C PROGRAMMING
Lecture 3

1st semester 2023-2024

Standard Input and Output

- Usually any program needs at least to print an output, some of them need also an input
- Formatted I/O: scanf, printf
- Character I/O: getchar, putchar
- Line I/O: gets, puts

Printf

- Sends output to *standard out*, the default output device can be seen as the terminal screen.
- General form

```
printf(format descriptor, var1, var2, …);
```

- format descriptor is composed of
  - Ordinary characters
    - copied directly to output
  - Special characters
    - Characters preceded by \
  - Conversion specifiers
    - Causes conversion and printing of next *argument* to printf
    - Each conversion specification begins with %

Printf

```
printf("hello world\n");
```

result: “print the string hello world followed by new line”, BUT!, according to general form, it should be

```
printf("%s\n", "hello world");
```

result: “print hello world as a *string* followed by a newline character”

```
printf("%d, %d\t%d\n", var1, var2, var3);
```

result: “print the value of the variable var1 as an integerfollowed by “,” then the value of var2 as an integer followed by a tab followed by the value of the variable var3 as an integer followed by a new line.”

```
printf("%f,  \b%f\n", var1, var2);
```

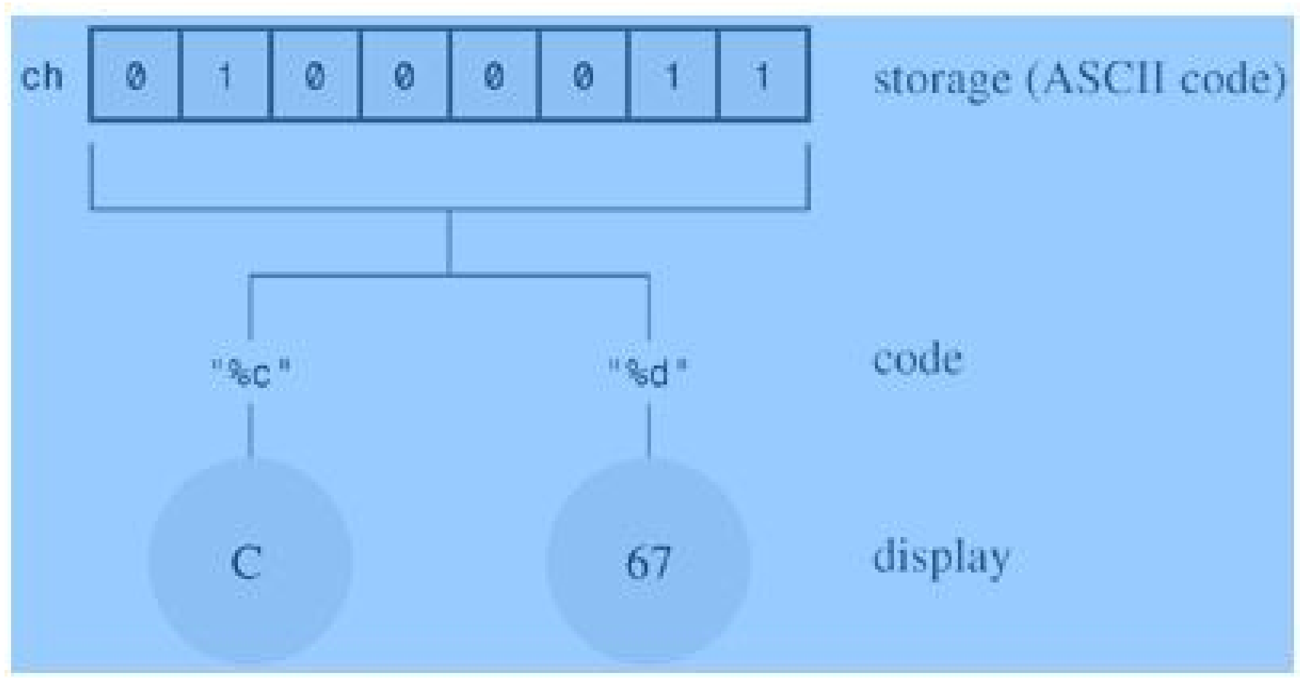
Result:?

Printf

- The format specifier in its simplest form is one of:
  - %s
    - sequence of characters known as a **String** (an *array* of chars, will be studied later)
    - Not a fundamental data type in C
  - %d
    - Decimal integer (base ten)
  - %f
    - Floating point
  - %c
    - Character

Printf Char Type

- Storage vs display values



Printf

- Special characters
  - Not wysiwyg; all begin with “\” char
    - \n new line
    - \t horizontal tab
    - \v vertical tab
    - \b backspace
    - \r carriage return
    - \c produce no further output
    - \f form feed
    - %% a single %

Printf

- Alignment and width options:
  - A minus(-) sign tells left alignment.
  - A number after % specifies the minimum field width to be printed; if the characters are less than the size of width the remaining space is filled with space and if it is greater than it printed as it is without truncation.
  - A period(.) symbol separates field width with the precision.
  - Precision tells the minimum number of digits in integer, maximum number of characters in string and number of digits after decimal part in floating value.

Printf

flag	effect
none	print normally (right justify, space fill)
-	left justify
0	leading zero fill
+	print plus on positive numbers

Printf

```
/* prints 100 in decimal, octal, and hex */
#include <stdio.h>
int main(void)
{
    int x = 100;
    printf("dec = %d; octal = %o; hex = %x\n",
  x, x, x);
    printf("dec = %d; octal = %#o; hex = %#x\n",
  x, x, x);
    return 0;
}
```

```
dec = 100; octal = 144; hex = 64
dec = 100; octal = 0144; hex = 0x64
```

Printf

```
//Fill spaces           ---
("%-3d",0)              0
("%-3d",-1)             -1
("%-3d",12345)          12345
("%-3d",-12345)         -12345

//Justify               ---
("%+3d",0)               +0
("%+3d",-1)              -1
("%+3d",12345)          +12345
("%+3d",-12345)         -12345
```

Printf

```
//Fill zeros          ---
("%03d",0)            000
("%03d",-1)           -01
("%03d",12345)        12345
("%03d",-12345)       -12345

//Invisible + Sign    ---
("% -3d",0)            0
("% -3d",-1)          -1
("% -3d",12345)        12345
("% -3d",-12345)      -12345
```

Printf

Let a floating point number be `nr=1.8765432`

```
// Precision digits
("%.0f",nr)        2
("%.0f.",nr)       2.
("%.1f",nr)    1.8
("%.6f",nr)    1.876543

// Width and precision -----
("%5.0f",nr)              2
("%5.0f.",nr)           2.
("%5.1f",nr)        1.8
("%5.6f",nr)       1.876543
```

Scanf

- reads user-typed input from *Standard input,* default input device can be seen the keyboard
- General form:
  ```
  scanf( format descriptor, &var1, &var2, ...);
  ```
- format descriptor is the same as for printf
- Blocking statement, until receives input
- Note the “&” sign (will be covered later in more depth)

double var1; scanf(“%f”,&var1);

Reading/writing chars

- The pair of functions getchar/putchar used for keyboard input/console output
- Reads/writes single char from/to standard input/output
- getchar() blocks until data is entered
- If more than 1 char entered, only first 1 is read

```
int c;
c=getchar();
putchar(c);
```

Reading/writing lines

- The pair of functions gets/puts used for keyboard input/console output
- Reads/writes lines of chars from/to standard input/output
- The way to read strings with whitespaces

```
char var_s[250];
printf("Input a string: ");
gets(var_s);
printf("The input string is: ");
puts(var_s);
```

Statements

- Represent the flow of the program
- Basic statements
  - empty statement
  - expression statements
  - sequential statements
  - iterative statements
  - selection statements
  - jump statement
- Compound statements
  - combines basic statements

Empty Statement

- Statements that contain only “;” character
- Used where the syntax needs a statement but the program doesn’t have to do something

Expression Statement

- Composed by an expression followed by “;” character:

```
    expression;
```

- Most often encountered statements
- Based on arithmetic/increment/decrement expressions, sometimes in conjunction with the assignment operator:

```
    a=a+1;
    b++;
```

Compound Statement

- Composed by grouping several statements and variable declarations
- Used where syntax requests one statement but the logic needs several actions;
- Grouping done by enclosing statements and declarations between {}:

```
{
    variables declarations;
    statements;
}
```

Flow Control Statements

- while
- if...else
- for
- do... while
- switch

While statement

- repeating a statement or group of statements until some specified condition is met
- General form:

```
while(expr){
   statement1;

   ...

   statementn;

}
```

- If expr evaluates to true (different from 0), then execute body, else go to next statement after body
- Repeat until expr is false (equals 0)

While statement

```
/* factorial of n */
#include<stdio.h>
int main(){
   int n, nfact;
   printf(“Enter a number >0:  “);
   scanf(“%d”,&n);
   while(n>0){
        nfact *= n;
        n -= 1;
   }
   printf(“Value of factorial is: %d \n”,nfact);
   return 0;
}
```

If … else statement

- General form:

```
if(expr)
   statement1
else
   statement2
```

- Else part is optional, statement can be simple or compound
- Expression is evaluated. If true (not 0), statement1 is executed and statement2 skipped (if exists). If false (0), statement1 is skipped and if else exists statement2 is executed

If … else example

```
#include <stdio.h>
int main(){
   float n1,n2;
   printf(“Enter 2 numbers:”);
   scanf(“%f %f”, &n1, &n2);
   if(n2==0)
      printf(“Divizion by 0\n”);
   else
      printf(“%6.2f divided by %6.2f is: %6.2f\n”,
  n1,n2,n1/n2);
   return 0;}
```

For statement

- Looping statement
- General form:

```
for(expr1 ; expr2 ; expr3)
          statement
```

- expr1 is called the initialization step; performed when for is to be executed
- expr2, the test/condition to control the execution of statement
- expr3, reinitialization step

For statement

- It is executed as follows:
  - expr1 is evaluated
  - expr2 is evaluated; if true, statement is executed, followed by expr3 and this step is repeated
  - if expr2 is evaluated to false, the next statement after if's statement is executed

equivalence:

```
    expr1;
        while(expr2){
              statement;
              expr3; }
```

For statement

```
/* print a multiplication table */
#include<stdio.h>
int main(){
   int type, start, end, j;
   printf("Type of table?");
   scanf("%d",&type);
   printf("start of table?");
   scanf("%d",&start);
   printf("end of table?");
   scanf("%d",&end);
   for(j=start;j<=end;j++)
      printf("\n%2d x %2d = %3d", j, type, j * 
  type);
   printf("\nEnd of program\n");
   return 0;
}
```

Do … while statement

- General form:

```
do
        statement
 while(expr)
```

- statement is executed
- expr is then evaluated; if true, repeat the above; if false, the next statement after while is executed
- !!! statement is executed at least once

Do … while statement

```
/* computes the greatest common divisor */
#include<stdio.h>
int main(){
   int m, n, r;
   do{
     printf("\nEnter two positive integers:");
     scanf("%d %d",&m, &n);
   }while(m<=0||n<=0);
   do{
     r=m%n;
     m=n;
     n=r;
   }while(r>0);
   printf("result is %d\n",m);
   return 0;
}
```

Switch statement

- Multi-way branching
- General form:

```
switch (expr) statement
```

- expr evaluates to an int; statement is almost always compound statement
- Any statement within compound statement:

```
case <constant expr>:
```

- Constant expr is an int; no 2 constants can be the same; one statement can be labeled

```
default:
```

Switch statement

- Once the control is given to a given statement as its label matches the value of expr, all statements down to the end are executed unless break or return jumps out of the switch

```
switch(ch){ /* counts lowercase vowels and
  nonvowels
   case 'a':
   case 'e':
   case 'i':
   case 'o':
   case 'u': vowels ++;
                break;
   default: nonvowels++;
                break;  /* not needed, just for
  clarity */
}
```

Exercises

- Write a program that reads a positive integer and determines:
  - Whether is even or odd
  - Whether is prime or not
  - Whether is perfect square