else if
    …
#else
    …
#endif

#if 0  // good for temporarily commenting out large blocks of code.
 …     // any C-code
#endif

#define HW_DEBUG
…
#ifdef HW_DEBUG
    … // any C-code, for example:
    #undef SIMULATOR             // changed my mind...
#endif
```

```
#define SIMULATOR
void delay_500ns(void)
{
#ifndef SIMULATOR
      delay_250ns();
      delay_250ns();
#endif
}
```

Att undvika "multiple definitions"

"Include guards" används för att undvika att samma .h-fil inkluderas flera gånger från någon .c-fil.

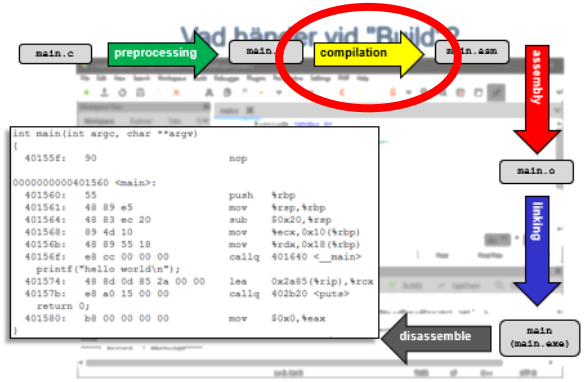
```c
// main.c
#include <stdio.h>
#include "declarations.h"
#include "more_declarations.h"

int main()
{
    int x;
    char c;
    ...
    printf("foo0=: %d\
    foo1();
    printf("foo2=: %c\
    return 0;
}
```

```c
#ifndef   _MORE_DECLARATIONS_H
#define   _MORE_DECLARATIONS_H
// more_declarations.h
...
#include "declarations.h"
...
#endif
```

```c
#ifndef  _DECLARATIONS_H
#define  _DECLARATIONS_H
// declarations.h

// user defined types
...
// macros
...
// function prototypes
int foo0(int x);
void foo1(void);
char foo2(short x);
#endif
```

Kompilering (C-kod till assemblerkod)

Vid översättningen skapas assemblerkod för den valda processorn, här Intel X86



main.i

```
...
    innehåll "stdio.h"
...

int main()
{
   printf(”Highest score is %d\n", 100);
   printf(”Square of 3 is %d\n", 3*3 );
   return 0;
}
```

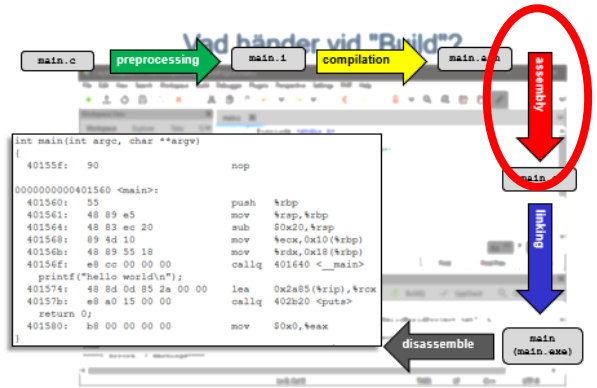
compilation

main.asm

```
main:
        push    %rbp
        mov     %rsp,%rbp
        sub     $0x20,%rsp
{
        printf("Highest score is %d\n", 100);
        mov     $0x64,%edx
        lea     0x2a87(%rip),%rcx
        callq   402bb8 <printf>
        printf("Square of 3 is %d\n", 3*3 );
        mov     $0x9,%edx
        lea     0x2a8b(%rip),%
        callq   402bb8 <printf>
        return 0;
        mov     $0x0,%eax
}
        add     $0x20,%rsp
        pop     %rbp
        retq
```

Assemblering, assemblerkod till maskinkod

Översättningen från assemblerkod till objektkod (maskinkod) görs av assemblatorn.
I objektkoden finns även symbolinformation för externa funktioner och variabler




```
int main()
{
        printf("Highest score is %d\n", 100);
        printf("Square of 3 is %d\n", 3*3 );
```

main.asm

```
main:
        push    %rbp
        mov     %rsp,%rbp
        sub     $0x20,%rsp
{
        printf("Highest score is %d\n", 100);
        mov     $0x64,%edx
        lea     0x2a87(%rip),%rcx
        callq   402bb8 <printf>
        printf("Square of 3 is %d\n", 3*3 );
        mov     $0x9,%edx
        lea     0x2a8b(%rip),%
        callq   402bb8 <printf>
        return 0;
        mov     $0x0,%eax
}
        add     $0x20,%rsp
        pop     %rbp
        retq
```

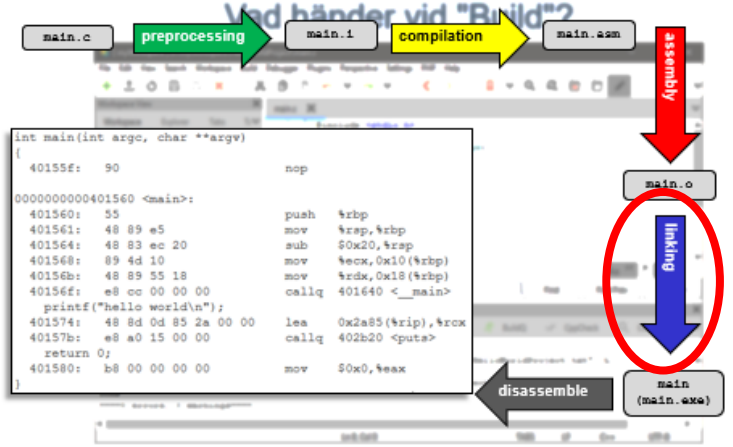
assembly

main.o

```
<main>:
55
48 89 e5
48 83 ec 20
e8 03 01 00 00
        printf("Highest score is %d\n", 100);
ba 64 00 00 00
48 8d 0d 87 2a 00 00
e8 3a 16 00 00
        printf("Square of 3 is %d\n", 3*3 );
ba 09 00 00 00
48 8d 0d 8b 2a 00 00
e8 29 16 00 00
        return 0;
b8 00 00 00 00
}
48 83 c4 20
5d
c3
```

Länkning

Vid länkningen kombineras de olika objektfilerna till en exekverbar fil.
Symboler översätts till relativa adresser.
Standardbiblioteket `libc` innehåller en lång rad användbara funktioner, som exempelvis `printf`



main.o

```
<main>:
55
48 89 e5
48 83 ec 20
e8 03 01 00 00
        printf("Highest score is
%d\n", 100);
ba 64 00 00 00
48 8d 0d 87 2a 00 00
e8 3a 16 00 00
        printf("Square of 3 is
%d\n", 3*3 );
ba 09 00 00 00
48 8d 0d 8b 2a 00 00
e8 29 16 00 00
        return 0;
b8 00 00 00 00
}
48 83 c4 20
5d
c3
```

Länkas samman till:

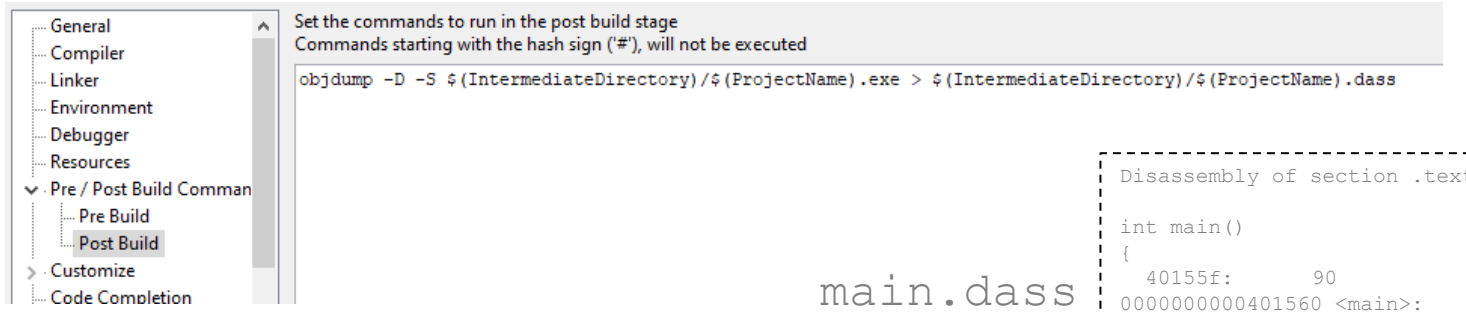
main.exe

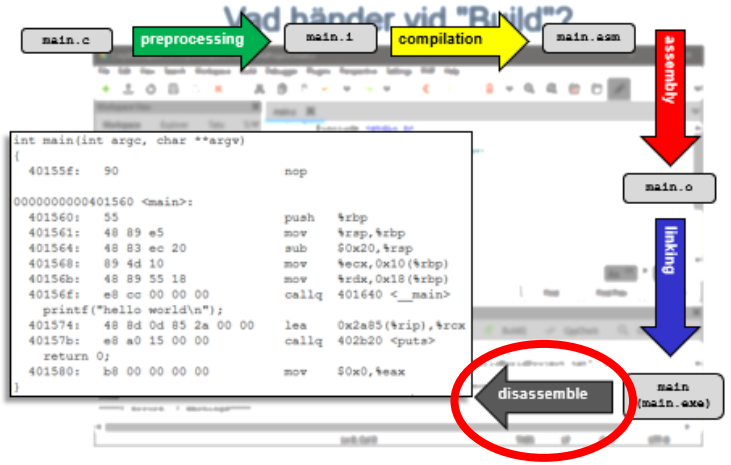
`printf` hämtas från programbibliotek `libc`:

```
0000000000402bb8 <printf>:
  402bb8:         ff 25 86 57 00 00
         jmpq   *0x5786(%rip)        #
408344 <__imp_printf>
  402bbe:         90
         nop
  402bbf:         90
         nop
```

Disassemblering

Under fliken Settings kan man låta `objdump` *återskapa* assemblerkoden från maskinkod (”disassemblering”)...





main.dass

```
Disassembly of section .text:

int main()
{
  40155f:       90                      nop
0000000000401560 <main>:
  401560:       55                      push   %rbp
  401561:       48 89 e5                mov    %rsp,%rbp
  401564:       48 83 ec 20             sub    $0x20,%rsp
  401568:       e8 03 01 00 00          callq  401670 <__main>
      printf("Highest score is %d\n", 100);
  40156d:       ba 64 00 00 00          mov    $0x64,%edx
  401572:       48 8d 0d 87 2a 00 00    lea    0x2a87(%rip),%rcx        # 404000 <.rdata>
  401579:       e8 3a 16 00 00          callq  402bb8 <printf>
      printf("Square of 3 is %d\n", 3*3 );
  40157e:       ba 09 00 00 00          mov    $0x9,%edx
  401583:       48 8d 0d 8b 2a 00 00    lea    0x2a8b(%rip),%rcx        # 404015
<.rdata+0x15>
  40158a:       e8 29 16 00 00          callq  402bb8 <printf>
      return 0;
  40158f:       b8 00 00 00 00          mov    $0x0,%eax
}
  401594:       48 83 c4 20             add    $0x20,%rsp
  401598:       5d                      pop    %rbp
  401599:       c3                      retq
```

Textsträngar och `printf`

En textsträng består av ASCII-tecken och avslutas med 0 (NULL)

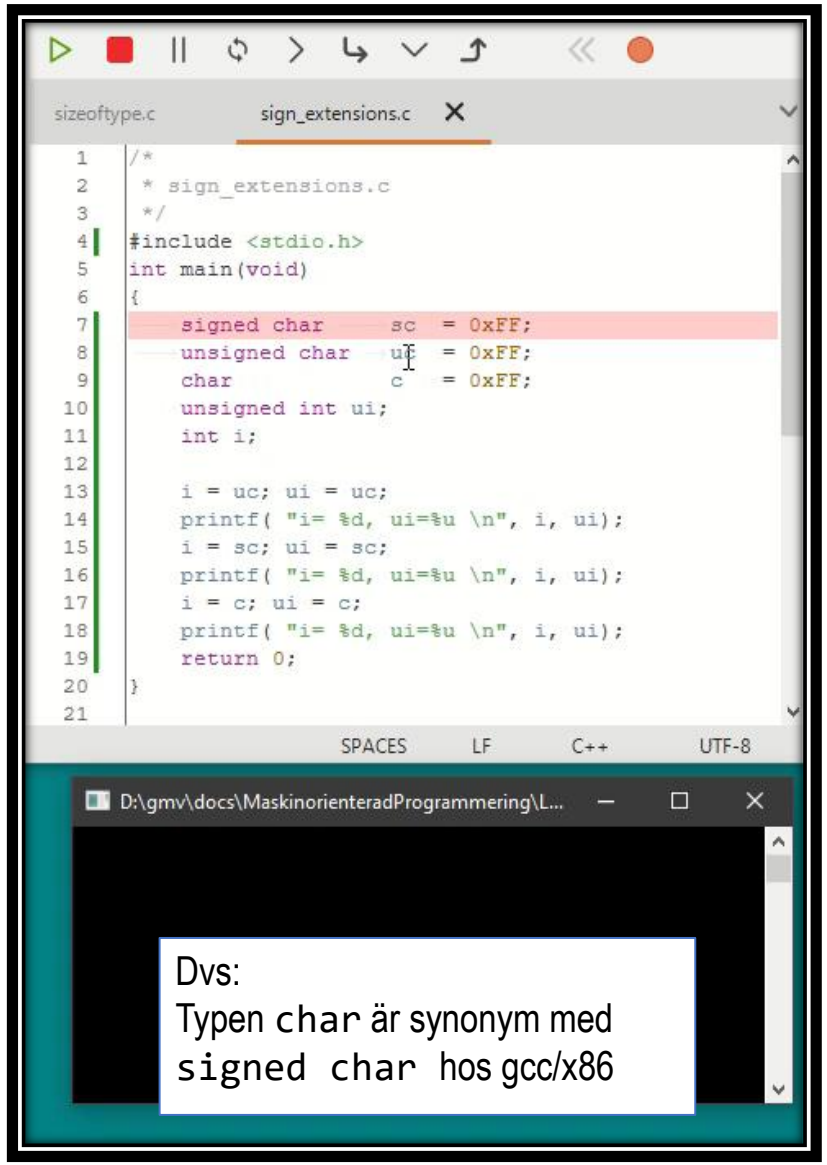
`printf` är en C-function i standardbiblioteket, för formatterade utskrifter:

Formattering för konvertering:
%d decimal
%u unsigned decimal
%o octal
%h hex
%e exponential
%f float
%c char
%s string

`printf` (formatsträng, argument...)

Exempel:

```
printf("i= %d, ui=%u \n", i, ui );
```



ASCII-tabellen

Övriga 128 koder är språkberoende (ISO)

Dec.	Bin.	Okt.	Hex	Tecken	Dec.	Bin	Okt.	Hex	Tecken	Dec.	Bin	Okt.	Hex	Tecken	Dec.	Bin.	Okt.	Hex	Tecken
0	00000000	0	00	NULL	32	00100000	40	20	SPACE	64	01000000	100	40	@	96	01100000	140	60	`
1	00000001	1	01	START OF HEADING	33	00100001	41	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	2	02	START OF TEXT	34	00100010	42	22	"	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	3	03	END OF TEXT	35	00100011	43	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	4	04	END OF TRANSMISSION	36	00100100	44	24	$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	5	05	ENQUIRY	37	00100101	45	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	6	06	ACKNOWLEDGE	38	00100110	46	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	7	07	BELL	39	00100111	47	27	'	71	01000111	107	47	G	103	01100111	147	67	g
8	00001000	10	08	BACKSPACE	40	00101000	50	28	(	72	01001000	110	48	H	104	01101000	150	68	h
9	00001001	11	09	HORIZONTAL TAB	41	00101001	51	29	)	73	01001001	111	49	I	105	01101001	151	69	i
10	00001010	12	0A	LINE FEED	42	00101010	52	2A	*	74	01001010	112	4A	J	106	01101010	152	6A	j
11	00001011	13	0B	VERTICAL TAB	43	00101011	53	2B	+	75	01001011	113	4B	K	107	01101011	153	6B	k
12	00001100	14	0C	FORM FEED	44	00101100	54	2C	,	76	01001100	114	4C	L	108	01101100	154	6C	l
13	00001101	15	0D	CARRIAGE RETURN	45	00101101	55	2D	-	77	01001101	115	4D	M	109	01101101	155	6D	m
14	00001110	16	0E	SHIFT OUT	46	00101110	56	2E	.	78	01001110	116	4E	N	110	01101110	156	6E	n
15	00001111	17	0F	SHIFT IN	47	00101111	57	2F	/	79	01001111	117	4F	O	111	01101111	157	6F	o
16	00010000	20	10	DATA LINK ESCAPE	48	00110000	60	30	0	80	01010000	120	50	P	112	01110000	160	70	p
17	00010001	21	11	DEVICE CONTROL 1	49	00110001	61	31	1	81	01010001	121	51	Q	113	01110001	161	71	q
18	00010010	22	12	DEVICE CONTROL 2	50	00110010	62	32	2	82	01010010	122	52	R	114	01110010	162	72	r
19	00010011	23	13	DEVICE CONTROL 3	51	00110011	63	33	3	83	01010011	123	53	S	115	01110011	163	73	s
20	00010100	24	14	DEVICE CONTROL 4	52	00110100	64	34	4	84	01010100	124	54	T	116	01110100	164	74	t
21	00010101	25	15	NEGATIVE ACKNOWLEDGE	53	00110101	65	35	5	85	01010101	125	55	U	117	01110101	165	75	u
22	00010110	26	16	SYNCHRONOUS IDLE	54	00110110	66	36	6	86	01010110	126	56	V	118	01110110	166	76	v
23	00010111	27	17	END OF TRANSMISSION BLOCK	55	00110111	67	37	7	87	01010111	127	57	W	119	01110111	167	77	w
24	00011000	30	18	CANCEL	56	00111000	70	38	8	88	01011000	130	58	X	120	01111000	170	78	x
25	00011001	31	19	END OF MEDIUM	57	00111001	71	39	9	89	01011001	131	59	Y	121	01111001	171	79	y
26	00011010	32	1A	SUBSTITUTE	58	00111010	72	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A	z
27	00011011	33	1B	ESCAPE	59	00111011	33	3B	;	91	01011011	133	5B	[	123	01111011	173	7B	{
28	00011100	34	1C	FILE SEPARATOR	60	00111100	74	3C	<	92	01011100	134	5C	\	124	01111100	174	7C	|
29	00011101	35	1D	GROUP SEPARATOR	61	00111101	75	3D	=	93	01011101	135	5D	]	125	01111101	175	7D	}
30	00011110	36	1E	RECORD SEPARATOR	62	00111110	76	3E	>	94	01011110	136	5E	^	126	01111110	176	7E	~
31	00011111	37	1F	UNIT SEPARATOR	63	00111111	77	3F	?	95	01011111	137	5F		127	01111111	177	7F	DEL

Teckenkonstanter skrivs:
`'Tecken'`

Prefix för numeriska konstanter:
Decimalt (inget)
Oktalt: `0` *(Noll)*
Hexadecimalt: `0x` *(Noll x)*

Exempel: 35 = 043=0x23= `'#'` 77 = 0115=0x4D= `'M'` 109 = 0155=0x6D= `'m'`

Specialtecken som Escape-sekvens

Dec.	Bin.	Okt.	Hex	Tecken	Escape	
7	00000111	7	07	*BELL*	`\a`	
8	00001000	10	08	*BACKSPACE*	`\b`	
9	00001001	11	09	*HORIZONTAL TAB*	`\t`	
10	00001010	12	0A	*LINE FEED*	`\n`	
11	00001011	13	0B	*VERTICAL TAB*	`\v`	
12	00001100	14	0C	*FORM FEED*	`\f`	
13	00001101	15	0D	*CARRIAGE RETURN*	`\r`	
27	00011011	33	1B	*ESCAPE*	`\e`	vanlig, men inte standard
34	00100010	42	22	*DOUBLE CITATION MARK*	`\"`	
39	00100111	47	27	*SINGLE CITATION MARK*	`\'`	
63	00111111	77	3F	*DOUBLE QUOTATION MARK*	`\?`	
				BYTE	`\nnn`	numeriskt värde (nnn), oktal form
				BYTE	`\xhh`	numeriskt värde (hh), hexadecimal form
				UNSIGNED 16 BIT	`\uhhhh`	numeriskt värde (hhhh), hexadecimal form
				UNSIGNED 32 BIT	`\Uhhhhhhhh`	numeriskt värde (hhhhxxxx), hexadecimal form

Escape-sekvenser används för att skriva tecken med speciell betydelse

Exempel:

```
printf("Text med \"citationstecken\" ");
```

ger utskriften

```
Text med "citationstecken"
```

Funktioner från standard C-biblioteket

Exempel: `isupper()`, undersök om ASCII-tecken är alfabetisk versal:

```
int isupper (int c)
{
   if ( c >= 'A' && c <= 'Z')
      return 1;
   return 0;
}
```

Dec.	Bin	Okt.	Hex	Tecken	Dec.	Bin	Okt.	Hex	Tecken	Dec.	Bin	Okt.	Hex	Tecken	Dec.	Bin	Okt.	Hex	Tecken
0	00000000	0	00	NULL	32	00100000	40	20	SPACE	64	01000000	100	40	@	96	01100000	140	60	`
1	00000001	1	01	START OF HEADING	33	00100001	41	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	2	02	START OF TEXT	34	00100010	42	22	"	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	3	03	END OF TEXT	35	00100011	43	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	4	04	END OF TRANSMISSION	36	00100100	44	24	$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	5	05	ENQUIRY	37	00100101	45	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	6	06	ACKNOWLEDGE	38	00100110	46	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	7	07	BELL	39	00100111	47	27	'	71	01000111	107	47	G	103	01100111	147	67	g
8	00001000	10	08	BACKSPACE	40	00101000	50	28	(	72	01001000	110	48	H	104	01101000	150	68	h
9	00001001	11	09	HORIZONTAL TAB	41	00101001	51	29	)	73	01001001	111	49	I	105	01101001	151	69	i
10	00001010	12	0A	LINE FEED	42	00101010	52	2A	*	74	01001010	112	4A	J	106	01101010	152	6A	j
11	00001011	13	0B	VERTICAL TAB	43	00101011	53	2B	+	75	01001011	113	4B	K	107	01101011	153	6B	k
12	00001100	14	0C	FORM FEED	44	00101100	54	2C	,	76	01001100	114	4C	L	108	01101100	154	6C	l
13	00001101	15	0D	CARRIAGE RETURN	45	00101101	55	2D	-	77	01001101	115	4D	M	109	01101101	155	6D	m
14	00001110	16	0E	SHIFT OUT	46	00101110	56	2E	.	78	01001110	116	4E	N	110	01101110	156	6E	n
15	00001111	17	0F	SHIFT IN	47	00101111	57	2F	/	79	01001111	117	4F	O	111	01101111	157	6F	o
16	00010000	20	10	DATA LINK ESCAPE	48	00110000	60	30	0	80	01010000	120	50	P	112	01110000	160	70	p
17	00010001	21	11	DEVICE CONTROL 1	49	00110001	61	31	1	81	01010001	121	51	Q	113	01110001	161	71	q
18	00010010	22	12	DEVICE CONTROL 2	50	00110010	62	32	2	82	01010010	122	52	R	114	01110010	162	72	r
19	00010011	23	13	DEVICE CONTROL 3	51	00110011	63	33	3	83	01010011	123	53	S	115	01110011	163	73	s
20	00010100	24	14	DEVICE CONTROL 4	52	00110100	64	34	4	84	01010100	124	54	T	116	01110100	164	74	t
21	00010101	25	15	NEGATIVE ACKNOWLEDGE	53	00110101	65	35	5	85	01010101	125	55	U	117	01110101	165	75	u
22	00010110	26	16	SYNCHRONOUS IDLE	54	00110110	66	36	6	86	01010110	126	56	V	118	01110110	166	76	v
23	00010111	27	17	END OF TRANSMISSION BLOCK	55	00110111	67	37	7	87	01010111	127	57	W	119	01110111	167	77	w
24	00011000	30	18	CANCEL	56	00111000	70	38	8	88	01011000	130	58	X	120	01111000	170	78	x
25	00011001	31	19	END OF MEDIUM	57	00111001	71	39	9	89	01011001	131	59	Y	121	01111001	171	79	y
26	00011010	32	1A	SUBSTITUTE	58	00111010	72	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A	z
27	00011011	33	1B	ESCAPE	59	00111011	33	3B	;	91	01011011	133	5B	[	123	01111011	173	7B	{
28	00011100	34	1C	FILE SEPARATOR	60	00111100	74	3C	<	92	01011100	134	5C	\	124	01111100	174	7C	|
29	00011101	35	1D	GROUP SEPARATOR	61	00111101	75	3D	=	93	01011101	135	5D	]	125	01111101	175	7D	}
30	00011110	36	1E	RECORD SEPARATOR	62	00111110	76	3E	>	94	01011110	136	5E	^	126	01111110	176	7E	~
31	00011111	37	1F	UNIT SEPARATOR	63	00111111	77	3F	?	95	01011111	137	5F		127	01111111	177	7F	DEL

Åtkomlighet och synlighet (*scope*, *visibility*)

Global synlighet
"global scope"

```
#include <stdio.h>

char x;

int foo()
{
   // x is visible
   // y is not visible
}
char y;
```

Synlig i denna källtext
"file scope"

```
#include <stdio.h>

static char x;

int foo(float x)
{
   /* argument x (float)
      is visible */
}
```

Synlig i denna funktion
"function scope"

```
#include <stdio.h>

char x;

int foo(float x)
{
   int x;
   /* local x (int)
      is visible */
}
```

Senare deklaration, parameter eller lokal, "döljer" tidigare...

Användning av funktioner

En funktion kan refereras före deklarationen...

```
int main(void)
{
   int x = foo();
}

int foo(void)
{
}
```

```
In function 'main':
9:11: warning: implicit declaration of function 'foo'
```

Varningen kan undvikas:

```
int foo(void)
{
}

int main(void)
{
   int x = foo();
}
```

```
int foo(void);
int main(void)
{
   int x = foo();
}

int foo(void)
{
}
```

Deklaration ("prototyp")

Definition

Vad är x?

```c
#include <stdio.h>

int x;

int foo(int x) {
   if( x == 0 ){
      int x = 4;
      return x;
   }
   return x;
}

int main() {
   x = 1;
   x = foo(0);
   printf("x is %d\n", x);
   return 0;
}
```