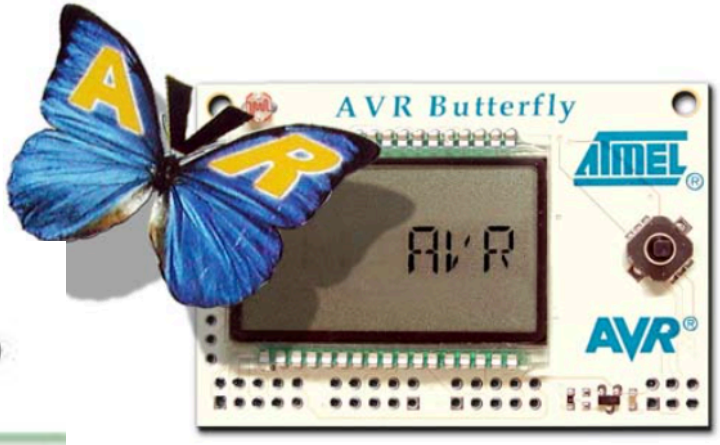
C Programming for Microcontrollers

Featuring ATMEL's AVR Butterfly and the Free WinAVR Compiler



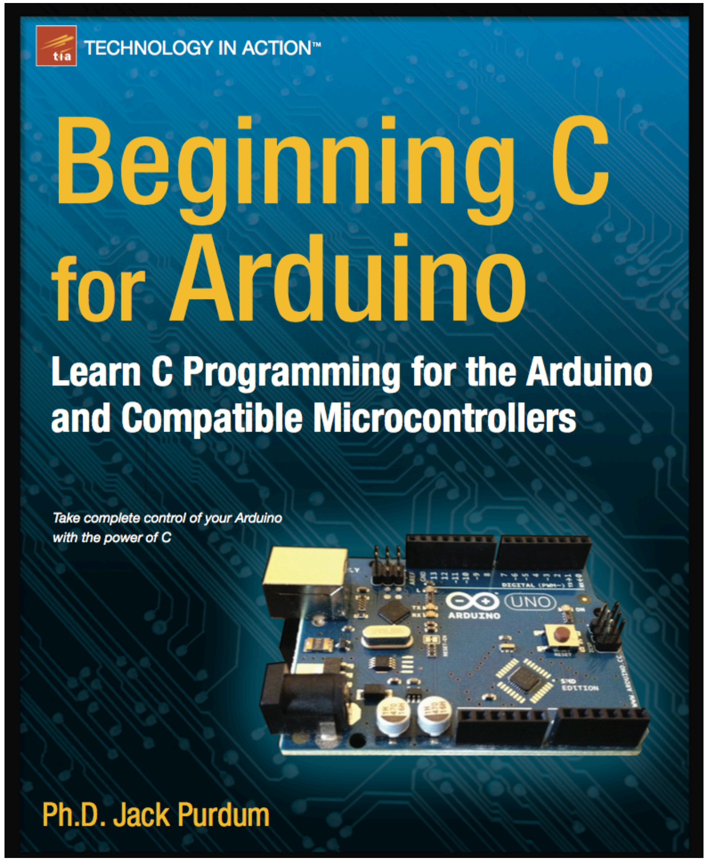
Joe Pardue

SmileyMicros.com

THE AVR MICROCONTROLLER AND EMBEDDED SYSTEMS USING ASSEMBLY AND C

SECOND EDITION: BASED ON ATMEGA328 AND ARDUINO BOARDS

MUHAMMAD ALI MAZIDI, SEPEHR NAIMI, AND SARMAD NAIMI



TECHNOLOGY IN ACTION™

Beginning C for Arduino

Learn C Programming for the Arduino and Compatible Microcontrollers

Take complete control of your Arduino with the power of C

Ph.D. Jack Purdum

IS6.1

High level language vs. Assembler

- Assembler => Assembling
  - Preprocessing
  - Lexical analysis
  - Code generation
  - Loading
  - Execution

- High level language => Compiling
  - Preprocessing
  - Lexical analysis
  - Syntactic analysis
  - Semantic analysis
  - Code generation
  - Linking
  - Loading
  - Execution

Programming in C

- Coding standard!
  - Why?
  - Different standards (e.g. GNU, Indian Hill)
  - *The International Obfuscated C Code Contest*
    - *http://www.ioccc.org*
    - *Example:*

    -
      ```
      main(int riguing,char**acters){puts(1[acters-~!(*(int*)1[acters]%4796%275%riguing)]);}
      ```

Differences between Java and C

- Java is derived from *C and C*++
- Many of its syntactic characteristics are similar to C
  - – see later slides

- However, there are some huge differences
  - – Let’s look at a few

“Conceptual” differences between Java and C

Java	C
Object Oriented	Procedural
Memory management (garbage collection)	Manual memory management
Object references	Pointers
Exceptions	Error codes

Example program

```
#include <stdio.h>      //Similar to ’import’ in Java
#include ”mydefs.h”

double g_val;           //Global variable
/* This is a comment, as in Java */
int main(void)          //Returns error code…
{
     int loc = 0;       //Local variable – before st!
     g_val = 0.42;
     printf("loc = %d g_val = %f\n", loc, g_val);
     return 0;
}
```

Expressions

- Arithmetic operators are the same:–

  `+, -, *, /, %, ++, --`

- Numerical type conversion is mostly the same
  - Java spells out divide by zero, **`NaN`** (not a number, etc.)
  - C is machine dependent

Data Types in AVR systems

- Use unsigned whenever you can
- unsigned char instead of unsigned int if you can

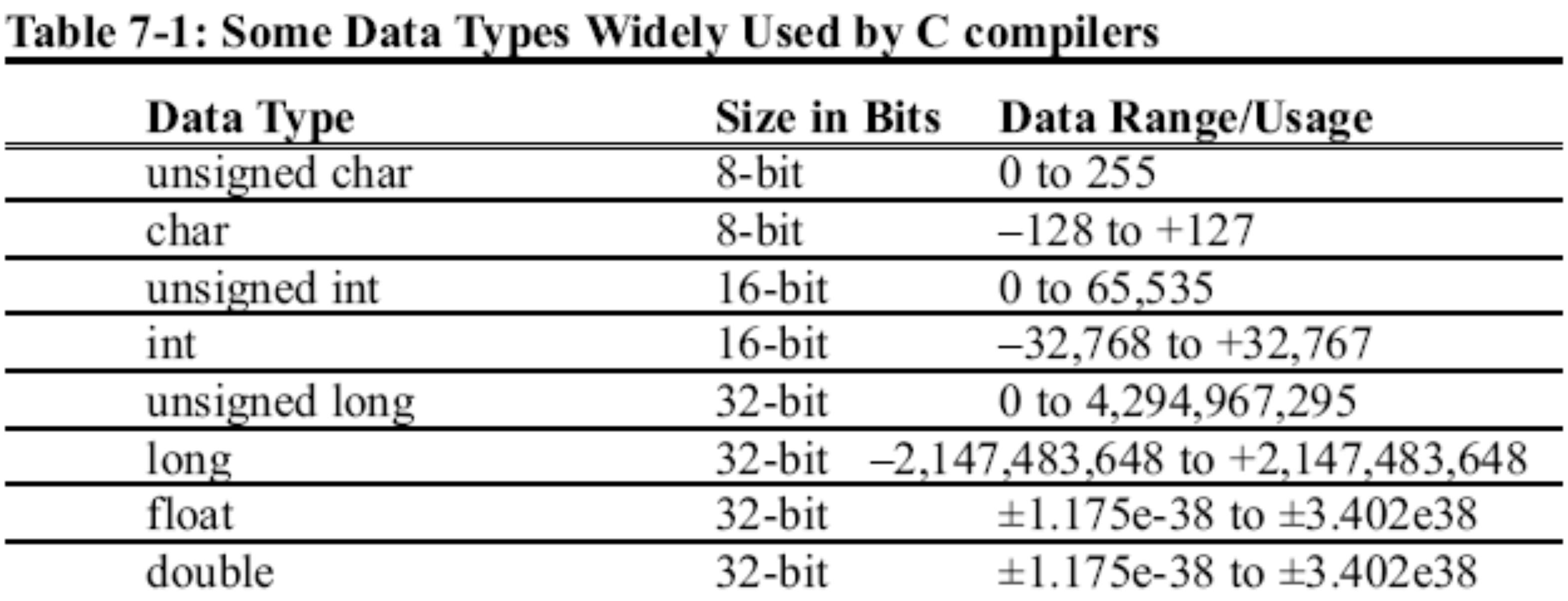


Table 7-1: Some Data Types Widely Used by C compilers

Data Type	Size in Bits	Data Range/Usage
unsigned char	8-bit	0 to 255
char	8-bit	–128 to +127
unsigned int	16-bit	0 to 65,535
int	16-bit	–32,768 to +32,767
unsigned long	32-bit	0 to 4,294,967,295
long	32-bit	–2,147,483,648 to +2,147,483,648
float	32-bit	±1.175e-38 to ±3.402e38
double	32-bit	±1.175e-38 to ±3.402e38

- Also `int8_t, uint8_t`, etc.
- See: http://en.cppreference.com/w/c/types/integer

Relational Operators

- Relational operators work the same way but return different result types:–

```
>, >=, <, <=, ==, !=
```

- In Java, they return values `FALSE` and `TRUE`
- In C, they return values `0` and `1`
- In C,
  - a value of `zero` means *false*
  - any value that is not zero means *true*
  - E.g., 1, 5, -1000000, 3.14159, 6.626068 × 10^{-34}

Very important!

Relational Operators

```
void example(void) {
      int y;
      int x = 100;
      if (x == 4) { }   /* false, since X is 100… */
      if (x) { }        /* true, since x <> 0 */
      while (x) { x--; } /* repeats 100 times until x is 0

      /* MISTAKES */
      if (x = 1) { }    /* We forgot an ’=‘ here! Now 1 is
                           assigned to X, which is OK. */
      while (y) { y--; } /* Y is not initialized, - OK */
}
```

- Note! Any type of variable can be used in if, while, etc.

Global/Local variables

- Variables declared outside functions are “Global” – visible and accessible to all!
  - This can be limited a bit (keywords `Extern`, `const`, `static`)
- Variables declared in a function are “local” and (normally) only live while function is executed
  - can be “fixed” by declaring `static`
- Variables can be declared `register` or `volatile`

Conditional and Bitwise Operators

- Conditional execution operators are same in Java and C:–

  `||, &&, ?` followed by `:`

- Bitwise operators are same in Java and C:–

  `|, &, ^` for bit-by-bit operations with a word

- Shift operators differ a little bit

  << (left shift) is the same

  >> (right shift) is machine dependent in C

  - I.e., whether to fill from left with zeros or sign bits
  - Java: >> arithmetic shift (fills with sign bits),
    >>> logical shift (fills with 0's)

Assignment and Unary Operators

- Assignment operators work the same:–

```
=, +=, -=, *=, /=, &=, |=, ^=
```

- The following unary operators are available C but not in Java

`~`	invert the bits of a word
`*`	pointer dereference
`&`	pointer creation (address of…)
`(type)`	cast (i.e., forceable type conversion)
`sizeof`	# of bytes in operand or data type
`->`	pointer dereference with field selection

Summary about Expressions and Operators

- Pretty much the same in C and Java
- *Be sure to check details*
- *Be sure to check operator precedence*
  - Table 2-1 in kernighan and ritchie (reference book, section 2.12, p 48)

Bit manipulation in C

- & and
- | inclusive or
- ^ exclusive or
- << left shift
- >> right shift
- ~ one’ s complement

a:	1	0	1	0	1	0	1	0	0xaa
b:	0	0	0	0	1	1	1	1	0x0f
a&b:	0	0	0	0	1	0	1	0	0x0a
a|b:	1	0	1	0	1	1	1	1	0xaf
a^b:	1	0	1	0	0	1	0	1	0xa5
a<<1:	0	1	0	1	0	1	0	0	0x54
b>>2:	0	0	0	0	0	0	1	1	0x03
~b:	1	1	1	1	0	0	0	0	0xf0

Statements

- Statements in *C*:–
    - Labeled statement
    - Expression statement
    - Compound statement
    - Selection statement
    - Iteration statement
    - Jump statement

Statements

- Statements in *C*:–
  - Labeled statement
  - Expression statement
  - Compound statement
  - Selection statement
  - Iteration statement
  - Jump statement

E.g., cases of a **`switch`** statement

Similar to Java

Statements

- Statements in *C*:–
  - Labeled statement
  - Expression statement
  - Compound statement
  - Selection statement
  - Iteration statement
  - Jump statement

Any expression followed by `';'`

Much like to Java

Statements

- Statements in *C*:–
  - Labeled statement
  - Expression statement
  - Compound statement
  - Selection statement
  - Iteration statement
  - Jump statement

Sequence of statements enclosed in "**{ }**"

Called a “block” in Java

Statements

- Statements in *C*:–
  - Labeled statement
  - Expression statement
  - Compound statement
  - Selection statement
  - Iteration statement
  - Jump statement

```
switch (expr)
if (expr)statement
if (expr) statement
 else statement
```

Same as in Java

Statements

- Statements in *C*:–
  - Labeled statement
  - Expression statement
  - Compound statement
  - Selection statement
  - Iteration statement ←
  - Jump statement

```
while (expr) statement
do statement while (expr
for (exp1; exp2, exp3)
 statement
```

Very similar to Java

Statements

- Statements in *C*:–
    - Labeled statement
    - Expression statement
    - Compound statement
    - Selection statement
    - Iteration statement
    - Jump statement ←

brea
cont
retu
Very

goto
Not present in Java
Not allowed in this course

Summary about Statements

- Pretty much the same in C and Java
- *Be sure to check details in textbooks*

Formatted Input & Output

- Very different between C and *Java*
- Very different in C
- Handled by library functions in C
  - `printf()`
  - `scanf()`
  - `getc()`
  - `putc()`
  - Many others!
- Only printf important in this course (lab 5), since we do our own I/O!

Summary

- Differences and similarities between Java and C
    - Expressions
    - Statements
- There are *lots* of other differences
    - Some will be covered during the course
    - Others will be covered in future courses…

Header files – a help to structure programs

Header files contains definitions that we need in our program, e.g.:

example.h contains:

```
char *test (void);
```

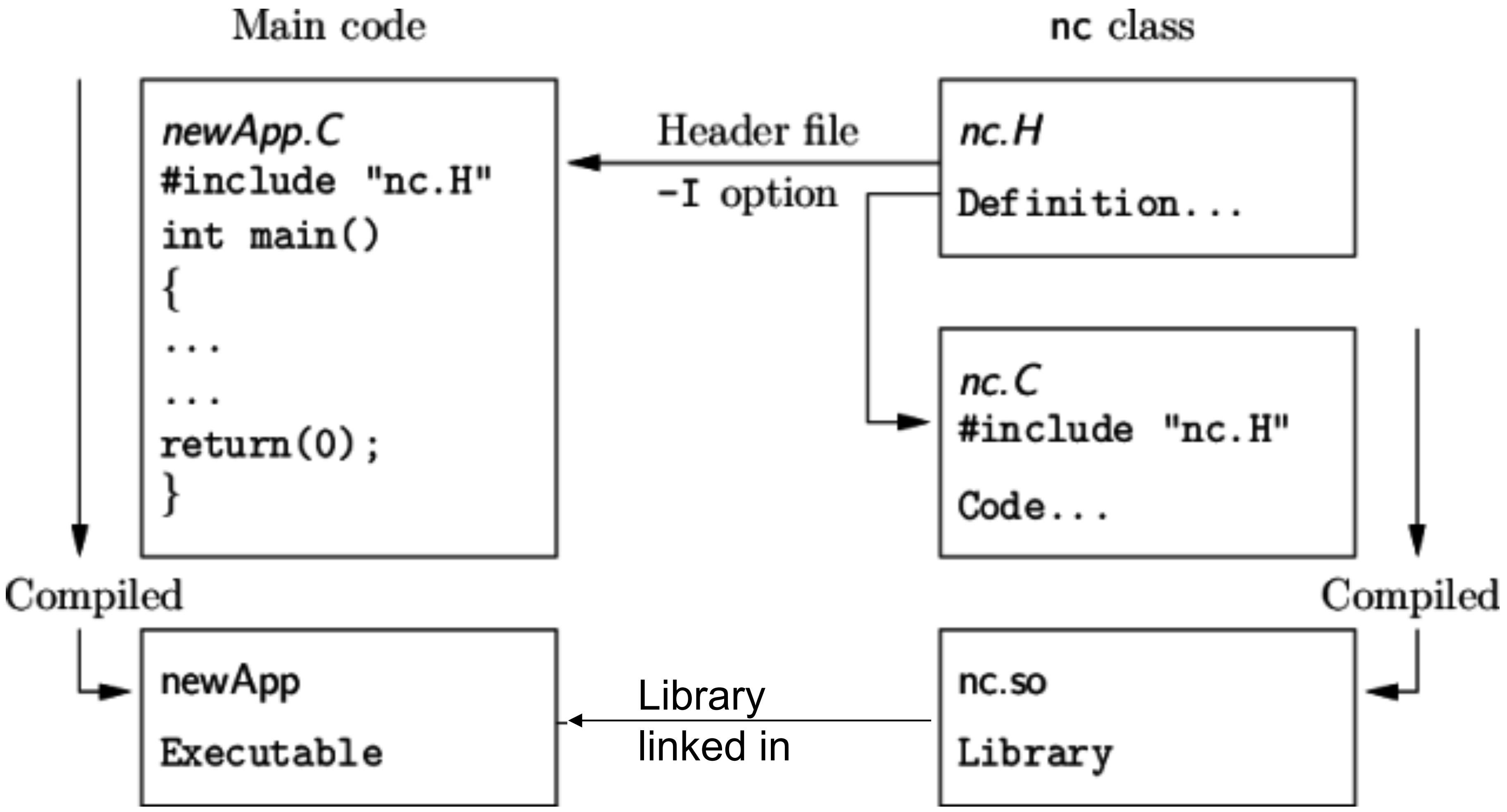
and a main program called *program.c* that uses the header file, like this:

```
int x;
#include "header.h”
int main (void)
{
   puts (test ());
}
```

the compiler will see the same token stream as it would if program.c read:

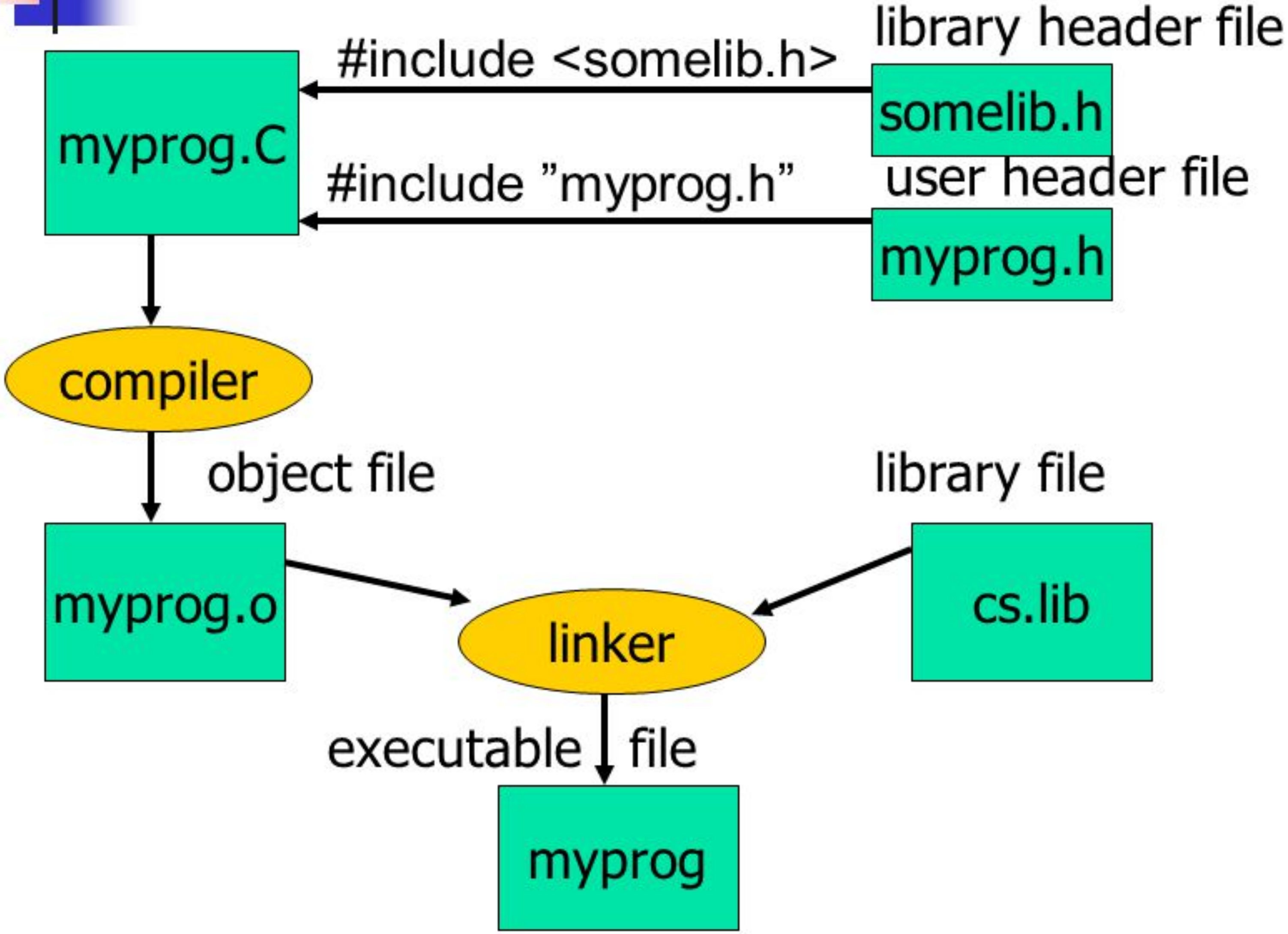
```
int x;
char *test (void);
int main (void)
{
   puts (test ());
}
```

Program with several files + libraries

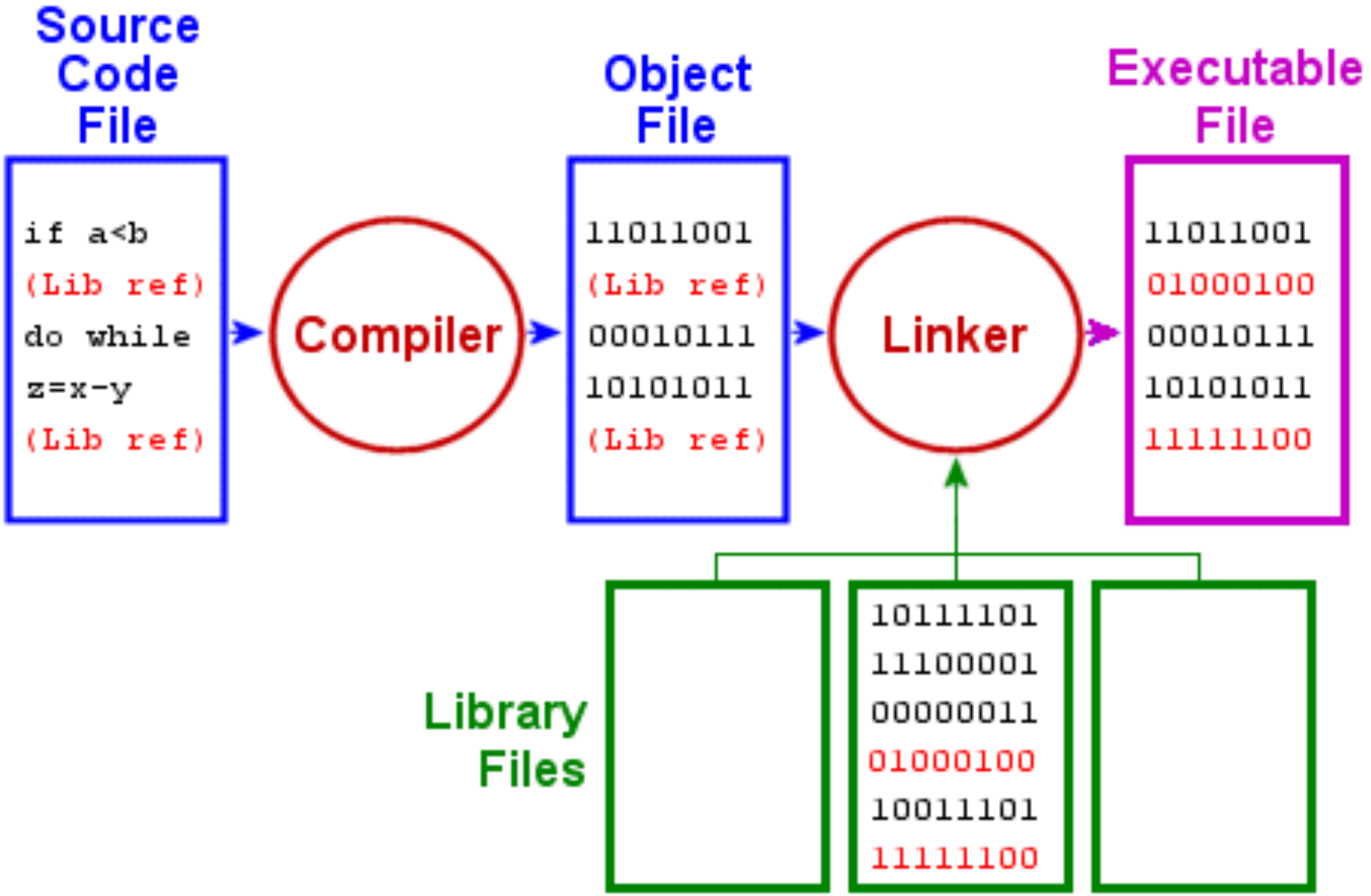


https://www.youtube.com/watch?v=tAgK0bEbmr0

Header and Library Files



The linker – what does it do?



Header files – assembly code

- **asmfunc_calledfrom_c** is a subroutine that will be implemented in assembly code.
- In order to call the subroutine from C, we need to create a function prototype so that C knows how to interface with the function.
- The *extern* keyword specifies that the function is defined in a different (external) file.

```
#include <avr/io.h>

extern void
       asmfunc_calledfrom_c(
       uint8_t val);

int main()
{
  asmfunc_calledfrom_c(3);
  return 0;
}
```

- When writing an assembly function that would be called by C we just use **R24** as the register containing the value passed to the function.

“Prototypes” – function definition

```
int max(int, int);                  /* prototype */

void example(int n) {
      int result = max(n, 8);
}
int max(int one, int two) {         /* function */
      if one > two return 1 else return example(2);
}
int main(void) {
      example(1);
}
```

- Functions need to be declared before they are used!
  - Tricky when mutual dependece exist…

AVR Programming in C

Chapter 7

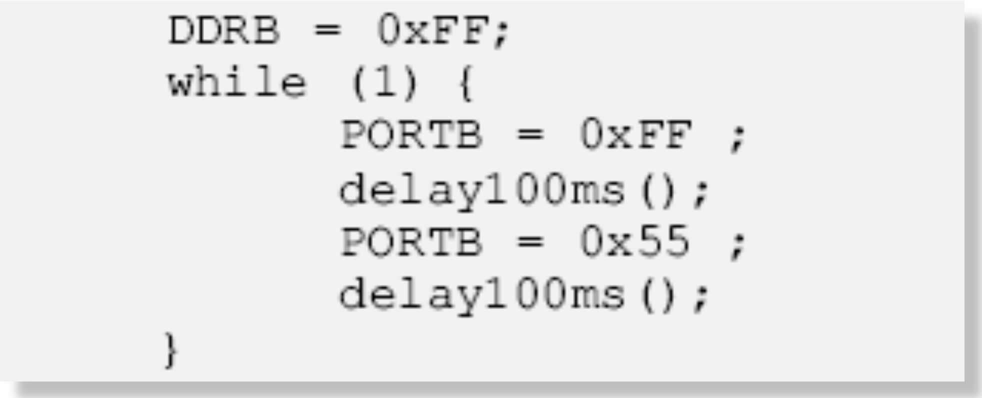
The AVR microcontroller
and embedded
systems
using assembly and c

Include files for AVR

- *#include <avr/io.h>*
  - Definitions for **PORTB**, **SREG**, etc

I/O programming in C

Byte size IO programming in C



```
DDRB = 0xFF;
while (1) {
        PORTB = 0xFF ;
        delay100ms();
        PORTB = 0x55 ;
        delay100ms();
}
```

Different compilers have different syntax for bit manipulations!

Masking is the best way

Logical Operations in C

- True AND True = True
- True OR False = True
- Not (True) = False

1110 1111 & 0000 0001 = ?

1110 1111 | 0001 0000 = ?

!(1110 1111) = ?

Answers on next slide…

Bit-Wise logical operators

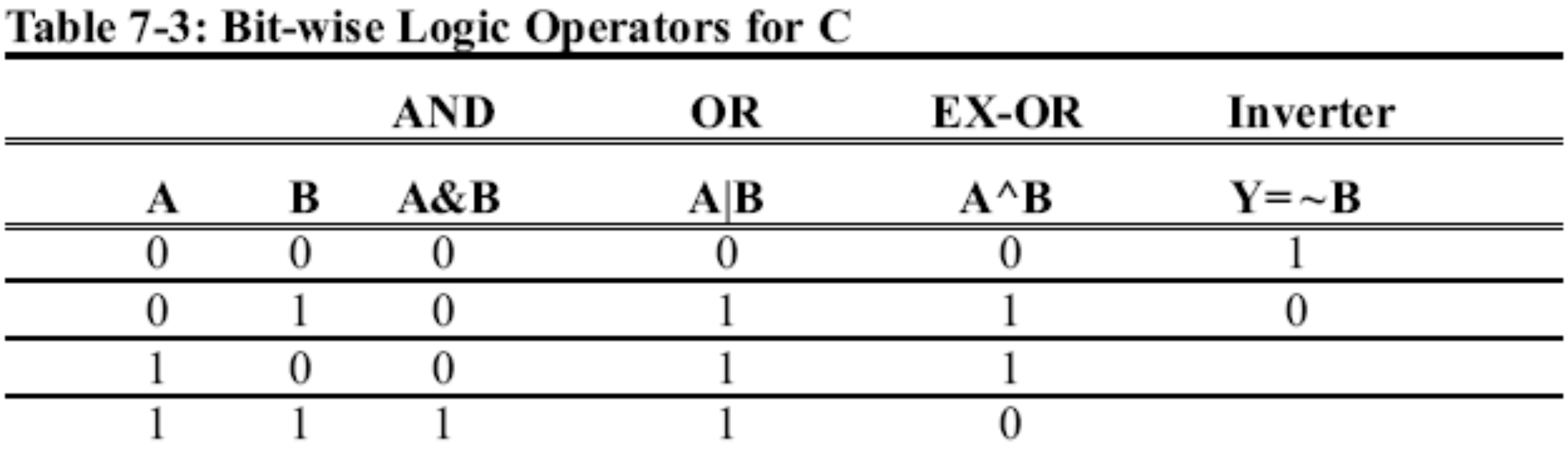


Table 7-3: Bit-wise Logic Operators for C

		AND	OR	EX-OR	Inverter
A	B	A&B	A|B	A^B	Y=~B
0	0	0	0	0	1
0	1	0	1	1	0
1	0	0	1	1	
1	1	1	1	0	

```
  1110 1111       1110 1111
& 0000 0001     | 0001 0000     ~ 1110 1011
 ------------    ------------     ------------
  0000 0001       1111 1111       0001 0100
```

Shift operations in C

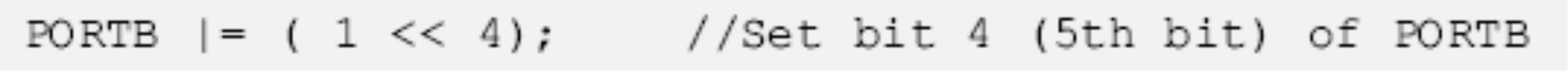
- data >> number of bits to be shifted right
- data << number of bits to be shifted left

```
 1110 0000 >> 3          0000 0001 <<2
---------------         ---------------
 0001 1100               0000 0100
```

Setting a bit in a Byte to 1

- We can use | operator to set a bit of a byte to 1

```
     xxxx xxxx                        xxxx xxxx
|    0001 0000          OR       |    1 << 4
     -------------                    -------------
     xxx1 xxxx                        xxx1 xxxx
```

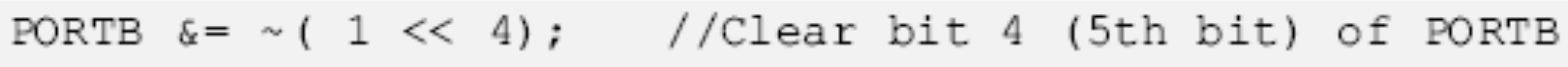


```
PORTB |= ( 1 << 4);      //Set bit 4 (5th bit) of PORTB
```

Clearing a bit in a Byte to 0

- We can use & operator to set a bit of a byte to 0

```
     xxxx xxxx                 xxxx xxxx
&    1110 1111      OR      &  ~(1 << 4)
     -------------             -------------
     xxx0 xxxx                 xxx0 xxxx
```



```
PORTB &= ~( 1 << 4);    //Clear bit 4 (5th bit) of PORTB
```

See Example 7-18 in book...

Making the code easier to read…

- Do you think “`PORTB &= ~(1 << 4)`” is easy to read?
- Maybe rewrite to:

  “`PORTB = PORTB & ~(1 << 4)`” or

  “`PORTB = PORTB & 0b11101111`”
- C allows MACROs to be defined:

```
#define clearbit(PORT, N)
            ((PORT) &= ~(1 << (N)))
```

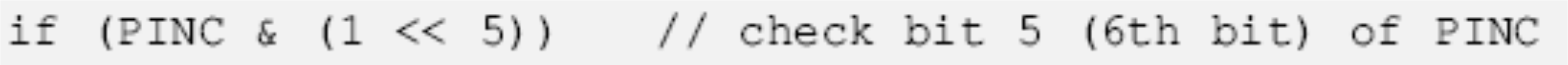
- Now you can write this instead:

```
clearbit(PORTB, 4)
```

Checking a bit in a Byte

- We can use & operator to see if a bit in a byte is 1 or 0

```
       xxxx xxxx                         xxxx xxxx
&      0010 0000          OR        &    (1 << 5)
       -------------                     -------------
       00x0 0000                         00x0 0000
```



```
if (PINC & (1 << 5))   // check bit 5 (6th bit) of PINC
```

Example: A door sensor is connected to the Port B, pin 1, and a LED is connected to Port C, pin 7. Write C code to monitor the door sensor and, when it opens, turn on the LED.

```
#inlcude <AVR/io.h>
#define LED         7
#define SENSOR      1

Int main(void) {
   DDRB &= ~(1<< SENSOR);           // SENSOR pin input
   DDRC |= (1<<LED);                // LED pin is output
   while (1) {
       if (PINB & (1<<SENSOR)       // check sensor
          PORTC |= (1<<LED);        // LED on
       else
          PORTC &= ~(1>>LED);       // LED off
   }
   return 0;
}
```

Potential problem in FOR statement

```
Java:
    for(int i=1; i<11; i++){
               <variable i is only available in here>
    }
C:
   for (i = 1; i < 11; i++) {
      <variable i is defined outside....>
   }
```

- The counter variable (i here) should be declared as other variables
- Atmel studio accepts to declare i in for statement!!! (hides i declared in surrounding function)

DA346A

- Example code…

Examples from keyboard code (1/2)…

```
static const char key_map[16] = {'1', '4', '7', '*',
                                  '2', '5', '8', '0',
                                  '3', '6', '9', '#',
                                  'A', 'B', 'C', 'D'};
…

char numkey_read(void)
{
        SET_BIT(PORTG, 5);                  // Set column 0
        CLR_BIT(PORTE, 3);                  // Clear column 1
        CLR_BIT(PORTH, 3);                  // Clear column 2
        CLR_BIT(PORTH, 4);                  // Clear column 3
```

- Why is only one column set (and the others cleared)?

Examples from keyboard code (2/2)…

```
       delay_ms(1);
       if (PINF & 0x20) {                 // Row 0?
              return key_map[0];
       } else if (PINF & 0x10) {          // Row 1?
              return key_map[1];
       } else if (PINE & 0x10) {          // Row 2?
              return key_map[2];
       } else if (PINE & 0x20) {          // Row 3?
              return key_map[3];
       }
…
```

- What is tested in the IF-statements?
- What is retrieved from the key_map?

LCD functions (From ”lcd.h”)

```
enum lcd_register {
     CMD, // to send a Command to the LCD
     CHR  // to send a character to the LCD
};
void lcd_init(void);
void lcd_write(enum lcd_register, uint8_t);
void lcd_write_str(char *);
void lcd_clear(void);
void lcd_set_cursor_pos(uint8_t, uint8_t);
```

LCD Instructions/Commands (C)

- The following instructions are provided (can be adjusted):

```
   lcd_write(CMD, 0x21);      //Function set (H=1) ext instr. set
   lcd_write(CMD, 0x13);      //Set bias mode 1:48
   lcd_write(CMD, 0xC6);      //Set Vop (contrast)
   lcd_write(CMD, 0x06);      //Set temp coefficient

   lcd_write(CMD, 0x20);      //Function set (H=0) normal instr set
   lcd_write(CMD, 0x0C);      //Set display control (normal mode)
Other commands (normal mode):
    lcd_write(CMD, 0x09);     //All segments ON
    lcd_write(CMD, 0x0D):     //inverse video
To show text(normal mode):
    lcd_write(CHR, ’A’);      //example…
```

There is NO command for ”Clear screen”!