Structured Programming

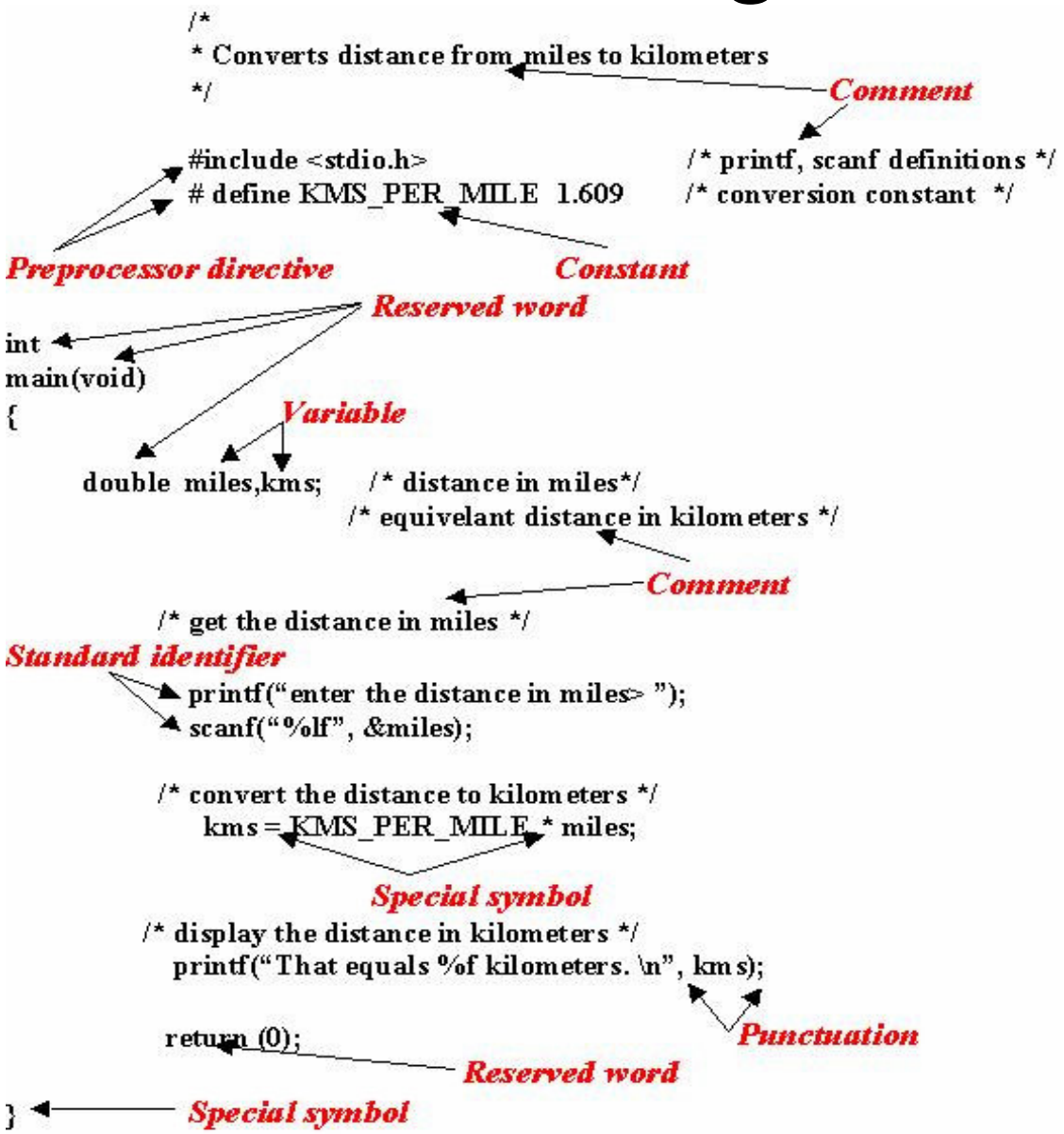
Dr. Mohamed Khedr

Lecture 3

http://webmail.aast.edu/~khedr

Example for Another C Program

- Write a program to Convert given distance in miles to kilometers.



C Language Elements
Preprocessor Directives

Examples

```
#define KMS_PER_MILE  1.609
# define AT  '@'
# define VOTING_AGE   18
```

- **Define is a preprocessor directive.**
- **Valid constant declarations**
- **A named constant is a location in memory that we can refer to by a name, and in which a data value that cannot be changed is stored.**
- **This directives instructs the processor to replace each occurrence of KMS_PER_MILE in the text of the C program by 1.609 before compilation begins.**

```c
/* Fig. 2.5: fig02_05.c
   Addition program */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   int integer1; /* first number to be input by user  */
   int integer2; /* second number to be input by user */
   int sum;      /* variable in which sum will be stored */

   printf( "Enter first integer\n" ); /* prompt */
   scanf( "%d", &integer1 );          /* read an integer */

   printf( "Enter second integer\n" ); /* prompt */
   scanf( "%d", &integer2 );           /* read an integer */

   sum = integer1 + integer2; /* assign total to sum */

   printf( "Sum is %d\n", sum ); /* print sum */

   return 0;  /* indicate that program ended successfully */

} /* end function main */
```

```
Enter first integer
45
Enter second integer
72
Sum is 117
```

Definitions of variables

scanf obtains a value from the user and assigns it to **integer1**

scanf obtains a value from the user and assigns it to **integer2**

Assigns a value to **sum**

```c
/* Fig. 2.13: fig02_13.c
   Using if statements, relational
   operators, and equality operators */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   int num1; /* first number to be read from user  */
   int num2; /* second number to be read from user */

   printf( "Enter two integers, and I will tell you\n" );
   printf( "the relationships they satisfy: " );

   scanf( "%d%d", &num1, &num2 ); /* read two integers */

   if ( num1 == num2 ) {
      printf( "%d is equal to %d\n", num1, num2 );
   } /* end if */

   if ( num1 != num2 ) {
      printf( "%d is not equal to %d\n", num1, num2 );
   } /* end if */

   if ( num1 < num2 ) {
      printf( "%d is less than %d\n", num1, num2 );
   } /* end if */

```

Checks if **num1** is equal to **num2**

Checks if **num1** is not equal to **num2**

Checks if **num1** is less than **num2**

```
29    if ( num1 > num2 ) {
30       printf( "%d is greater than %d\n", num1, num2 );
31    } /* end if */
32
33    if ( num1 <= num2 ) {
34       printf( "%d is less than or equal to %d\n", num1, num2 );
35    } /* end if */
36
37    if ( num1 >= num2 ) {
38       printf( "%d is greater than or equal to %d\n", num1, num2 );
39    } /* end if */
40
41    return 0;   /* indicate that program ended successfully */
42
43 } /* end function main */
43 } /* end function main */
```

Checks if **num1** is greater than **num2**

Checks if **num1** is less than or equal to **num2**

Checks if **num1** is greater than equal to **num2**

```
Enter two integers, and I will tell you
the relationships they satisfy: 3 7
3 is not equal to 7
3 is less than 7
```

(continued from previous slide…)

```
Enter two integers, and I will tell you
the relationships they satisfy:
22 is not equal to 12
22 is greater than 12
22 is greater than or equal to 12
```

```
Enter two integers, and I will tell you
the relationships they satisfy:
7 is equal to 7
7 is less than or equal to 7
7 is greater than or equal to 7
```

Conversion specifier	Description
`d`	Display as a signed decimal integer.
`i`	Display as a signed decimal integer. [*Note:* The `i` and `d` specifiers are different when used with `scanf`.]
`o`	Display as an unsigned octal integer.
`u`	Display as an unsigned decimal integer.
`x` or `X`	Display as an unsigned hexadecimal integer. `X` causes the digits `0-9` and the letters `A-F` to be displayed and `x` causes the digits `0-9` and `a-f` to be displayed.
`h` or `l` (letter `l`)	Place before any integer conversion specifier to indicate that a `short` or `long` integer is displayed, respectively. Letters `h` and `l` are more precisely called **length modifiers**.

Conversion specifier	Description
`e` or `E`	Display a floating-point value in exponential notation.
`f`	Display floating-point values in fixed-point notation.
`g` or `G`	Display a floating-point value in either the floating-point form `f` or the exponential form `e` (or `E`), based on the magnitude of the value.
`L`	Place before any floating-point conversion specifier to indicate that a `long double` floating-point value is displayed.

Printing Strings and Characters

- `c`
  - Prints `char` argument
  - Cannot be used to print the first character of a string
- `s`
  - Requires a pointer to `char` as an argument
  - Prints characters until `NULL` (`'\0'`) encountered
  - Cannot print a `char` argument
- Remember
  - Single quotes for character constants (`'z'`)
  - Double quotes for strings `"z"` (which actually contains two characters, `'z'` and `'\0'`)
- `p`
  - Displays pointer value (address)
- `n`
  - Stores number of characters already output by current `printf` statement
  - Takes a pointer to an integer as an argument
  - Nothing printed by a `%n` specification
  - Every `printf` call returns a value
    - Number of characters output
    - Negative number if error occurs
- `%`
  - Prints a percent sign
  - %%

```c
#include <stdio.h>

int main( void )
{
   int *ptr;      /* define pointer to int */
   int x = 12345; /* initialize int x */
   int y;         /* define int y */

   ptr = &x;      /* assign address of x to ptr */
   printf( "The value of ptr is %p\n", ptr );
   printf( "The address of x is %p\n\n", &x );

   printf( "Total characters printed on this line:%n", &y );
   printf( " %d\n\n", y );

   y = printf( "This line has 28 characters\n%n" );
   printf( "%d characters were printed\n\n", y );

   printf( "Printing a %% in a format control string\n" );

   return 0; /* indicates successful termination */

} /* end main */
```

p specifies a memory address will be printed

n stores the number of characters printed on a line

% prints a percent sign

```
The value of ptr is 0012FF78
The address of x is 0012FF78

Total characters printed on this line: 38

This line has 28 characters
28 characters were printed

Printing a % in a format control string
```

9.8 Printing with Field Widths and Precision

- Field width
  - Size of field in which data is printed
  - If width larger than data, default right justified
    - If field width too small, increases to fit data
    - Minus sign uses one character position in field
  - Integer width inserted between % and conversion specifier
  - %4d – field width of 4

```c
/* Fig 9.8: fig09_08.c */
/* Printing integers right-justified */
#include <stdio.h>

int main( void )
{
   printf( "%4d\n", 1 );
   printf( "%4d\n", 12 );
   printf( "%4d\n", 123 );
   printf( "%4d\n", 1234 );
   printf( "%4d\n\n", 12345 );

   printf( "%4d\n", -1 );
   printf( "%4d\n", -12 );
   printf( "%4d\n", -123 );
   printf( "%4d\n", -1234 );
   printf( "%4d\n", -12345 );

   return 0; /* indicates successful termination */

} /* end main */
```

A field width of 4 will make C attempt to print the number in a 4-character space

Note that C considers the minus sign a character

```
   1
  12
 123
1234
12345

  -1
 -12
-123
-1234
-12345
```

9.8 Printing with Field Widths and Precision

- Precision
  - Meaning varies depending on data type
  - Integers (default 1)
    - Minimum number of digits to print
      - If data too small, prefixed with zeros
  - Floating point
    - Number of digits to appear after decimal (`e` and `f`)
      - For `g` – maximum number of significant digits
  - Strings
    - Maximum number of characters to be written from string
  - Format
    - Use a dot (`.`) then precision number after %

```
%.3f
```

```c
/* Fig 9.9: fig09_09.c */
/* Using precision while printing integers,
   floating-point numbers, and strings */
#include <stdio.h>

int main( void )
{
   int i = 873;                  /* initialize int i */
   double f = 123.94536;         /* initialize double f */
   char s[] = "Happy Birthday"; /* initialize char array s */

   printf( "Using precision for integers\n" );
   printf( "\t%.4d\n\t%.9d\n\n", i, i );

   printf( "Using precision for floating-point numbers\n" );
   printf( "\t%.3f\n\t%.3e\n\t%.3g\n\n", f, f, f );
```

Precision for integers specifies the minimum number of characters to be printed

Precision for the **g** specifier controls the maximum number of significant digits printed

Precision for **f** and **e** specifiers controls the number of digits after the decimal point

```
17
18    printf( "Using precision for strings\n" );
19    printf( "\t%.11s\n", s );
20
21    return 0; /* indicates successful termination */
22
23 } /* end main */
```

Precision for strings specifies the maximum number of characters to be printed

```
Using precision for integers
        0873
        000000873

Using precision for floating-point numbers
        123.945
        1.239e+002
        124

Using precision for strings
        Happy Birth
```

Escape sequence	Description
`\'` (single quote)	Output the single quote (') character.
`\"` (double quote)	Output the double quote (") character.
`\?` (question mark)	Output the question mark (?) character.
`\\` (backslash)	Output the backslash (\) character.
`\a` (alert or bell)	Cause an audible (bell) or visual alert.
`\b` (backspace)	Move the cursor back one position on the current line.
`\f` (new page or form feed)	Move the cursor to the start of the next logical page.
`\n` (newline)	Move the cursor to the beginning of the next line.
`\r` (carriage return)	Move the cursor to the beginning of the current line.
`\t` (horizontal tab)	Move the cursor to the next horizontal tab position.
`\v` (vertical tab)	Move the cursor to the next vertical tab position.

Fig. 9.16 | **Escape sequences.**

```c
/* Fig 9.18: fig09_18.c */
/* Reading integers */
#include <stdio.h>

int main( void )
{
   int a;
   int b;
   int c;
   int d;
   int e;
   int f;
   int g;

   printf( "Enter seven integers: " );
   scanf( "%d%i%i%i%o%u%x", &a, &b, &c, &d, &e, &f, &g );

   printf( "The input displayed as decimal integers is:\n" );
   printf( "%d %d %d %d %d %d %d\n", a, b, c, d, e, f, g );

   return 0; /* indicates successful termination */

} /* end main */
```

d specifies a decimal integer will be input

i specifies an integer will be input

o specifies an octal integer will be input

u specifies an unsigned decimal integer will be input

x specifies a hexadecimal integer will be input

```
Enter seven integers: -70 -70 070 0x70 70 70 70
The input displayed as decimal integers is:
-70 -70 56 112 56 70 112
```

```c
/* Fig 9.19: fig09_19.c */
/* Reading floating-point numbers */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   double a;
   double b;
   double c;

   printf( "Enter three floating-point numbers: \n" );
   scanf( "%le%lf%lg", &a, &b, &c );

   printf( "Here are the numbers entered in plain\n" );
   printf( "floating-point notation:\n" );
   printf( "%f\n%f\n%f\n", a, b, c );

   return 0; /* indicates successful termination */

} /* end main */
```

e, **f**, and **g** specify a floating-point number will be input

l specifies a **double** or **long double** will be input

```
Enter three floating-point numbers:
1.27987 1.27987e+03 3.38476e-06
Here are the numbers entered in plain
floating-point notation:
1.279870
1279.870000
0.000003
```

```c
/* Fig 9.20: fig09_20.c */
/* Reading characters and strings */
#include <stdio.h>

int main( void )
{
   char x;
   char y[ 9 ];

   printf( "Enter a string: " );
   scanf( "%c%s", &x, y );

   printf( "The input was:\n" );
   printf( "the character \"%c\" ", x );
   printf( "and the string \"%s\"\n", y );

   return 0; /* indicates successful termination */

} /* end main */
```

c specifies a character will be input

s specifies a string will be input

```
Enter a string: Sunday
The input was:
the character "S" and the string "unday"
```

```c
/* Fig 9.21: fig09_21.c */
/* Using a scan set */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   char z[ 9 ]; /* define array z */

   printf( "Enter string: " );
   scanf( "%[aeiou]", z ); /* search for set of characters */

   printf( "The input was \"%s\"\n", z );

   return 0; /* indicates successful termination */

} /* end main */
```

```
Enter string: ooeeooahah
The input was "ooeeooa"
```

`[]` specifies only the initial segment of a string that contains the characters in brackets will be read

```c
/* Fig 9.22: fig09_22.c */
/* Using an inverted scan set */
#include <stdio.h>

int main( void )
{
   char z[ 9 ];

   printf( "Enter a string: " );
   scanf( "%[^aeiou]", z ); /* inverted scan set */

   printf( "The input was \"%s\"\n", z );

   return 0; /* indicates successful termination */

} /* end main */
```

```
Enter a string: String
The input was "Str"
```

[] and **^** specify only the initial segment of a string that does **not** contain the characters in brackets will be read

```c
/* Fig 9.23: fig09_23.c */
/* inputting data with a field width */
#include <stdio.h>

int main( void )
{
   int x;
   int y;

   printf( "Enter a six digit integer: " );
   scanf( "%2d%d", &x, &y );

   printf( "The integers input were %d and %d\n", x, y );

   return 0; /* indicates successful termination */

} /* end main */
```

A field width of 2 tells C to only read the first 2 characters of that input

```
Enter a six digit integer: 123456
The integers input were 12 and 3456
```

```c
/* Fig 9.24: fig09_24.c */
/* Reading and discarding characters from the input stream */
#include <stdio.h>

int main( void )
{
   int month1;
   int day1;
   int year1;
   int month2;
   int day2;
   int year2;

   printf( "Enter a date in the form mm-dd-yyyy: " );
   scanf( "%d%*c%d%*c%d", &month1, &day1, &year1 );

   printf( "month = %d  day = %d  year = %d\n\n", month1, day1, year1 );

   printf( "Enter a date in the form mm/dd/yyyy: " );
   scanf( "%d%*c%d%*c%d", &month2, &day2, &year2 );

   printf( "month = %d  day = %d  year = %d\n", month2, day2, year2 );

   return 0; /* indicates successful termination */

} /* end main */
```

***** is a wildcard—**scanf** will disregard anything between the two inputs on either side of it

```
Enter a date in the form mm-dd-yyyy: 11-18-2003
month = 11  day = 18  year = 2003

Enter a date in the form mm/dd/yyyy: 11/18/2003
month = 11  day = 18  year = 2003
```

Conditional (ternary) Operator

- Syntax

```
expr1 ? expr2 : expr3
```

  - If **expr1** ≠ 0, then execute **expr2** and ignore **expr3**
  - If **expr1** = 0, then execute **expr3** and ignore **expr2**

Example: `x = i+j ? i+1 : j+1`

Example:

```
x = 5 ? 4 : 2;          /* x = 4 */
```

Example:

```
j = 4;
i = 2
x = i+j ? i+1 : j-1     /* x = 3 */
```

Example:

```
l = a > b ? a : b;      /* the larger of a and b */
```

Example:

```
max =(a > b)?((a>c)?a:c):(b>c)?b:c);
  /* the maximum number among a, b, and c */
```

Example:

```
x = a > 0 ? a: -a;      /* the absolute value of a */
```

sizeof Operator

- Syntax

  `sizeof(expr)`

  – The number of bytes occupied by **`expr`**

  – For most computers

  sizeof(3) 2 or 4 (bytes)

  (depending on16 bit CPU or 32 bit CPU), where 3 is an integer

  sizeof(3L) 4 (long int)

  sizeof(3.0) 8 (double float)

  Example:

```
double i;
printf("%d",sizeof(i));      8
```

  – Usually, this operator is used to get the size of an organized variable (like **`struct`**, **`union`**, …)

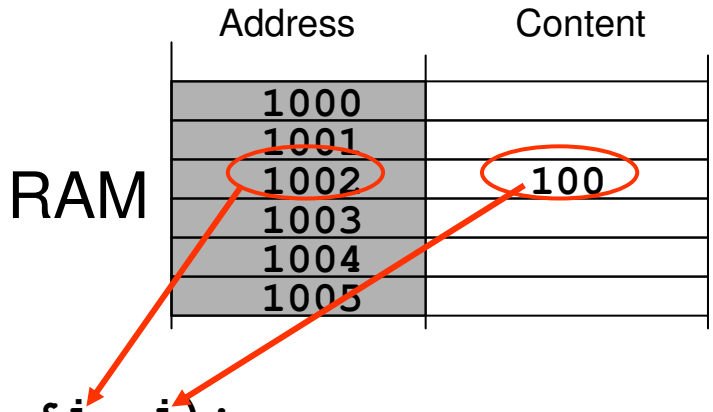
  – This is one of a few functions that are *built-in*. No #include is required.

Address Operator

- Syntax

`&var`

  - Get the address of the variable
  - **`&`** means the address of **`var`**
  - Type of **`var`** may be
    (a) fundamental data type
    (b) organized data type



Example:

```
int i=100;
printf("%d %d", &i, i);
```

Assignment Operators

- Syntax:

  `var = expression;`

  – Assign the value of expression to variable (**`var`**)

  Example:

```
int x, y, z;
 x = 5;
 y = 7;                ⇒   Z = (x = 5) + (y = 7)   much faster
 z = x + y;
```

```
int x, y, z;
 x = y = z = 0;        ⇒ same as   x = (y = (z = 0));
```

```
int x = y = z = 0;     ⇒ wrong !   int x = 0, y = 0, z = 0;
```

```
int i, j;
float f, g;
 i = f = 2.5;          ⇒ i = 2;      f = 2.5;
 g = j = 3.5;          ⇒ g = 3.0;    j = 3;
```

Short Hand Assignment

- Syntax

  `f = f op g` can be rewritten to be `f op= g`

  such as: `a = a + 2` ⇒ `a += 2`, `a = a - 2` ⇒ `a -= 2`, `a = a * 2` ⇒ `a *= 2`,
  `a = a / 2` ⇒ `a /= 2`, `a = a % 2` ⇒ `a %= 2`, `a = a << 2` ⇒ `a <<= 2`,
  `a = a & 2` ⇒ `a &= 2`, `a = a | 2` ⇒ `a |= 2`, `a = a ^ 2` ⇒ `a ^= 2`

  - No blanks between **op** and **=**
  - `x *= y + 1` is actually `x = x * (y+1)` rather than `x = x * y + 1`

    Example:

    ```
    q = q / (q+2)  ⇒  q /= q+2
    j = j << 2     ⇒  j <<= 2
    ```

  - Advantage: help compiler to produce more efficient code

    More complicated examples:

    ```
    int a=1, b=2, c=3, x=4, y=5;
    a += b += c *= x + y - 6;
    printf("%d %d %d %d\n",a,b,c,x,y); /* result is 12 11 9 4 5 */
    a += 5 + b += c += 2 + x + y;      /* wrong */
    a += 5 + (b+= c += 2 + x + y);     /* result is 22 16 14 4 5 */
    ```

Increment / Decrement Operators

++ (increment) -- (decrement)

- Prefix Operator
  - Before the variable, such as **++n** or **--n**
  - Increments or decrements the variable before using the variable
- Postfix Operator
  - After the variable, such as **n++** or **n--**
  - Increments or decrements the variable after using the variable

- `++n`
  1. Increment **n**
  2. Get value of **n** in expression
- `--n`
  1. Decrement **n**
  2. Get value of **n** in expression
- `n++`
  1. Get value of **n** in expression
  2. Increment **n**
- `n--`
  1. Get value of **n** in expression
  2. Decrement **n**

Increment / Decrement Operators (cont.)

– Simple cases

```
++i;
i++;      (i = i + 1; or i += 1;)
--i;
i--;      (i = i - 1; or i -= 1;)
```

Example:

```
i = 5;
i++; (or ++i;) ⇒ 6

i = 5;
i--; (or --i;)
printf("%d", i) ⇒ 4
```

– Complicated cases

```
i = 5;
j = 5 + ++i;
```

i = 6, j = 11

```
i = 5;
j = 5 + i++;
```

i = 6, j = 10

```
i = 5;
j = 5 + --i;
```

i = 4, j = 9

```
i = 5;
j = 5 + i--;
```

i = 4, j = 10

Increment / Decrement Operators (cont.)

- Invalid cases

```
++3          3++           --3          3--
++(x+y+z)    (x+y+z)++     --(x+y+z)    (x+y+z)--
++x++        --x--         ++x--        --x++
```

Note: Can not increment or decrement constant and expression

```
i ++j        or      i --j (WRONG)
i + ++j      i + --j       i - --j   i - ++j      (OK)
```

Other Input / Output

`puts(line)` Print a string to standard output and append a newline

Example:
```
puts("12345");
```

`putchar(c)` Print a character to standard output

Example:
```
putchar('A');
```

`gets(line)` Read a string from standard input (until a newline is entered)

Example:
```
char buf[128];
gets(buf);  /* space is OK, and the '\n' won't be read in */
```

- Newline will be replaced by '\0'

`getchar()` Get a character from standard input

Example:
```
int c;
c = getchar();    /* c must be int */
```

- In-memory Format Conversion

```
sprintf(string, control, variables);
```