

Overview: Fundamentals of C, part 2

- Input and output with C
- Conditionals and Relational operators
- Logical operators

1

Output with C – printf()

```
int length=3, x=1, y=2;
float area=length*2;
printf("Hello World.\n");
printf("the value of 1+1 is %d\n",2);
printf("the value of 1+1 is %d\n",1+1);
printf("the value of y/x is %d\n",y/x);
printf("Length is %d\n",length);
printf("Area is %.4f\n",area);
printf("x is %d and y is %d\n",x,y);
```

screen output:
Hello World.
the value of 1+1 is 2
the value of 1+1 is 2
the value of y/x is 2
Length is 3
Area is 6.0000
x is 1 and y is 2

- **placeholders** (e.g. %d, %.4f) are substituted with the values of their corresponding **arguments** (left-to-right) when output sent to the screen
 - argument can be constant, variable, or expression
 - if the placeholder type doesn't match the type of the corresponding argument: program compiles ok, but garbage values output for this and subsequent placeholders
 - if there are more placeholders than arguments: program compiles ok, but garbage values output for the unmatched placeholders
 - if there are fewer placeholders than arguments: no problem!

2

Output with C (contd.)

```
int value=-145;
printf("value is %d units\n",value);
printf("value is %i units\n",value);
printf("value is %4d units\n",value);
printf("value is %3d units\n",value);
printf("value is %6d units\n",value);
printf("value is %-6d units\n",value);
```

screen output:
value is -145 units
value is -145 units
value is -145 units
value is -145 units
value is -145 units
value is -145 units

Notes: minimum field width will be *increased if necessary* to print the value

%- means *left justify*

%+ means *always print the sign (+ or -)*

```
double dbl=157.8926;
printf("dbl is %f units\n",dbl);
printf("dbl is %.2f units\n",dbl);
printf("dbl is %+8.2f units\n",dbl);
printf("dbl is %7.5f units\n",dbl);
printf("dbl is %e units\n",dbl);
printf("dbl is %.3E units\n",dbl);
```

screen output:
dbl is 157.892600 units
dbl is 157.89 units
dbl is +157.89 units
dbl is 157.89260 units
dbl is 1.578926e+02 units
dbl is 1.579E+02 units

3

Output with C – escape sequences

used to represent characters which would be awkward or impossible to enter directly into a source program:

\a	alert (bell)
\b	backspace
\f	formfeed
\n	newline
\r	carriage return
\t	horizontal tab
\v	vertical tab
\\	backslash
\?	question mark
\'	single quote
\"	double quote

4

Input with C – scanf()

```
float value_punt;
int intvar1, intvar2;
double dblvar;

printf("enter a value for value_punt: ");
scanf("%f",&value_punt);

printf("enter a value for intvar1: ");
scanf("%d",&intvar1);

printf("enter a value for intvar2: ");
scanf("%i",&intvar2);

printf("enter a value for dblvar: ");
scanf("%lf",&dblvar);
```

& is the unary address-of operator

5

Input with C (contd.)

more than one input value at a time:

```
double distance, velocity;
printf("enter values for distance and velocity: ");
scanf("%lf %lf",&distance, &velocity);
```

↙
space between %lf's means: input values will be separated by *one or more whitespace characters* (e.g. blank space, <CR>, tab)

scanf() is easy to get wrong – if placeholders and arguments don't match exactly, the results are "unpredictable". Be careful!

6

Conditionals in C

- all programming languages have some sort of **conditional** statement: do different things depending on some **condition** or **test**

- if** statement (the most common conditional statement):

```
if (numteams > 2) {
    numteams = 2;
    printf("You had too many teams!\n");
}
```

general
syntax:

```
if (condition) {
    statements;
}
```

the **statements** are executed if the **condition** evaluates to **true**

- if **condition** evaluates to a **non-zero** value: **true**
- if **condition** evaluates to **zero**: **false**

7

Relational operators in C

Relational operators – for building the conditions:

==	equal to
!=	not equal to
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to

Warning: do not confuse == (equality) and = (assignment)

8

Conditionals and relational operators in C – examples

```
int a = 20;
if (a != 10) {
    printf("a is not equal to 10\n");
}
if (a > 10) {
    printf("a is greater than 10\n");
}
if (a == 10) {
    printf("a is equal to 10\n");
}
```

produces the screen output: **a is not equal to 10**
 a is greater than 10

9

Conditionals and relational operators in C – examples

```
int x = 10;
int y = 10*x;
if (y > x) {
    if (x < 100) {
        if (y > 80) {
            printf("A");
        }
        printf("B");
    }
    printf("C");
}
printf("D\n");
```

produces the screen output:

10

Conditionals and relational operators in C

- convert mark to grade: 70 & over is **A**, under 40 is **C**, otherwise **B**

```
float mark;
char grade; /* character datatype */
printf("enter an exam mark: ");
scanf("%f", &mark);
if (mark >= 70) {
    grade = 'A';
}
if ((mark >= 40) && (mark < 70)) { /* logical AND */
    grade = 'B';
}
if (mark < 40) {
    grade = 'C';
}
printf("The grade is: %c\n", grade);
```

Problem? All 3 conditions are tested regardless of the value of **mark**. But if 1st condition is **true**, no point in testing 2nd and 3rd; if 2nd is **true**, no point in testing 3rd

11

Conditionals and relational operators in C – if-else

- **else** specifies alternative action to take if condition evaluates to **false**

```
if (condition) {
    statements; /* executed if condition true */
} else {
    statements; /* executed if condition false */
}
```

so previous program could be re-written:

```
if (mark >= 70) {
    grade = 'A';
} else {
    if ((mark >= 40) && (mark < 70)) {
        grade = 'B';
    } else {
        if (mark < 40) {
            grade = 'C';
        }
    }
}
```

12

Logical operators in C

- combine conditions to form more complex ones by using the logical operators:

<code>if (condition1 && condition2) {</code> statements; <code>}</code>	AND
<code>if (condition1 condition2) {</code> statements; <code>}</code>	OR
<code>if (!condition) {</code> statements; <code>}</code>	NOT

13

Logical operators in C (contd.)

if **A** and **B** are conditions in C:

A	B	A&&B	A B	!A	!B
False	False	False	False	True	True
False	True	False	True	True	False
True	False	False	True	False	True
True	True	True	True	False	False

Example: if salary exceeds €30,000 and price is below €75,000, print “OK”; otherwise print “No”

```
if ((salary > 30000) && (price < 75000)){
    printf("OK\n");
} else {
    printf("No\n");
}
```

14

Logical operators in C (contd.)

another example: a year is a Leap Year if it is divisible by 4 (e.g. 1980, 2004), but “century” years only count if they are divisible by 400 (1600, 2000 – yes; 1900, 2100 – no).

One possible solution:

```
if (((year%4 == 0) && (year%100 != 0)) ||
    ((year%100 == 0) && (year%400 == 0))) {
    printf("%d is a Leap Year\n",year);
} else {
    printf("%d not a Leap Year\n",year);
}
```

15

Logical operators in C – Boolean algebra

- need to be careful with parentheses:

A && (B && C) == (A && B) && C == A && B && C
A || (B || C) == (A || B) || C == A || B || C
A && (B || C) != (A && B) || C == A && B || C

CHECK WITH “TRUTH TABLE” – ALL POSSIBLE COMBINATIONS OF VALUES FOR A,B,C

- de Morgan’s Theorem:

!(A && B) == !A || !B
!(A || B) == !A && !B

A	B	A&&B	!(A&&B)	!A	!B	!A !B
F	F	F	T	T	T	T
F	T	F	T	T	F	T
T	F	F	T	F	T	T
T	T	T	F	F	F	F



16

Logical operators and expressions

What is the value of `y` in each case?

```
int i=3, j=5, x=5, y;
```

```
y = i && j;
```

```
y = i + j > 10 || x < 1;
```

```
y = ! ! j;
```

Note: when evaluating a condition, C interprets *non-zero value* as **True** and *zero value* as **False**. However, when a condition is evaluated to **True**, C assigns it the numerical value **1**; when it is **False**, C assigns it the numerical value **0**.

17

Operator precedence in C

OPERATOR PRECEDENCE (partial list)	ASSOCIATES
!	R-to-L
* (multiplication) / %	L-to-R
+ (addition) - (subtraction)	L-to-R
< > <= >=	L-to-R
== !=	L-to-R
&&	L-to-R
	L-to-R
=	R-to-L

- if you have in your program:

```
I + 1 > 5 || J - 1 < 10 && K < I + J
```

- then the C compiler interprets this expression as:

```
((I+1) > 5) || (((J-1) < 10) && (K < (I+J)))
```

- putting in explicit parentheses is always a good idea!

18