CS100
Introduction to Programming

Lecture 3. Simple Input / Output

About HW1

- Plagiarism is academic misconduct, and we take it very serious at ShanghaiTech. Any student who plagiarizes will be heavily punished. There is an important message on plagiarism on the first page of the homework description, please read carefully and make sure you understand it.

- We will be using professional coding plagiarism detection software, which will not be deceived by simple tricks. So do not simply think changing variable names after copying someone else's code will make you safe.

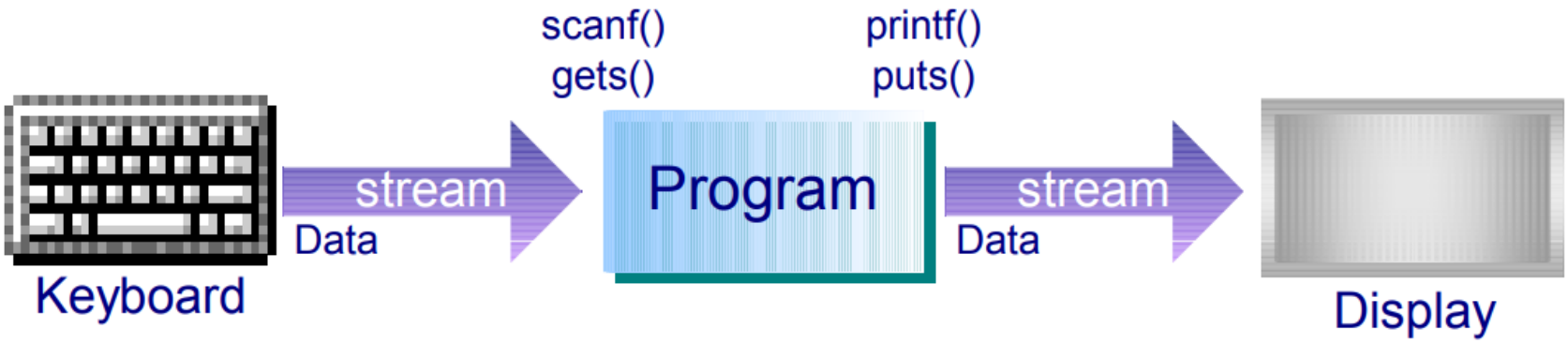
- Simple late submission policy:
  - 50% penalty if submitted before 23:59 on the day following the deadline
  - 100% penalty if no submission by 23:59 on the day following the deadline

Formatted Input/Output

Formatted Input/Output

- **Input/output (I/O)** is the way a program communicates with the user. For C, the I/O operations are carried out by the I/O functions in the I/O libraries.

- Input from the keyboard / output to the monitor screen is referred to as **standard input/output**.



I/O Functions

- A function is a piece of code to perform a specific task.
- A library contains a group of functions, usually for related tasks, e.g. standard I/O functions are in the library <stdio.h>, maths functions in the library <math.h>

- To use the I/O functions in <stdio>, the line

  #include <stdio.h>

  need be included as the preprocessor instructions in a program

- Two I/O functions are used most frequently:
  - **printf(…)** : output function
  - **scanf(…)** : input function

Simple Output: printf()

- The printf() statement has the form:

  printf(control-string, argument-list);

- The **control-string** is a string constant. It is printed on the screen.
  - %?? is a conversion specification. An item will be substituted for it in the printed output.

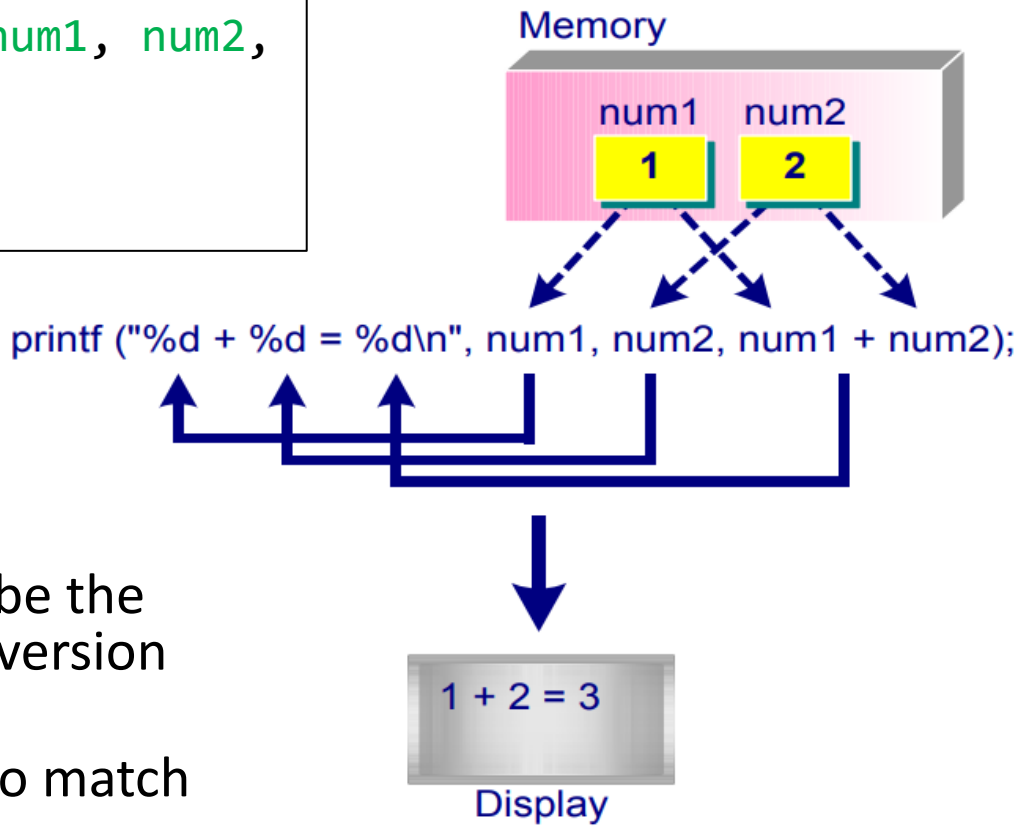
- The **argument-list** contains a list of items such as item1, item2, …, etc.
  - Values are to be substituted into places held by the conversion specification in the control string.
  - An item can be a constant, a variable or an expression like num1 + num2.

printf() – Example 1

```
#include <stdio.h>
int main()
{
   int num1 = 1, num2 = 2;
   printf("%d + %d = %d\n", num1, num2,
      num1 + num2);
   return 0;
}
```

Output:

1 + 2 = 3



Note:

- The number of items must be the same as the number of conversion specifiers.
- The types of items must also match the conversion specifiers.

printf() – Conversion Specification

Type of *Conversion Specifiers*

d	signed decimal conversion of int
o	unsigned octal conversion of unsigned
x, X	unsigned hexadecimal conversion of unsigned
c	single character conversion
f	signed decimal floating point conversion
s	string conversion

printf() – Example 2

```
#include <stdio.h>
int main()
{
  int num = 10;
  float i = 10.3;
  double j = 100.0;

  printf("int num = %d\n", num);
  printf("float i = %f\n", i);
  printf("double j = %f\n", j);
   /* by default, 6 digits are
      printed after the decimal
      point */

  return 0;
}
```

Output:

int num = 10

float i = 10.300000

double j = 100.000000

Examples of Escape Sequence

- Some useful **non-printable control characters** are referred to by the ***escape sequence*** which is a better alternative, in terms of memorization, than numbers, e.g. `'\n'` the newline (or linefeed) character instead of the number **10**.

'\a'	**alarm bell**	'\f'	**form feed**	'\n'	**newline**
'\t'	**horizontal tab**	'\"'	**double quote**	'\v'	**vertical tab**
'\b'	**back space**	'\\'	**backslash**	'\r'	**carriage return**
'\''	**single quote**				

General Structure of Conversion Specification for Formatted Output

- A conversion specification is of the form

  % [flag] [minimumFieldWidth] [.precision] [sizeSpecification] conversionSpecifier

  - ***%*** and ***conversionSpecifier*** are compulsory. The others are optional.
  - We will focus on using **%** and **conversionSpecifier** for printing integers, floating point numbers and strings.
  - Students should refer to the reference book or web materials for other options of formatted output.

Printing Integer Values

	Conversion Specification	Flag	Field Width	Conversion Specifier	Output on Screen
(1)	**%d**	none	none	**d**	125
(2)	**%+6d**	+	6	**d**	□□+125
(3)	**%-6d**	–	6	**d**	125□□□

- A ***flag*** is used to control the display of plus or minus sign of a number, and left or right justification.
  - The **+ flag** is used to print values with a plus sign "+" if positive, and a minus sign "–" otherwise.
  - The **– flag** is used to print values left-justified.
- The ***minimum field width*** gives the lower bound of the field width to be used during printing (padding with blanks or zeros if the item is less wide than it)

Printing Floating-Point Values

	Conversion Specification	Flag	Field Width	Precision	Conversion Specifier	Output on Screen
(1)	**%f**	none	none	none	f	10.345689
(2)	**%+11.5f**	+	11	5	f	☐☐+10.34568
(3)	**%−11.5f**	−	11	5	f	10.34568☐☐☐
(4)	**%+12.3e**	+	12	3	e	☐☐+1.034e+01
(5)	**%−12.3e**	−	12	3	e	1.034e+01☐☐☐

- The ***precision*** field can be used for printing floating-point numbers. The precision field specifies **the number of digits after the decimal point** to be printed.

Simple Input: scanf()

- A scanf() statement has the form:

scanf(control-string, argument-list);

- The **control-string** is a string constant containing conversion specifications.
- The **argument-list** contains a list of items.
  - The items in scanf() may be any variable matching the type given by the conversion specification. It cannot be a constant. It cannot be an expression like n1 + n2.
  - The variable name has to be preceded by an & ("ampersand") sign. This is to tell scanf() the address of the variable so that scanf() can read the input value and store it in the variable.
- scanf() stops reading when it has read **all** the items as indicated by the control string or the EOF (end of file) is encountered.

scanf() – Example 1

```
#include <stdio.h>

int main()
{
   int n1, n2;
   printf("Please enter 2 integers:\n");
   scanf("%d %d", &n1, &n2);
   printf("The sum = %d\n", n1 + n2);
   return 0;
}
```

Output:

Please enter 2 integers:

5 10

The sum = 15

scanf() – Example 2

```
#include <stdio.h>
int main()
{
   int number;
   printf("Please enter a number:");
   scanf("%d", &number);
   printf("The number read is %d\n", number);
   // read in a char
   char reply;
   printf("Correct(y/n)?");
   scanf("%c", &reply);
   printf("your reply: %c\n", reply); // display char
   return 0;
}
```

Output:

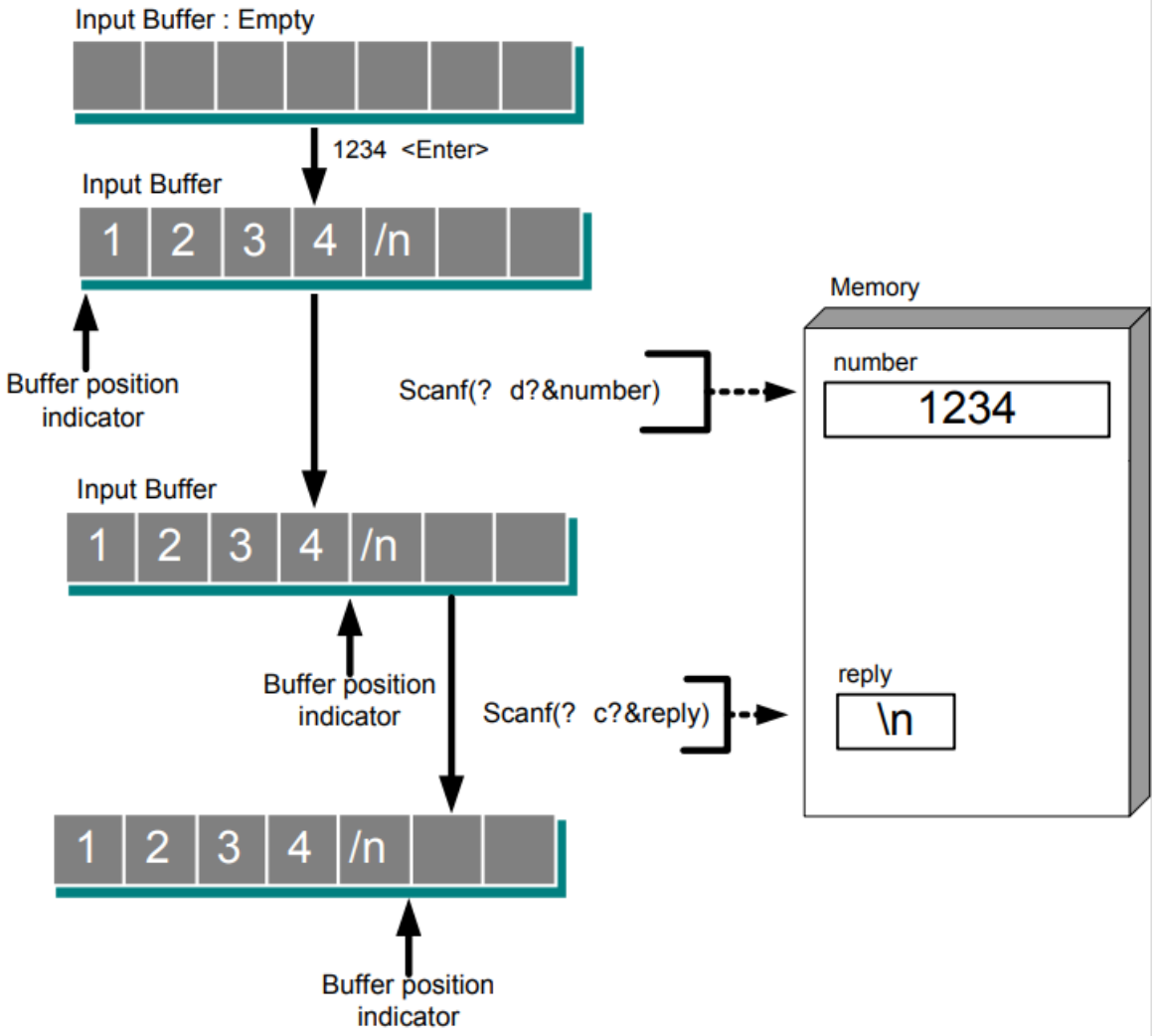
Please enter a number: ***1234<Enter>***

The number read is 1234

Correct(y/n)? your reply:

an error here

scanf() – Example 2



Reason:

There is a hidden character '\n' entered when you type *1234<Enter>*

scanf() – Example 2

- **Solution 1: ?**

```
   ...
fflush(stdin); // flush the input buffer with newline
printf("Correct(y/n)?");
scanf("%c", &reply);
printf("your reply: %c\n", reply);
   ...
```

- **Solution 2:**

```
   ...
printf("Correct(y/n)?");
scanf("\n%c", &reply); // read the newline
printf("your reply: %c\n", reply);
   ...
```

Character Input/Output

putchar()

- The syntax of calling putchar() is

  putchar(characterConstantOrVariable);

- It is equivalent to

  printf("%c", characterConstantOrVariable);

- The difference is that **putchar() is faster** because printf() need process the control string for formatting. Also, if an error occurs, **putchar()** returns either the integer value of the written character or EOF.

getchar()

- The syntax of calling getchar() is

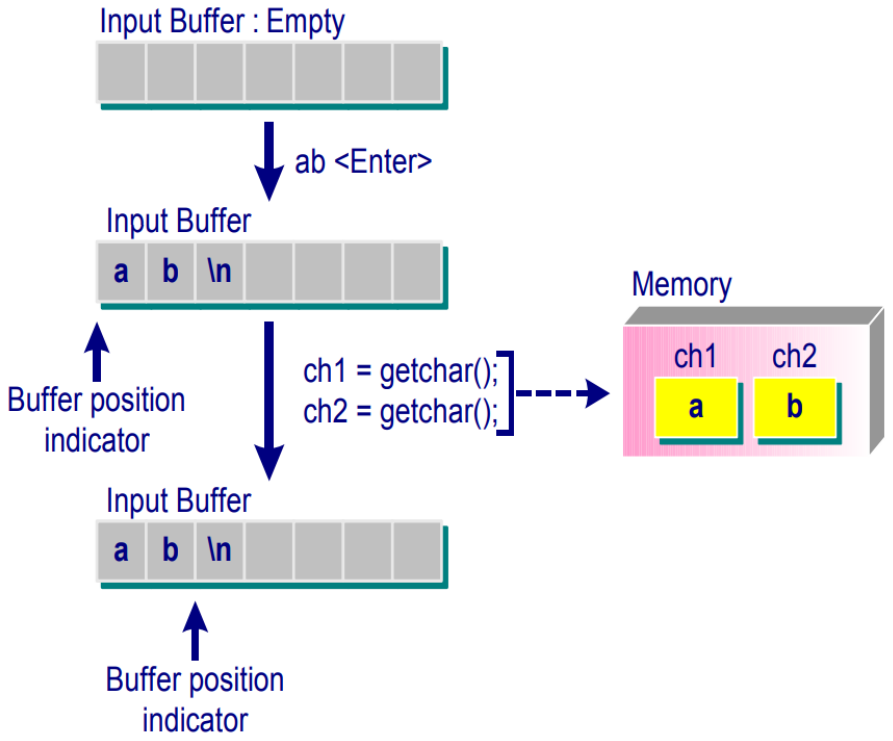
  ch = getchar(); // ch is a character variable.

- It is equivalent to

  scanf("%c", &ch);

Character Input/Output - Example

```
/* example to use getchar() and putchar() */
#include <stdio.h>
int main()
{
   char ch, ch1, ch2;
   putchar('1');
   putchar(ch='a');
   putchar('\n');
   printf("%c%c\n", 49, ch);
   ch1 = getchar();
   ch2 = getchar();
   putchar(ch1);
   putchar(ch2);
   putchar('\n');
   return 0;
}
```



Output:

1a

1a

ab *(User Input)*

ab

Logical Operations

Relational Operators

Used for **comparison** between **two values**.

Return **Boolean** result: **true** or **false**.

Relational Operators:

operator	example	meaning
==	ch == 'a'	equal to
!=	f != 0.0	not equal to
<	num < 10	less than
<=	num <=10	less than or equal to
>	f > -5.0	greater than
>=	f >= 0.0	greater than or equal to

Logical Operators

- Work on one or more relational expressions to yield a logical value: **true** or **false**.
- Allow testing and combining the results of comparison expressions.

Logical Operators:

operator	example	meaning
!	!(num < 0)	not
&&	(num1 > num2) && (num2 >num3)	and
||	(ch == '\t') || (ch == ' ')	or

	A is true	A is false
!A	false	true

A && B	A is true	A is false
B is true	true	false
B is false	false	false

A || B	A is true	A is false
B is true	true	true
B is false	true	false

Precedence of operators

- List of operators of **decreasing precedence**:

!	not
* /	multiply and divide
+ -	add and subtract
< <= > >=	less, less or equal, greater, greater or equal
== !=	equal, not equal
&&	logical and
||	logical or

- **Example**: The expression `!(5 >= 3)||(7 > 3)` is true, where the **logical or operator ||** is executed in the end.

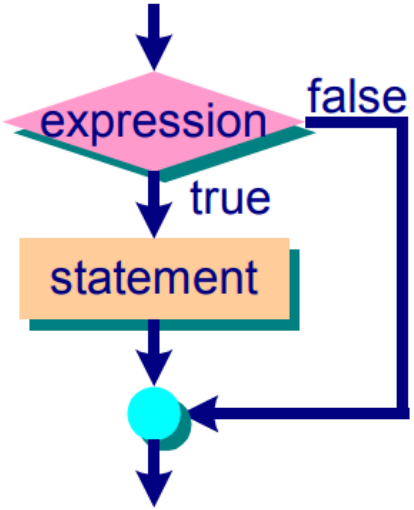
Boolean Result

- The result of evaluating an expression involving relational and/or logical operators is
  - either **true** or **false**
  - either **1** or **0**
  - When the result is **true**, it is **1**. Otherwise, it is **0**. That is, the C language uses 0 to represent a false condition.
- In general, **any integer expression whose value is non-zero is considered true**; otherwise it is **false**.
- Examples:

`3`	is true
`0`	is false
`1 || 0`	is true
`!(5 >= 3) || 0`	is false

The if Statement

```
if (expression)
    statement;
    /* simple or compound statement
    enclosed with brackets */
```




```
#include <stdio.h>
int main(void)
{
   int num;  /* value supplied by user. */
   printf("Give an integer from 1 to 10: ");
   scanf("%d", &num);
   if (num > 5)
      printf("Your number is larger than 5.\n");
   printf("%d is the number you entered.\n", num);
   return 0;
}
```

Output 1:

Give an integer from 1 to 10: ***3***

3 is the number you entered.

Output 2:

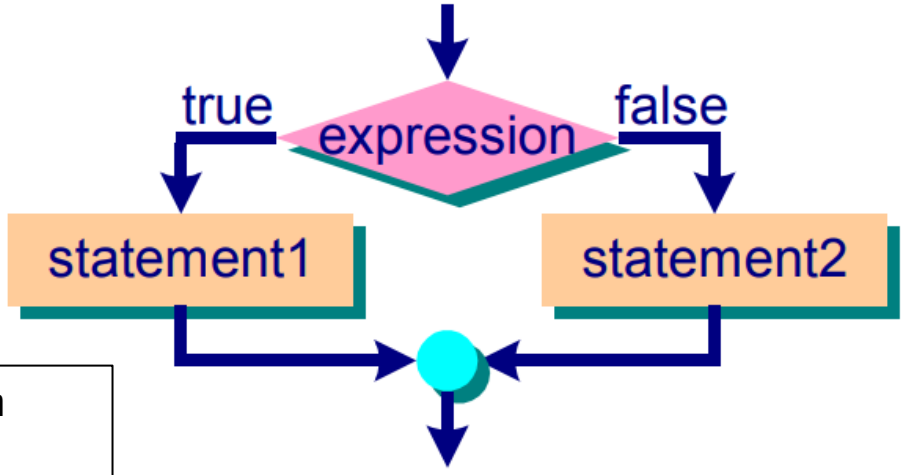
Give an integer from 1 to 10: ***7***

Your number is larger than 5.

7 is the number you entered.

The if-else Statement

```
if (expression)
    statement1;
else
    statement2;
```




```
/* This program computes the maximum
value of num1 and num2 */
#include <stdio.h>
int main(void)
{
   int num1, num2, max;
   printf("Please enter two integers:");
   scanf("%d %d", &num1, &num2);
   if (num1 > num2)
      max = num1;
   else
      max = num2;
   printf("The maximum of the two \
        is %d\n", max);
   return 0;
}
```

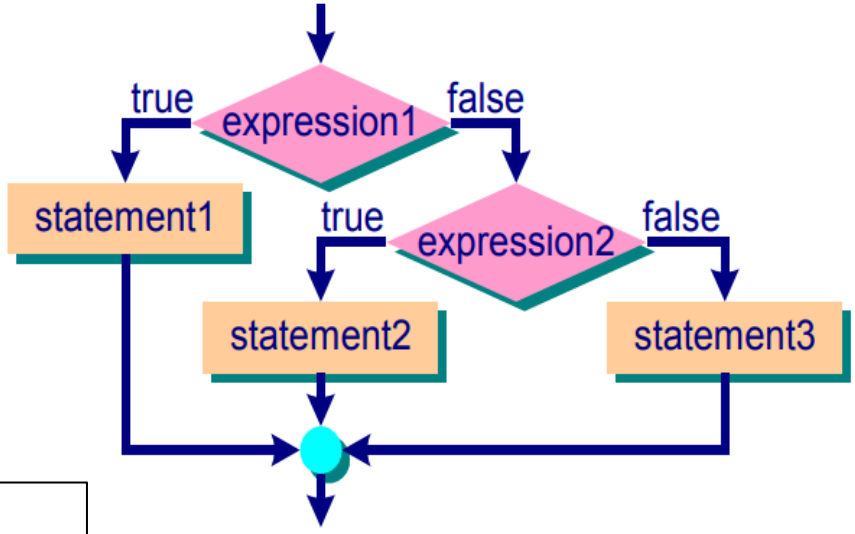
Output:

Please enter two integers: ***9 4***
The maximum of the two is 9

Please enter two integers: ***-2 0***
The maximum of the two is 0

The if…else if…else Statement

```
if (expression1)
    statement1;
else if (expression2)
    statement2;
else
    statement3;
```




```
#include <stdio.h>
int main(void)
{
   float temp;  // temperature reading.
   printf("Temperature reading:");
   scanf("%f", &temp);
   if (temp >= 100.00 && temp <= 120.0)
      printf("Temperature OK.\n");
   else if (temp < 100.0)
      printf("Temperature too low.\n");
   else
      printf("Temperature too high.\n");
   return 0;
}
```

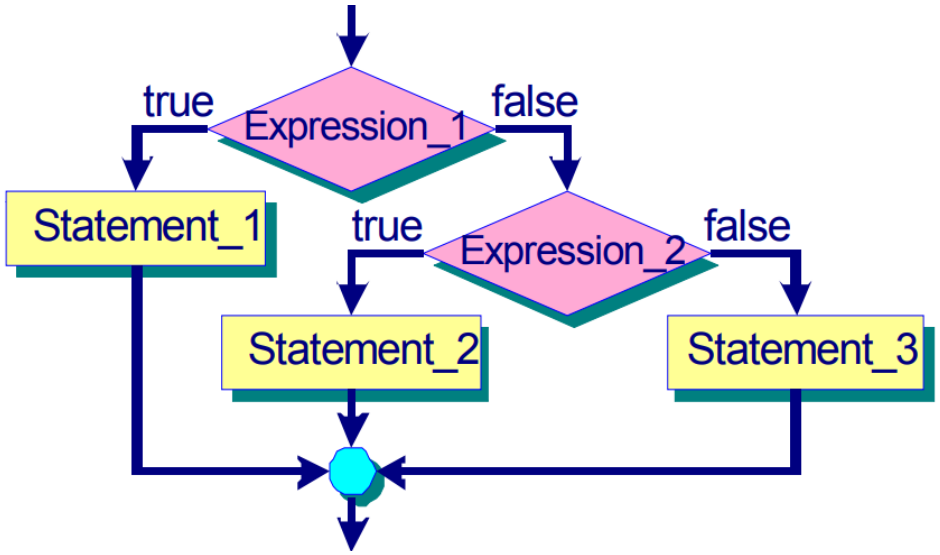
Output:

Temperature reading: ***105.0***
Temperature OK.

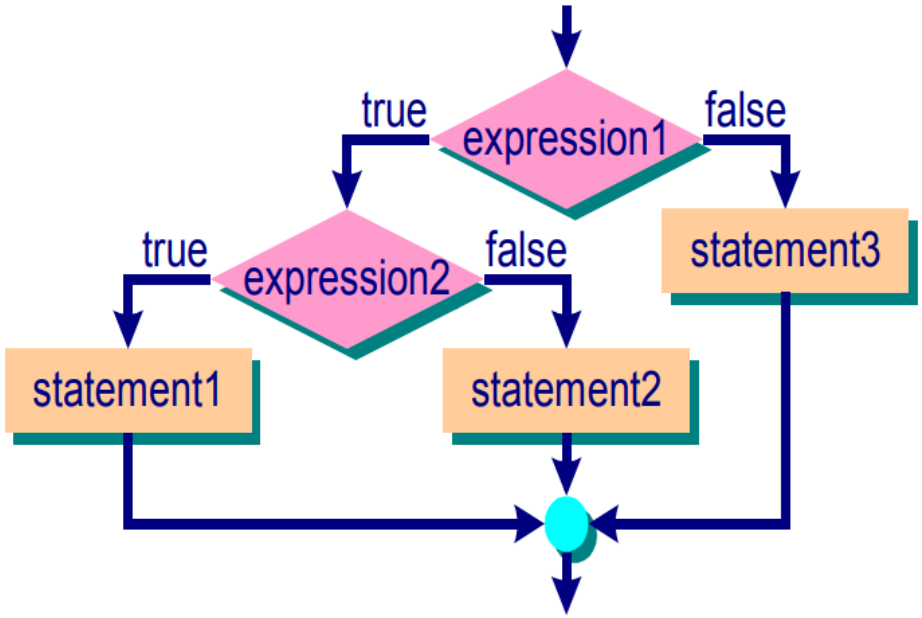
Temperature reading: ***130.0***
Temperature too high.

Nested-if

```
if (expression 1)
      statement1;
else
      if (expression2)
            statement2;
      else
            statement3;
```



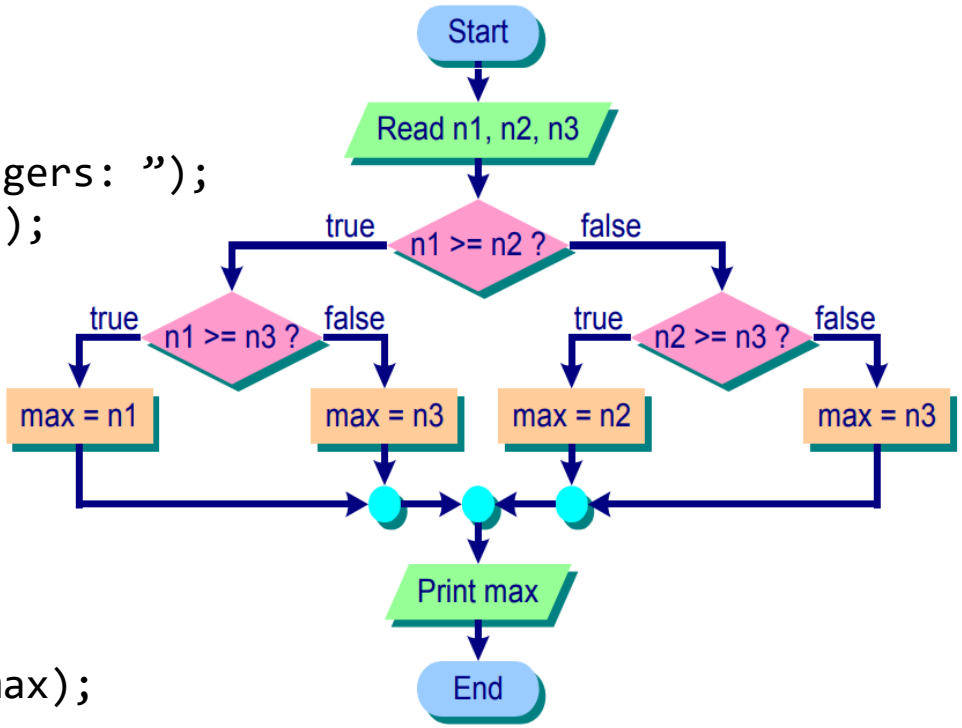

```
if (expression 1)
      if (expression2)
            statement1;
      else
            statement2;
else
      statement3;
```



Nested-if Example

```
/* This program computes the maximum value of
three numbers */
#include <stdio.h>
int main(void)
{
   int n1, n2, n3, max;
   printf("Please enter three integers: ");
   scanf("%d %d %d", &n1, &n2, &n3);
   if (n1 >= n2)
      if (n1 >= n3)
         max = n1;
      else max = n3;
   else if (n2 >= n3)
      max = n2;
   else max = n3;

   printf("The maximum is %d\n", max);
   return 0;
}
```



Output:

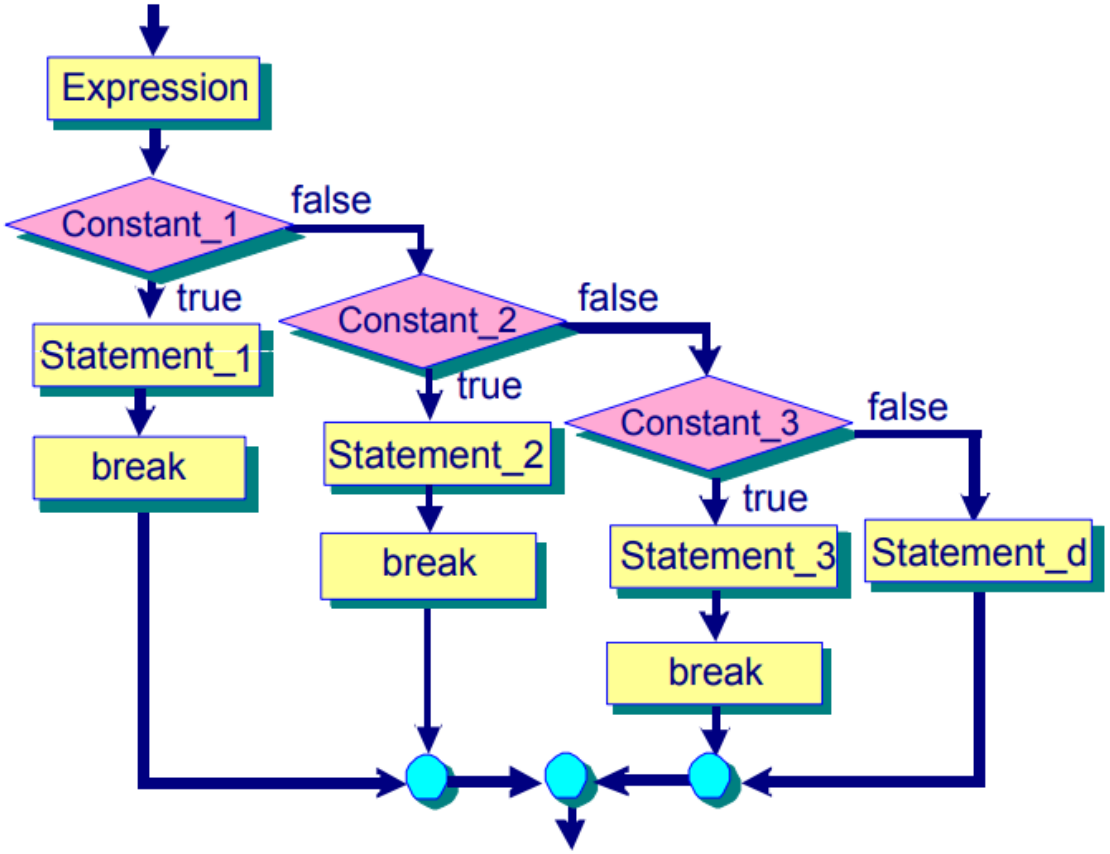
Please enter three integers: ***1 2 3***

The maximum of the three is 3

The switch Statement

The **switch** is for multi-way selection. The syntax is:

```
switch (Expression) {
      case Constant_1:
            Statement_1;
            break;
      case Constant_2:
            Statement_2;
            break;
      case Constant_3;
            Statement_3;
            break;
      default:
            Statement_d;
}
```



The switch Statement

- ***switch***, ***case***, ***break*** and ***default*** are reserved words.
- The result of ***Expression*** in ( ) must be an **integral type**.
- *Constant_1*, *Constant_2*, … are called **labels**. Each must be an integer constant, a character constant or an integer constant expression, e.g. 3, 'A', 4+'b', 5+7, etc.
- Each of the labels *Constant_1*, *Constant_2*, … must deliver a **unique integer value**. Duplicates are not allowed.
- We may also have **multiple labels** for a statement, for example, to allow both the lower and upper case selection.
- If we do not use **break** after some statements in the switch statement, execution will continue with the statements for the subsequent labels until a break statement or the end of the switch statement. This is called the **fall through** situation.

switch: Example

```
#include <stdio.h>
main(void) {
   char choice;
   int num1, num2, result;
   printf("Enter your choice (A, S or M)=> ");
   scanf("%c", &choice);
   printf("Enter two numbers:");
   scanf("%d %d", &num1, &num2);
   switch (choice) {
      case 'a':
      case 'A': result = num1 + num2;
         printf("num1 + num2 = %d\n", result);
         break;
      case 's':
      case 'S': result = num1 - num2;
         printf("num1 - num2 = %d\n", result);
         break;
      case 'm':
      case 'M': result = num1 * num2;
         printf("num1 * num2 = %d\n", result);
         break;
      default: printf("Not a proper choice.\n");
   }
   return 0;
}
```

Output:

Enter your choice (A, S or M) => ***S***

Enter two numbers: ***9 5***

9 – 5 = 4

A switch Example: Converting Score to Grade

Weighted Average Score S	Grade
90 <= S	A
80 <= S < 90	B
70 <= S < 80	C
60 <= S < 70	D
50 <= S < 60	E
S < 50	F

```
switch ((int)averageScore/10) {
  case 10: case 9:
    grade = 'A'; break;
  case 8:
    grade = 'B'; break;
  case 7:
    grade = 'C'; break;
  case 6:
    grade = 'D'; break;
  case 5:
    grade = 'E'; break;
  default: grade = 'F';
}
```

The Conditional Operator

- The conditional operator is used in the following way:

Expression_1 ? Expression_2 : Expression_3

The value of this expression depends on whether **Expression_1** is true or false.

if Expression_1 is true

=> value of the expression is that of Expression_2

else

=> value of the expression is that of Expression_3

- **Example**:

```
max = (x > y) ? x : y;
```

<==>

```
if (x > y)
    max = x;
else
    max = y;
```

Conditional Operator: Example

```
/* An example to show a conditional expression */
#include <stdio.h>
int main(void)
{
   int selection; /* User input selection */
   printf("Enter a 1 or a 0 => ");
   scanf("%d", &selection);

   selection ? printf("A one.\n") : printf("A zero.\n");
   return 0;
}
```

Output:

Enter a 1 or a 0 => ***1***

A one.

Enter a1 or a0 => ***0***

A zero.

Recap

- This lecture covers the following concepts:
  - Simple output: printf()
  - Conversion specification for formatted output
  - Simple input: scanf()
  - Character input/output: getchar() and putchar()
  - Some basic Control Flow
- Next:
  - More details about Control Flow