אוניברסיטת
אריאל
בשומרון

Program style

Code needs to tell a story

אוניברסיטת
אריאל
בשומרון

Every line or at least several lines

should be self-explantory

about **what** it does, and **how** it does that

Why to Bother?

אוניברסיטת
אריאל
בשומרון

- **Good style:**
  - Easy to understand code
  - Concise & polished
  - Easy to debug

- **Sloppy style ➔bad code**
  - Hard to read
  - Broken flow
  - Harder to find errors & correct them

Program style

- **Very important**. Also for others, who wants to continue working on the code
- Common sense
- Read "real" coding guidelines document – will give you insight how important it is. e.g. good one from Google (http://code.google.com/p/google-styleguide/).
- Principles:
  1. Readability
  2. Common Sense
  3. Clarity
  4. Right focus

What's your informative name

אוניברסיטת
אריאל
בשומרון

- Less reasonable
  - #define ONE 1
  - #define TEN 10
  - #define TWENTY 20

- More reasonable
  - #define INPUT_MODE 1
  - #define INPUT_BUFSIZE 10
  - #define OUTPUT_BUFSIZE 20

What's your consistent name

אוניברסיטת
אריאל
בשומרון

- Use descriptive names
- int PendingLen = 0; // current length of queue
- Naming conventions vary (style)
  - numPending,
  - num_pending,
  - NumberOfPendingEvents
- Consider (wording)
  - int noOfItemsInQ;
  - int frontOfTheQueue;
  - int queueCapacity;
- Be consistent, with yourself and peers.

What's your exhausting name

אוניברסיטת
אריאל
בשומרון

- Use short names as possible

- Compare

```
for( theElementIndex = 0;
        theElementIndex < numberOfElements;
        theElementIndex++ )
        elementArray[theElementIndex] = theElementIndex;
```

and

```
for( i = 0; i < nelems; i++ )
        elem[i] = i;
```

What's your functional name

אוניברסיטת
אריאל
בשומרון

Use active name for functions

- now = getDate()

- Compare

  if(checkdigit(c)) **…**

  to

  if(isdigit(c)) **…**

- Accurate active names makes code bugging more clear and make life easier for a new developer (getting up to speed)

8

Indentation

- **Use indentation to show structure**
- Compare

```
for(size_t n=0; n <100; field[n++] = 0);
c = 0; return '\n';
```

To

```
for(size_t n=0; n <100; n++)
{
    field[n] = 0;
}
c = 0;
return '\n';
```


9

Expressions

- **Use parentheses to resolve ambiguity**
- Compare

```
leap_year = y % 4 == 0 && y %100 != 0
  || y % 400 == 0;
```

to

```
leap_year = ((y % 4 == 0) && (y %100 != 0))
  || (y % 400 == 0);
```


10

Statements

- **Use braces to resolve ambiguity**
- Compare

```
if( i < 100 )
x = i;
i++;
```

To

```
if( i < 100 )
    {
        x = i;
    }
i++;
```


11

Idioms

Compare:

```
if( x > 0 )
   if( y > 0 )
     if( x+y < 100 )
     {
        ...
     }
     else

printf(Too large!\n" );
   else

printf("y too small!\n");
else
   printf("x too small!\n");
```

```
if( x <= 0 )
{
   printf("x too small!\n");
}
else if( y <= 0 )
{
   printf("y too small!\n");
}
else if( x+y >= 100 )
{
   printf("Sum too large!\n" );
}
else
{
  ...
}
```


12

Comments

אוניברסיטת
אריאל
בשומרון

- Don’t write the obvious
  - // return SUCCESS

    return SUCCESS;

  - // Initialize total to number_received

    total = number_received;

- **Test:**
  - Does comment add something that is not evident from the code?

Style summary

- Descriptive names
- Clarity in expressions
- Straightforward flow
- Readability of code & comments
- Consistent conventions & idioms

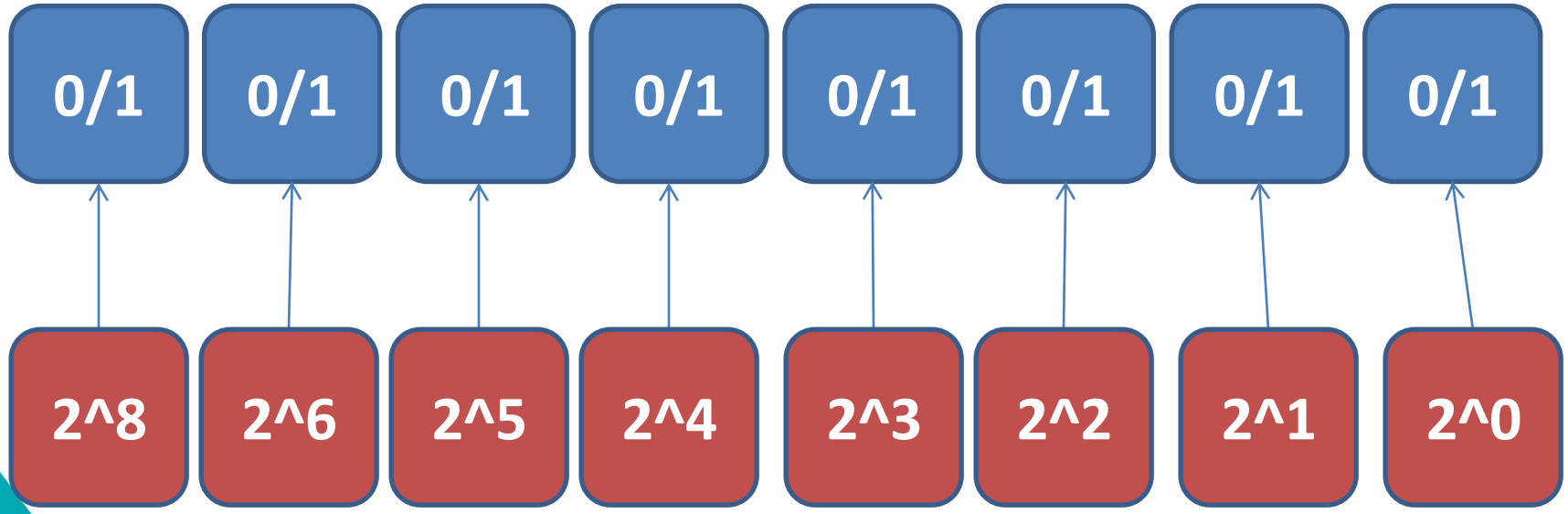
אוניברסיטת
אריאל
בשומרון

Data Types &
Memory & Representation

Memory allocation

- Bit – a binary digit - zero or one

- Byte – 8 bits



Basic data types

אוניברסיטת
אריאל
בשומרון

- Primitive data types are similar to JAVA
  - Char
  - Int
  - Short
  - Long
  - Float
  - Double

- Unlike in JAVA, All can be **signed/unsigned.** Default: signed

- Unlike in JAVA, the **types sizes are machine dependant**!

Basic data types

אוניברסיטת
אריאל
בשומרון

Type	Size (bytes)	Format Specifier
`int`	at least 2, usually 4	`%d`, `%i`
`char`	1	`%c`
`float`	4	`%f`
`double`	8	`%lf`
`short int`	2 usually	`%hd`
`unsigned int`	at least 2, usually 4	`%u`
`long int`	at least 4, usually 8	`%ld`, `%li`
`long long int`	at least 8	`%lld`, `%lli`
`unsigned long int`	at least 4	`%lu`
`unsigned long long int`	at least 8	`%llu`
`signed char`	1	`%c`
`unsigned char`	1	`%c`
`long double`	at least 10, usually 12 or 16	`%Lf`

Signed VS. Unsigned

אוניברסיטת
אריאל
בשומרון

- The property can be applied to most of the numeric data types including int, char, short and long
- Unsigned - can hold zero and positive numbers,
- Signed - holds negative, zero and positive numbers.
- Example - N-bit integers –
  - Unsigned integer has a range of $[0, 2^N -1]$
  - Signed integer goes from $[-2^{N-1}-1, +2^{N-1}]$, Zero mean
  - The range is the same, but it is shifted on the number line.

1st Vs. 2nd complements

אוניברסיטת
אריאל
בשומרון

- One's complement and two's complement are two important binary concepts.
- One's complement is the interim step to finding the two's complement.
- One's complement - all bits in a byte are inverted by changing each 1 to 0 and each 0 to 1, we have formed the one's complement of the number.
- Two's complement is especially important because it allows us to represent signed numbers
- Two's complement also provides an easier way to subtract numbers

1st complement

- Invert all bits so 0→1 and 1→0

Original Value	One's Complement
0	1
1	0
1010	0101
1111	0000
11110000	00001111
10100011	01011100
11110000 10100101	00001111 01011010

2nd complement

- Invert all bits so 0→1 and 1→0 and add 1

0 1 1 0 1 1 1 0	Original binary value
1 0 0 1 0 0 0 1	1's complement
1 0 0 1 0 0 0 1 + 1	
1 0 0 1 0 0 1 0	2's complement

2nd complement - example

- Present the 6-bits binary code for 27-6

חיבור ארוך
שבו 1+1=0
ומעבירים
את ה-1
הנוסף לצד
שמאל

$27=16+8+2+1=2^4+2^3+2^1+2^0=011011$

$6=2^2+2^1=000110$

$(-6)=111001+1=111010$

011011 +

<u>111010</u>

010101 = $2^0+2^2+2^4=1+4+16=21$

Most significant bit is ignored as we are working with 6-bit accuracy

The sizeof() operator

אוניברסיטת
אריאל
בשומרון

- We can use the sizeof() operator to check the size of a variable.
- It can be used in different ways
- When operand is a data type -

```
#include <stdio.h>
int main()
{
        printf("%lu\n", sizeof(char));
        printf("%lu\n", sizeof(int));
        printf("%lu\n", sizeof(float));
        printf("%lu", sizeof(double));
        return 0;
}
```

→

```
1
4
4
8
```

The sizeof() operator

אוניברסיטת
אריאל
בשומרון

- When operand is an expression -

```
#include <stdio.h>
int main()
{
        int a = 0;
        double d = 10.21;
        printf("%d", sizeof(a + d));
        return 0;
}
```

8

Size of int and double is 4 and 8 respectively, a is int variable while d is a double variable. The final result will be a double, Hence the output of our program is 8 bytes.

Example - conversions

אוניברסיטת
אריאל
בשומרון

Decimal to Hexadecimal

Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Decimal Value = 775

Integer Part	Quotient	Remainder	Remainder in Hexadecimal
775 / 16	48	7	7
48 / 16	3	0	0
3 / 16	0	3	3

Hexadecimal Value = 307

$$(775)_{10} = (307)_{16}$$

Decimal Value = 1256

Integer Part	Quotient	Remainder	Remainder in Hexadecimal
1256 / 16	78	8	8
78 / 16	4	14	E
4 / 16	0	4	4

Hexadecimal Value = 4E8

$$(1256)_{10} = (4E8)_{16}$$

Example – Convert decimal number 210 into octal number.

Since given number is decimal integer number, so by using above algorithm performing short division by 8 with remainder.

Division	Remainder (R)
210 / 8 = 26	2
26 / 8 = 3	2
3 / 8 = 0	3

Additional example

אוניברסיטת
אריאל
בשומרון

- Write a C program to print the alphabet set in decimal and character form

```
#include <stdio.h>
 #define N 10
int main()
{
  char chr;
  printf("\n");
  for (chr = 65; chr <= 122; chr = chr + 1)
  {
    if (chr > 90 && chr < 97)
      continue;
    printf("[%2d-%c] ", chr, chr);
   }
   return 0;
}
```

Chars

- Chars can represent small integers or a single character.
- Examples:
  - char ch = ‘A’;
  - char ch = 65;
  - printf("The character %c has the ASCII code %u.\n", ch, ch);
  - for (char ch= 'A'; ch <= 'Z'; ++ch)

```
{
        printf("%c", ch);
}
```

  - putchar(), getchar() – similar to printf and scanf BUT for single characters

Boolean types

- Boolean type **doesn't exist in C**!
- Use char/int instead
- zero = false, non-zero = true
- Examples:

```
while (1)
{
}
```

infinite loop

```
#define TRUE 1
while (TRUE)
{
}
```

infinite loop

```
i = (3==4);
```

i equals zero

```
if (-1974)
{
}
```

true statement

Casting

אוניברסיטת
אריאל
בשומרון

- Operands with different types get converted when you do arithmetic.
- Everything is converted to the type of the **floatiest, longest operand, signed if possible without losing bits**
- Casting possible between all primitive variable types.
- Casting up – short (2bits) => int (4bits) => long (8bits)
- Casting down – long => int=> short

```
int i;
short s;
long l;
i=s;   // no problem
l=i;   // no problem
s=l;   // might lose info,
       // warning not guaranteed
```

Arithmetic operators

- If both the operands of an arithmetic operation belong to the *same type*, the operation is carried out in that type, and the result belongs to that type. For example, int/int → int; double/double → double.
- However, if the two operands belong to *different types*, the compiler promotes the value of the *smaller* type to the *larger* type (known as *implicit type-casting*). The operation is then carried out in the *larger* type. For example, int/double → double/double → double.

```
float f=1/3;
float f=(float)1/3;
float f=1/3.0;
```

int/int=int => 0

(float) convert to float => 0.33

.0 covert to float => 0.33

Implicit type-casting

אוניברסיטת
אריאל
בשומרון

Type	Example	Operation
int	2 + 3	int 2 + int 3 → int 5
double	2.2 + 3.3	double 2.2 + double 3.3 → double 5.5
mix	2 + 3.3	int 2 + double 3.3 → double 2.0 + double 3.3 → double 5.3
int	1 / 2	int 1 / int 2 → int 0
double	1.0 / 2.0	double 1.0 / double 2.0 → double 0.5
mix	1 / 2.0	int 1 / double 2.0 → double 1.0 / double 2.0 → double 0.5

Expressions as Values

- && - logical “and”
  - & - bitwise “and”
- || - logical “or”
  - | - bitwise “or”

```
int i=0;
if (i==1 && x>5)
{
   …
}
```

Might not be evaluated

Bitwise vs Logical Operators	
Bitwise operator is the type of operator provided by the programming language to perform computations.	Logical Operator is a type of operator provided by the programming language to perform logic-based operations.
Functionality	
Bitwise operators work on bits and perform bit by bit operations.	Logical operators are used to making a decision based on multiple conditions.
Themes	
Bitwise operators are &, |, ^, ~, <<, >>.	Logical operators are &&, ||, !

Example (Bitwise Vs. Logical)

אוניברסיטת
אריאל
בשומרון

- Write a program that gets 2 variables (x=3, y=7) and check - if y>1 and y>x

  it outputs z=x+y.

Example (Bitwise Vs. Logical)

אוניברסיטת
אריאל
בשומרון

- Write a program that gets 2 variables (x=3, y=7) and check - if y>1 and y>x

  it outputs z=x+y.

```
int main()
        {
                int x = 3; //...0011
                int y = 7; //...0111
                if (y > 1 && y > x)
                        int z = x & y; // 1010;
        }
```

אוניברסיטת
אריאל
בשומרון

Functions declarations and definitions

Declarations and definitions

אוניברסיטת
אריאל
בשומרון

- Function declaration:
  - Specification of the function prototype.
  - No specification of the function operations (during definition).
  - <OutputType> <FunctionName> (<InputType>)
    - int func(int a, int b);
  - int g(), f, x,i, k();

    The above line declares 3 int variables and 2 int functions.

- Function definition
  - Specification of the exact operations the function performs.

Declarations and definitions

אוניברסיטת
אריאל
בשומרון

- If I declare a function f with arguments, but define it without them, will it make a difference ? Should I be able to address the arguments from the function body ?

It will not be compiled

- *void f(void);* means that f does not take any parameters.
- *void f();* means that function f may or may not have parameters ,and if it does - we don't know what kind of parameters those are ,or how many there is of them.

אוניברסיטת
אריאל
בשומרון

enum

enums – why?

אוניברסיטת
אריאל
בשומרון

- The basic reason to have "enums" is to avoid "magic numbers“.
- Let's say you have three "states": STOP, CAUTION and GO. How do you represent them in your program?
  - One way is to use the string literals "STOP", "CAUTION" and "GO". But that has several problems - including the fact that you can't use them in a C "switch/case" block.
  - Another way is to Map" them to the integer values "0", "1" and "2". You can use them in “switch/case” block, but the names are not meaningful.
  - Seeing "STOP" in your code is a lot more meaningful than seeing a "0". Using "0" in your code just like that is an example of a "magic number". Magic numbers are bad: you want to use a "meaningful name" instead.

Before enums were introduced in the language,

- C programmers used macros:

```
#define STOP 0
#define CAUTION 1
#define GO 2
```

- A better, cleaner approach in modern C/C++ is to use an enum instead:

```
enum traffic_light_states {
  STOP,
  CAUTION,
  GO
};
```

- Using a "typedef" just simplifies declaring a variable of this type:

```
typedef enum {   STOP,
  CAUTION,
  GO
} traffic_light_states;
```

enum – define a new type

אוניברסיטת
אריאל
בשומרון

- Enum is a data type that consists of integers
- To define enums, the enum keyword is used

```
enum Season
{
        WINTER,// = 0 by default
        SPRING, // = WINTER + 1
        SUMMER,         // = WINTER + 2
        AUTUMN          // = WINTER + 3
};
```

enum

- The keyword ‘enum’ is used to declare new enumeration types in C and C++. Below you can see an example of enum declaration.

```
// The name of enumeration is "flag" and the constant
// are the values of the flag. By default, the values
// of the constants are as follows:
// constant1 = 0, constant2 = 1, constant3 = 2 and
// so on.
enum flag{constant1, constant2, constant3, ....... };
```

enum

- Variables of type enum can be defined in two ways:

```
// In both of the below cases, "day" is
// defined as the variable of type week.

enum week{Mon, Tue, Wed};
enum week day;

// Or

enum week{Mon, Tue, Wed}day;
```

week – general variable of type enum

day – specific variable that will be part of "type" week

Another example - your turn

- Get three colors (red, green and blue) and choose the red one.
- Do that with and without typedef

Another example - your turn

- Using the *tag name* just after the *enum*

```
enum color
{
    RED,
    GREEN,
    BLUE
};
```

This enumeration must then always be used with the keyword *and* the tag like this:

```
enum color chosenColor = RED;
```

- Using *typedef* directly when declaring the *enum,* omitting the tag name (“color”) and using the type without the enum keyword

```
typedef enum
{
    RED,
    GREEN,
    BLUE
} color;

color chosenColor = RED;
```

Enum – numbering the variables

אוניברסיטת
אריאל
בשומרון

```
#include<stdio.h>
enum week{Mon=10, Tue, Wed, Thur, Fri=10, Sat=16, Sun};
enum day{Mond, Tues, Wedn, Thurs, Frid=18, Satu=11, Sund};
int main() {
   printf("The value of enum week: %d\t%d\t%d\t%d\t%d\t%d\t%d\n\n",Mon
   printf("The default value of enum day: %d\t%d\t%d\t%d\t%d\t%d\t%d",M
   return 0;
}
```

What is the output of this code ?

The value of enum week : 10 11 12 13 10 16 17

The default value of enum day : 0 1 2 3 18 11 12

Exercise

- Please write a program (contains enum) that gets a number between 1 and 4 and outputs the season (spring, summer, autumn and winter)

```c
#include <stdio.h>

enum _season{spring=1, summer, autumn, winter} season;

int main()
{
    int val;
    printf(" enter the number of the season: ");
    scanf("%d", &val);

    season = val;

    switch (season) {
        case 1:
            printf("spring");
            break;
        case 2:
            printf("summer");
            break;
        case 3:
            printf("autumn");
            break;
        case 4:
            printf("winter");
            break;
        default:
            printf("the seasons are only 4");
    }
    return 0;
}
```

- _season is a defined enum type
- season is a specific variable of type _season

אוניברסיטת
אריאל
בשומרון

switch

Switch---Case

- Please write a program that ask the user to choose one color from red, green, blue and print the “favorite color is:” . Please use switch command

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{

    int color;

    /* ask user to choose color */
    printf("Please choose your favorite color: (0. red, 1. green, 2. blue):");
    scanf("%d", &color);

    /* print out the result */
    switch (color)
    {
    case 0:
        printf("your favorite color is Red");
        break;
    case 1:
        printf("your favorite color is Green");
        break;
    case 2:
        printf("your favorite color is Blue");
        break;
    default:
        printf("you did not choose any color");
    }

    return 0;
}
```

enum---Switch---Case

- Please write the same program but now – do that with *enum*

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{

    enum color { red, green, blue };

    enum  color favorite_color;

    /* ask user to choose color */
    printf("Please choose your favorite color: (1. red, 2. green, 3. blue):");
    scanf("%d", &favorite_color);

    /* print out the result */
    switch (favorite_color)
    {
    case red:
        printf("your favorite color is Red");
        break;
    case green:
        printf("your favorite color is Green");
        break;
    case blue:
        printf("your favorite color is Blue");
        break;
    default:
        printf("you did not choose any color");
    }

    return 0;
}
```

enum---switch importance

אוניברסיטת
אריאל
בשומרון

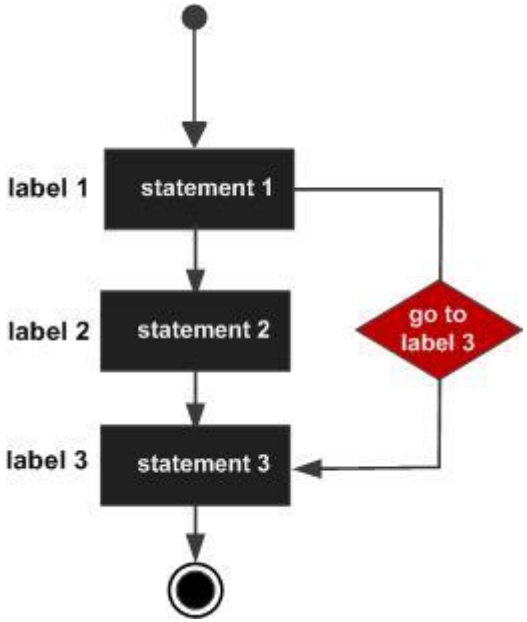
- Conclusion –
  - If you look at the switch-case code, you will see that it is not perfect in term of readability.
  - In the switch-case statement, we used integer numbers to represent colors.
  - It is better to use symbolic names like red, green and blue to represent integer constants 0, 1 and 2. For that purpose - we have enum.

אוניברסיטת
אריאל
בשומרון

goto

Goto command

- Use of **goto** statement is highly discouraged because it makes difficult to trace the control flow of a program, making the program hard to understand and hard to modify.
- Any program that uses a goto can be rewritten to avoid them




```
goto label;
... .. ...
... .. ...
label:
statement;
```

Goto command

- Write a program to calculate the sum and average of positive numbers.

  If the user enters a negative number, the sum and average are displayed

```
// Program to calculate the sum and average of positive numbers
// If the user enters a negative number, the sum and average are displayed.

#include <stdio.h>

int main() {

   const int maxInput = 100;
   int i;
   double number, average, sum = 0.0;

   for (i = 1; i <= maxInput; ++i) {
      printf("%d. Enter a number: ", i);
      scanf("%lf", &number);

      // go to jump if the user enters a negative number
      if (number < 0.0) {
         goto jump;
      }
      sum += number;
   }

jump:
   average = sum / (i - 1);
   printf("Sum = %.2f\n", sum);
   printf("Average = %.2f", average);

   return 0;
}
```