7.3.1 Structure of a C program

- A C program typically has two main sections.
  - #include section: to insert header files.
  - main() section: code that runs when the program starts.
- In the example below, <avr/io.h> is a header file that contains all register definitions for the AVR microcontroller.

```
#include <avr/io.h> // avr header file for all registers/pins
int main(void){
    unsigned char i; // temporary variable
    DDRA = 0x00;     // set PORTA for input
    DDRB = 0xFF;     // set PORTB for output
   PORTB = 0x00;    // turn ON all LEDs initially
    while(1){
          // Read input from PORTA, which is connected to the 8 switches
          i = PINA;
          // Send output to PORTB, which is connected to the 8 LEDs
          PORTB = i;
    }
    return 1;
}
```

C comments

- Comments are text that the compiler ignores.
- For a single-line comment, use double back slashes

```
DDRA = 0x00;     // set PORTA for input
```

- For a multi-line comment, use the pair /* and */

```
/* File: led.c
   Description: Simple C program for the ATMEL AVR(ATmega16 chip)
   It lets user turn on LEDs by pressing the switches on the STK500
   board
*/
```

- Always use comments to make program easy to understand.

C statements and blocks

- C statements
  - C statements control the program flow.
  - They consist of keywords, expressions and other statements.
  - A statement ends with semicolon.

```
DDRB = 0xFF;     // set PORTB for output
```

- C blocks
  - A C block is a group of statements enclosed by braces {}.
  - Usually, a C block is run depending on some logical conditions.

```
while (1){
  // Read input from PORTA - connected to the 8 switches
  i = PINA;
  // Send output to PORTB - connected to the 8 LEDs
  PORTB = i;
}
```

7.3.2 Data types and operators

- The main data types in C are
  - char: 8-bit integer
  - int: 16-bit integer
  - long int: 32-bit integer
- The above data types can be modified by keyword 'unsigned'

```
char a;               // a value range -128, 0, …, 127
unsigned char b;      // b value range 0, 1, 2, …, 255
unsigned long int c; // c value range 0,…, 2^32 - 1
```

- Some examples of variable assignment

```
a = 0xA0;          // a stores hexadecimal value of A0
b = '1';           // b stores ASCII code of character '1'
c = 2000ul;        // c stores a unsigned long integer 2000
```

C operators

- C has a rich set of operators
  - Arithmetic operators
  - Relational operators
  - Logical operators
  - Bit-wise operators
  - Data access operators
  - Miscellaneous operators

Arithmetic operators

Operator	Name	Example	Description
*	Multiplication	x * y	Multiply x times y
/	Division	x / y	Divide x by y
%	Modulo	x % y	Remainder of x divided by y
+	Addition	x + y	Add x and y
-	Subtraction	x - y	Subtract y from x
++	Increment	x++ ++x	Increment x by 1 after using it Increment x by 1 before using it
--	Decrement	x-- --x	Decrement x by 1 after using it Decrement x by 1 before using it
-	Negation	-x	Negate x

Relational operators

Operator	Name	Example	Description
>	Greater than	x > 5	1 if x is greater than 5, 0 otherwise
>=	Greater than or equal to	x >=5	1 is x is greater than or equal to 5, 0 otherwise
<	Less than	x < y	1 if x is smaller than y, 0 otherwise
<=	Less than or equal to	x <= y	1 is x is smaller than or equal to y, 0 otherwise
==	Equal to	x == y	1 is x is equal to y, 0 otherwise
!=	Not equal to	x != 4	1 is x is not equal to 4, 0 otherwise

Logical operators

- These operate on logical variables/constants.

Operator	Name	Example	Description
!	Logical NOT	!x	1 if x is 0, otherwise 0
&&	Logical AND	x && y	1 is both x and y are 1, otherwise 0
||	Logical OR	x || y	0 if both x and y are 0, otherwise 1

Bit-wise operators

- These operate on individual bits of a variable/constant.

Operator	Name	Example	Description
~	Bit-wise complement	`~x`	Toggle every bit from 0 to 1, or 1 to 0
&	Bitwise AND	`x & y`	Bitwise AND of x and y
|	Bitwise OR	`x | y`	Bitwise OR of x and y
^	Bitwise XOR	`x ^ y`	Bitwise XOR of x and y
<<	Shift left	`x << 3`	Shift bits in x three positions to the left
>>	Shift right	`x >> 1`	Shift bits in x one position to the right

Data-access operators

- These operate on arrays, structures or pointers.
- We'll learn more about these operators later.

Operator	Name	Example	Description
[]	Array element	`x[2]`	Third element of array x
.	Member selection	`x.age`	Field 'age' of structure variable x
->	Member selection	`p->age`	Field 'age' of structure pointer p
*	Indirection	`*p`	Content of memory location pointed by p
&	Address of	`&x`	Address of the memory location where variable x is stored

Miscellaneous operators

Operator	Name	Example	Description
`()`	Function	`_delay_ms(250)`	Call a function to create delay of 250ms

`(type)`	Type cast	`char x = 3;`
		`(int) x`
			x is 8-bit integer
			x is converted to 16-bit integer

?	Conditional evaluation	`char x;`
		`y=(x>5)?10:20;`
			This is equivalent to
			`if (x > 5)`
			`    y = 10;`
			`else`
			`    y = 20;`

commonly used by C coders.

7.3.3 Flow control in C

- By default, C statements are executed sequentially.
- To change the program flow, there are six types of statements

  - ❑ if-else statement
  - ❑ switch statement

  } Conditional

  - ❑ while statement
  - ❑ for statement
  - ❑ do statement

  } Iterative

  - ❑ goto statement

  } Should be avoided!

If-else statement

- General syntax

```
if (expression)
   statement_1;
else
   statement_2;
```

- Example code

```
char a, b, sum;
a = 4; b = -5;
sum = a + b;
if (sum < 0)
   printf(“sum is negative”);
else if (sum > 0)
   printf(“sum is positive”);
else
   printf(“sum is zero”);
```

Switch statement

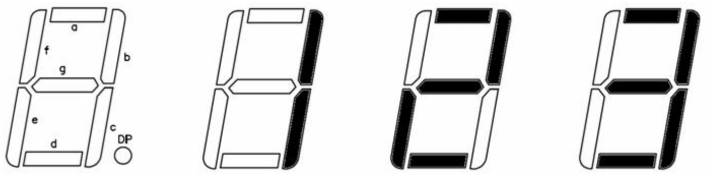
- General syntax

```
switch (expression)
case constant_1:
   statement_1;
   break;
case constant_2:
   statement_2;
   break;
…
case constant_n:
   statement_n;
   break;
default:
   statement_other;
}
```

Use ‘break’ to separate different cases.

Switch statement — Example

Lab 7: Find the bit pattern to display a digit on the 7-segment LED.



(a) 7 segments of the LED

Bit number:	7	6	5	4	3	2	1	0
Purpose:	DP	g	f	e	d	c	b	a

(b) Bit assignment on the LED plug

Figure C.2: 7-segment display

- Bit pattern for digit ‘1’: 0 0 0 0 0 1 1 0
- Bit pattern for digit ‘2’: 0 0 0 1 1 0 1 1

Switch statement — Example

```
unsigned char digit;
unsigned char led_pattern;
switch (digit)
case ‘0’:
   led_pattern = 0b00111111;
   break;
case ‘1’:
   led_pattern = 0b00000110;
   break;
case ‘2’:
   led_pattern = 0b01011011;
   break;
//you can complete more cases here...
default:
}
PORTB = led_pattern; // send to PORTB and 7-segment LED
```

While statement

- General syntax

```
while (expression){
   statements;
}
```

- Example code: Compute the sum of 1 + 2+ …+ 100

```
int sum, i;
i = 1; sum = 0;
while (i <= 100){
   sum = sum + i;
   i = i + 1;
}
```

For statement

- General syntax

```
for (expression1; expression2; expression3){
   statements;
}
```

  - ❑ expression1 is run before the loop starts.
  - ❑ expression2 is evaluated before each iteration.
  - ❑ expression3 is run after each iteration.

- Example code: Compute the sum of 1 + 2+ …+ 10

```
int sum;
sum = 0;

for (int i = 1; i <= 10; i++){
   sum = sum + i;
}
```

Do statement

- General syntax

```
do {
   statements;
} while (expression);
```

- Example code: compute the sum of 1 + 2 + … + 10

```
int sum, i;
i = 1; sum = 0;
do{
   sum = sum + i;
   i = i + 1;
} while (i <= 10);
```

Break statement in loop

- The 'break' statement inside a loop forces early termination of the loop.

- What is the value of 'sum' after the following code is executed?

```
int sum, i;
i = 1; sum = 0;
while (i <= 10){
   sum = sum + i;
   i = i + 1;
   if (i > 5)
       break;
}
```

sum = ?

Continue statement in loop

- The ‘continue’ statement skips the subsequent statements in the code block and forces the execution of the next iteration.

- What is the value of ‘sum’ after the following code is executed?

```
int sum, i;
i = 1; sum = 0;
while (i <= 10){
   i = i + 1;
   if (i < 5)
       continue;
   sum = sum + i;
}
```

sum = ?

C arrays

- An array is a list of values that have the same data type.
- In C, array index starts from 0.
- An array can be one-dimensional, two-dimensional or more.
- This code example creates a 2-D array (multiplication table):

```
int a[8][10];
for (int i = 0; i < 8; i++)
    for (int j = 0; i < 10; j++)
        a[i][j]= i * j;
```

- An array can be initialized when it is declared.

```
int b[3] = {4, 1, 10};
unsigned char keypad_key[3][4] = {{'1', '4', '7', '*'},
                                  {'2', '5', '8', '0'},
                                  {'3', '6', '9', '#'}};
```

7.3.4 C functions

- C functions are sub-routines that can be called from the main program or other functions.

- Functions enable modular designs, code reuse, and hiding of complex implementation details.

- A C function can have a list of parameters and produce a return value.

- Let us study C functions through examples.

Functions — Example 1

Write a function to compute the factorial n! for a given n.

```
// factorial is the name of the custom function
// it accepts an input n of type int, and return an output of type int
int factorial(int n){
    int prod = 1;
    for (int i = 1; i <=n; i++)
         prod = prod * i;
    return prod;       // return the result
}
```

```
int main(void){
     int n = 5;         // some example value of n
     int v;             // variable to storage result
     v = factorial(n); // call the function, store return value in v
     return 1;
}
```

Functions — Example 2

Write a function to compute the factorial n! for a given n.

```
// factorial is the name of the custom function
// it accepts an input n of type int,
// it stores output at memory location by int pointer p
void factorial(int n, int* p){
    int prod = 1;
    for (int i = 1; i <=n; i++)
         prod = prod * i;
    *p = prod;// store output at memory location pointed by p
}
```

```
int main(void){
     int n = 5;         // some example value of n
     int v;             // variable to storage result
     factorial(n, &v); // call the function, store return value in v
}
```

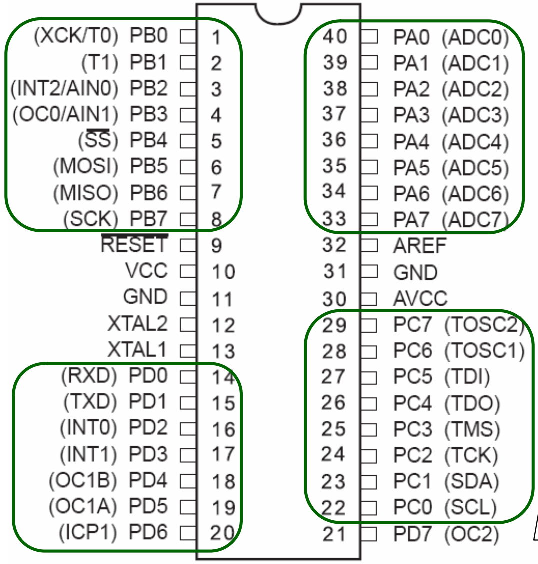
Guidelines on C coding and documentation

- **Optimize the C code for efficiency and length.**
- **Delete unnecessary lines of code.**
- **The C code must be properly formatted.**
- **For printing, use a fixed-width font such as `Courier New` for code.**
- **Use indentation to show the logical structure of the program.**
- **Use a blank line to separate code sections.**
- **Use meaningful variable names and function names.**
- **If a C statement is too long for one printed line, split it logically into multiple lines.**
- **Use C comments concisely to explain code.**
- **Observe the way that C code is presented in the lecture notes or lab notes.**

7.4 Digital IO in ATmega16

- **ATmega16 has fours 8-bit digital IO ports:**
  - **PORT A,**
  - **PORT B,**
  - **PORT C, and**
  - **PORT D.**
- **Each port has 8 data pins.**
- **Every port is bi-directional. Each of the 8 pins can be individually configured as**
  - **input (receiving data into microcontroller), or**
  - **output (sending data from microcontroller).**

Digital IO in ATmega16 — Pins



ATmega16 chip

Digital IO in ATmega16 — Configuring for input/output

- For each port, there are three relevant 8-bit registers.
  - ❑ Data Direction Register (DDRx)
  - ❑ Input Pins Address (PINx)
  - ❑ Data Register (PORTx)

  Here, x denotes A, B, C or D.

- Data Direction Register (DDRx) is used to configure a specific port pin as output (1) or input (0).
  - ❑ Example: To set Port A pins 0 to 3 for input, pins 4 to 7 for output, we write C code

```
DDRA = 0b11110000; // configure pins
```

bit	7	6	5	4	3	2	1	0
DDRA	1	1	1	1	0	0	0	0

for output

for input

Digital IO in ATmega16 — Reading from/Writing to Port

- Register Data Register (PORTx) is used to write output data to port.
  - ❑ Example: To write a binary 0 to output pin 6, binary 1 to other pins of Port A, we write C code

```
PORTA = 0b10111111; // write output
```

- Register Input Pins Address (PINx) is used to read input data from port.
  - ❑ Example: To read the input pins of Port A, we write C code

```
unsigned char temp; // temporary variable
temp = PINA;        // read input
```

- Where do the C names PINA, PORTA, DDRA come from?

```
// extract for header file <avr/iom16>
#define PINA    _SFR_IO8(0x19)
#define DDRA    _SFR_IO8(0x1A)
#define PORTA   _SFR_IO8(0x1B)…
```

AVR header file

- To access all AVR microcontroller registers, your program must include the header file <io.h>, which is found in the WinAVR folder.

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

- Depending on which device selected in your project, file 'io.h' will automatically redirect to a specific header file.

- Example
  - ❑ For ATmega16, the specific header file is 'avr/iom16.h'.
  - ❑ This header file is printed in Appendix A of the lab notes.
  - ❑ The header file lists the C names for all registers in ATmega16, and their memory locations.
  - ❑ We always use the C names in our code.

Digital IO in ATmega16 — Example

```
/* File: led.c
   Description: Simple C program for the ATMEL AVR uC (ATmega16 chip)
   It lets user turn on LEDs by pressing the switches on STK500 board
*/
#include <avr/io.h>     // AVR header file for all registers/pins
int main(void){
   unsigned char i;     // temporary variable

   DDRA = 0x00;        // set PORTA for input
   DDRB = 0xFF;        // set PORTB for output
   PORTB = 0x00;       // turn ON all LEDs initially

   while(1){
        // Read input from PORTA.
        // This port will be connected to the 8 switches
        i = PINA;

        // Send output to PORTB.
        // This port will be connected to the 8 LEDs
        PORTB = i;
   }
   return 1;
}
```

Demo in slide 15.