CS100
Introduction to Programming

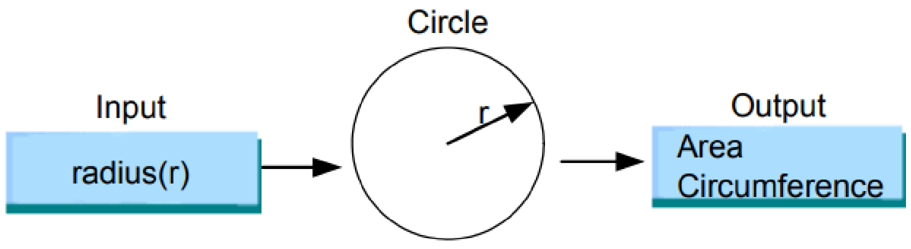
Lecture 3. Simple Input/Output

Learning Objectives

- At the end of this lecture, you will be able to use the following:
  - Simple output: **printf()**
  - Simple input: **scanf()**
  - Character input/output: **getchar()** and **putchar()**

An Example C Program

```
/*
   Purpose: To calculate the area and circumference.
   Author: S.C. Hui (NTU); Date: Jan 2012
*/
#include <stdio.h>
int main()
{
   const float PI = 3.14;
   float radius, area, circumference;
   // Read the radius of the circle
   printf("Enter the radius: ");
   scanf("%f", &radius);
   // Calculate the area
   area = PI * radius * radius;
   // Calculate the circumference
   circumference = 2 * PI * radius;
   // Print the area and circumference of the circle
   printf("The area is %0.1f\n", area);
   printf("The circumference is %0.1f", circumference);
   return 0;
}
```



Area = π*r*r

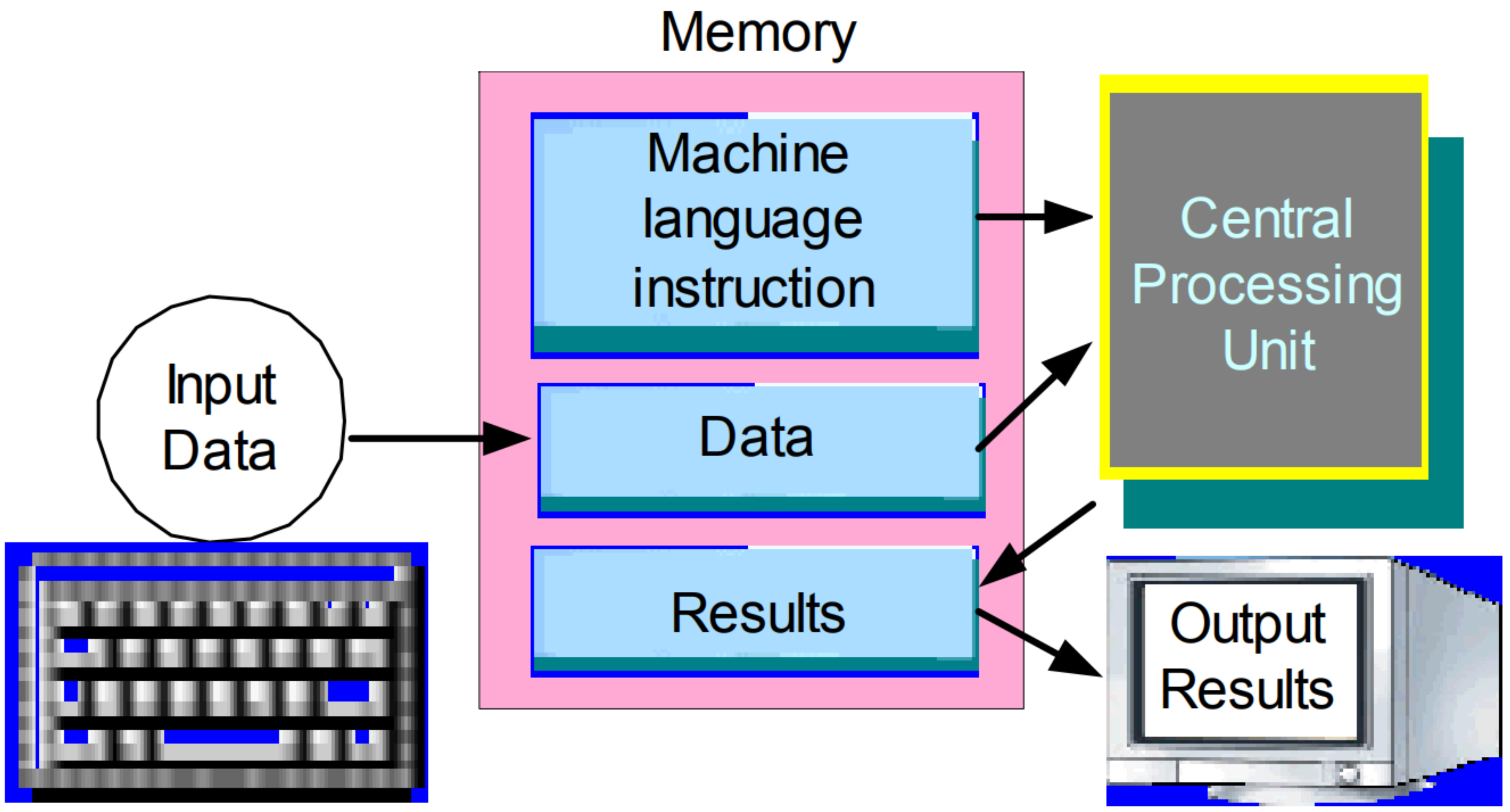
Circumference= 2*π*r

In C:

Output function: printf()

Input function: scanf()

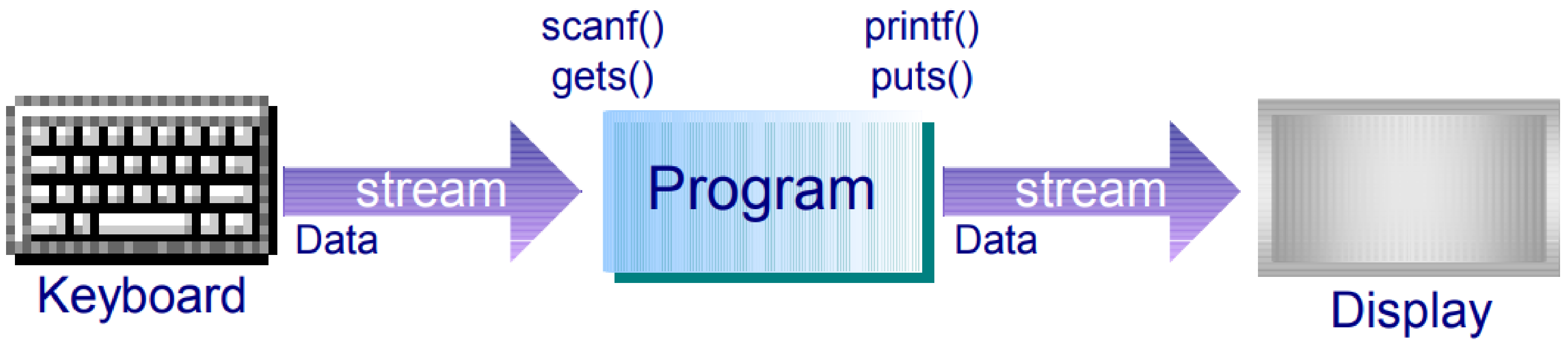
Executing Programs



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>, maths functions in the library <math>

- 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.
  - %x is a conversion specification. An item will be substituted for it in the printed output.

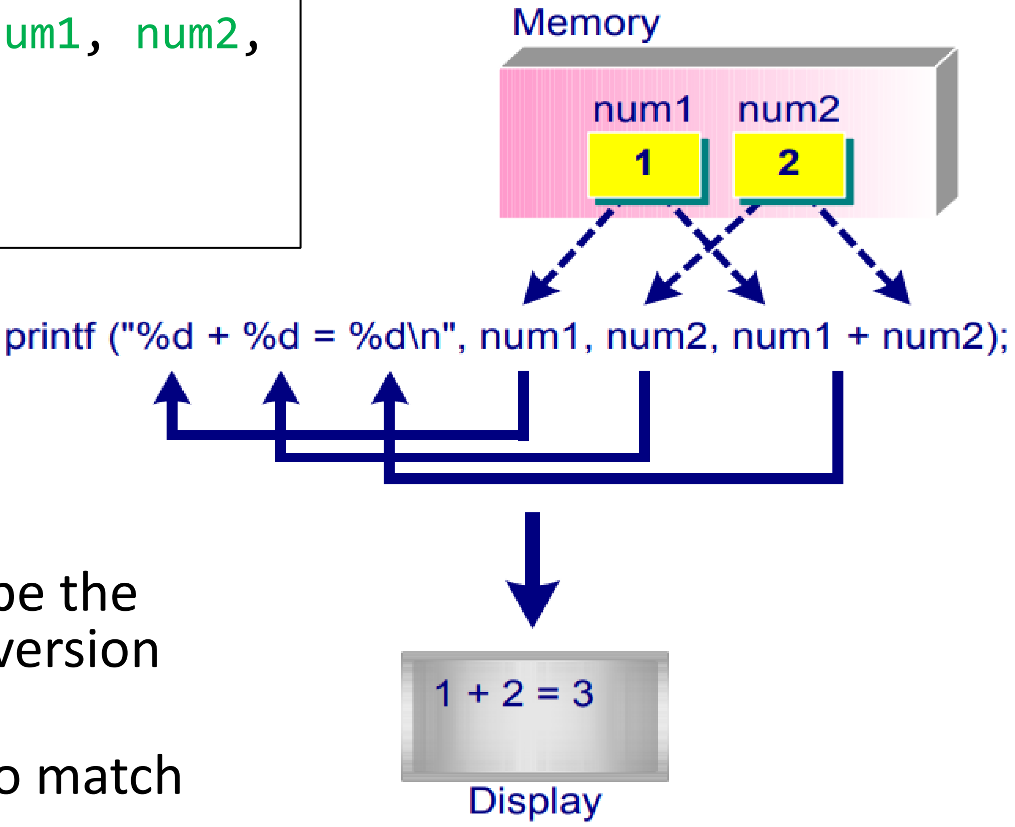
- The **argument-list** contains a list of items such as item1, item2, …, etc.
  - Values are to be substituted into places held by the conversion specification in the control string.
  - An item can be a constant, a variable or an expression like num1 + num2.

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
```

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

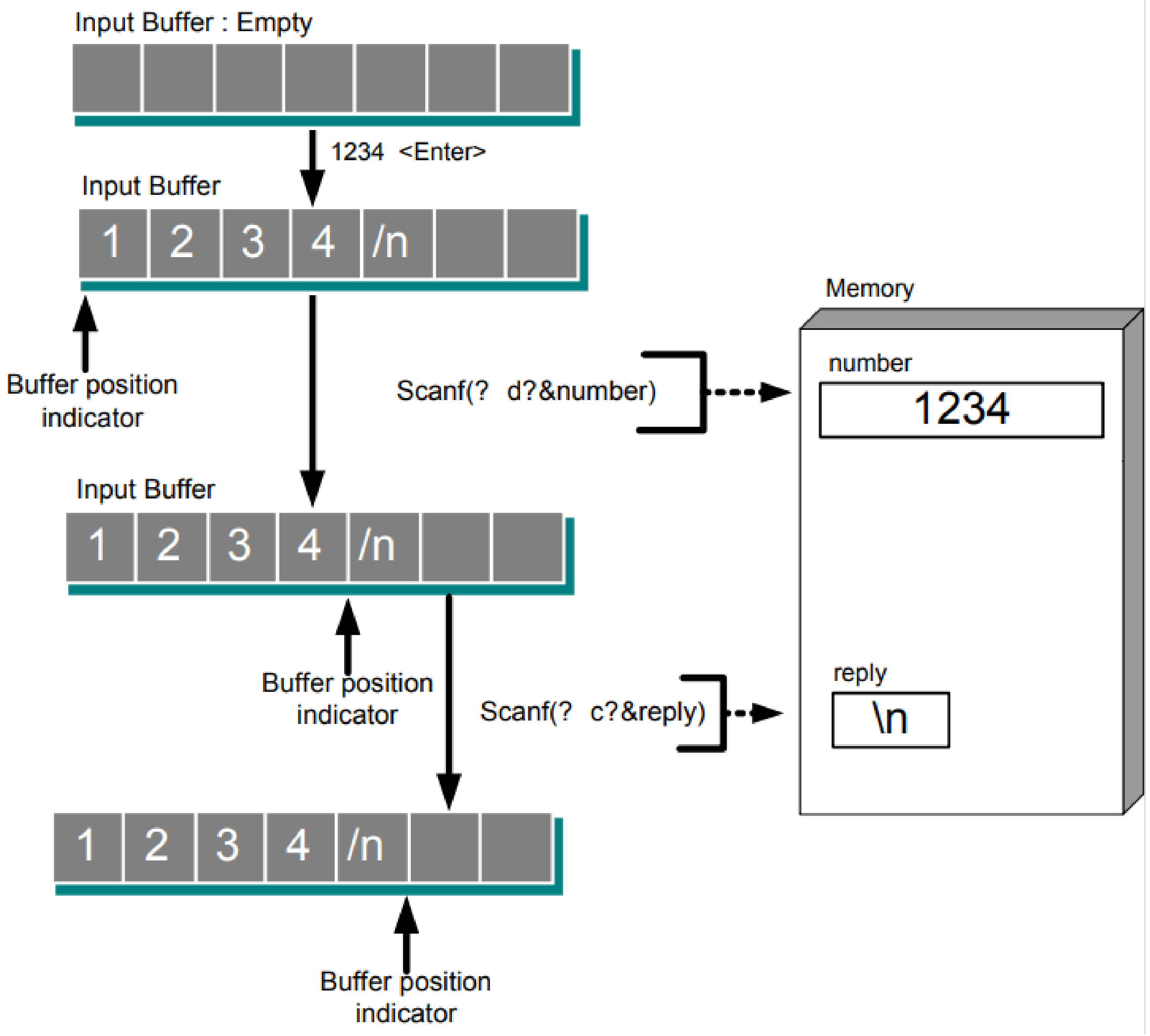
```
#include <stdio.h>
int main()
{
   int number;
   printf("Please enter a number:");
   scanf("%d", &number);
   printf("The number read is %d\n", number);
   // read in a char
   char reply;
   printf("Correct(y/n)?");
   scanf("%c", &reply);
   printf("your reply: %c\n", reply); // display char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

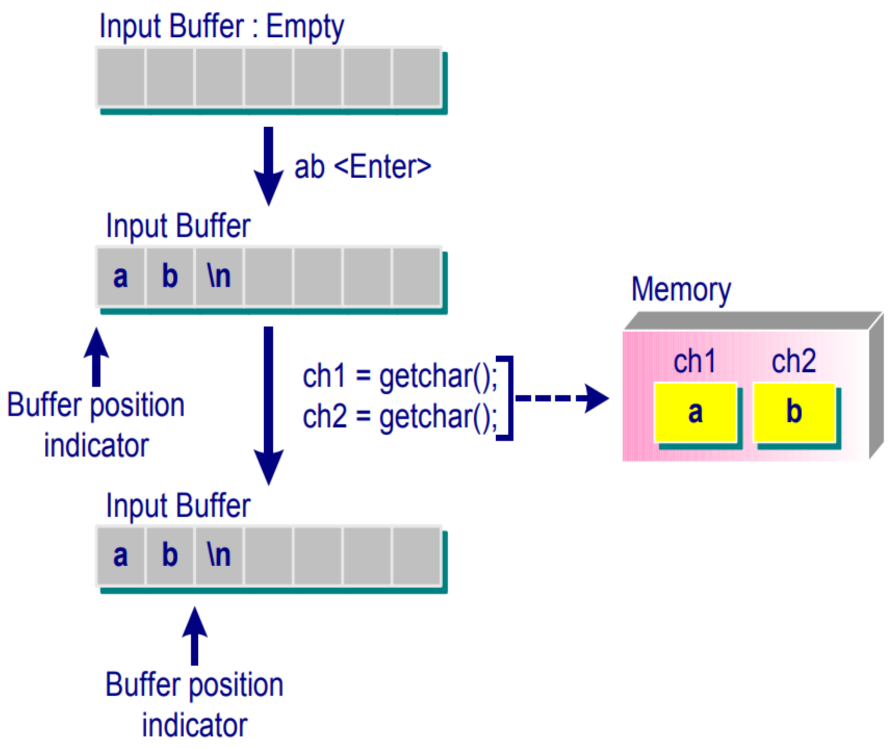
  `ch = getchar();    // ch is a character variable.`

- 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
```

Recap

- We have learned the following concepts in this lecture:
  - Simple output: printf()
  - Conversion specification for formatted output
  - Simple input: scanf()
  - Character input/output: getchar() and putchar()