Programming Languages -1 (Introduction to C)

data types, operators, io, control structures

Instructor: M.Fatih AMASYALI

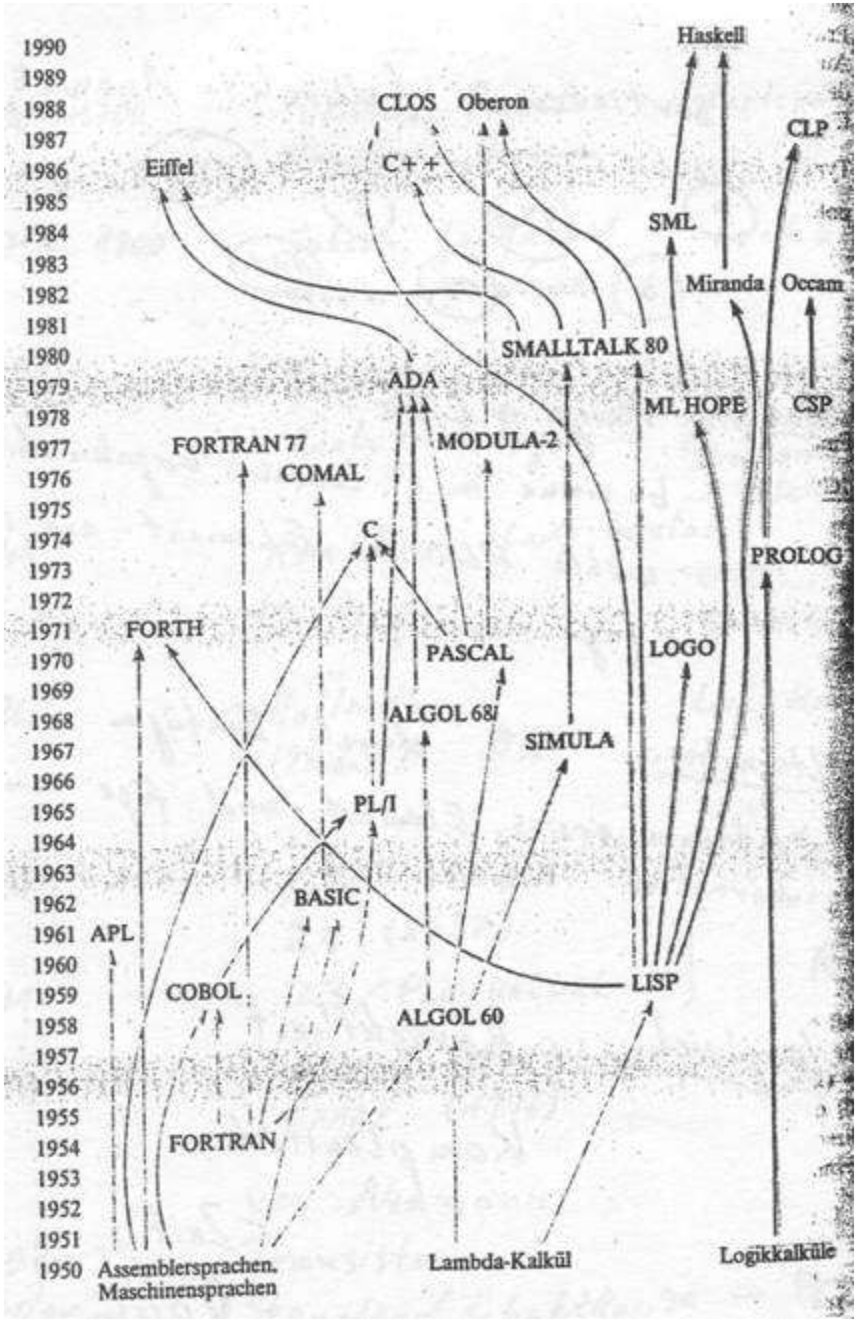
E-mail:mfatih@ce.yildiz.edu.tr

Course Details

- Textbook:
  - Kaan Aslan, A’dan Z’ye C Klavuzu
- Compiler:
  - Dev C++

    http://www.bloodshed.net/dev/devcpp.html

Family Tree of Programming Languages



Why learn C?

- Good starting point for learning other languages
  - Subset of C++, similar to Java
- Closeness to machine allows one to learn about system-level details
- Portable – compilers available for most any platform!
- Very fast (almost as fast as assembly)
  - C/C++ are languages of choice for most programmers

“first C program”

```
#include <stdio.h>
#include <conio.h>

int main()
{
      printf( “Hello, world!\n” );
      getch();
}
```

- All programs run from the ‘main’ function
- ‘printf’ is a function in the library “stdio.h”
- To include library functions use “#include”
  - – All programs use library functions

That wasn’t too hard, let’s try another!

```
#include <stdio.h>

int main()
{
      int x = 1, y;
      int sum;
      y = 3;
      sum = x + y; /* adds x to y, places value
                      in variable sum */
      printf( “%d plus %d is %d\n”, x, y, sum );
}
```

Comments

- Any string of symbols placed between the delimiters /* and */.
- Can span multiple lines
- Can’t be nested! Be careful.
- /* /* /* Hi */ is an example of a comment.
- /* Hi */ */ is going to generate a parse error

Keywords

- Reserved words that cannot be used as variable names
- OK within comments . . .
- Examples: *break*, *if*, *else*, *do*, *for*, *while*, *int*, *void*

Identifiers

- Used to give names to variables, functions, etc.
- A “token” (“word”) composed of a sequence of letters, digits, and underscore (“_”) character. (NO spaces.)
  - First character cannot be a digit
  - C is case sensitive, so beware (e.g. printf≠Printf)
- Identifiers such as “printf” normally would not be redefined; be careful
- Only the first 31 characters matter

Constants

0, 77, 3.14 examples.

- Strings: double quotes. “Hello”
- Characters: single quotes. ‘a’ , ‘z’
- Have types implicitly associated with them…

Fundamental Data Type

–char is an 8 bit (=1 byte) number

Data Type		Abbreviation	Size (byte)	Range
char	**`char`**		1	-128 ~ 127
	`unsigned char`		1	0 ~ 255
int	**`int`**		2 or 4	$-2^{15} \sim 2^{15}-1$ or $-2^{31} \sim 2^{31}-1$
	`unsigned int`	`unsigned`	2 or 4	$0 \sim 65535$ or $0 \sim 2^{32}-1$
	`short int`	`short`	2	-32768 ~ 32767
	`unsigned short int`	`unsigned short`	2	0 ~ 65535
	`long int`	`long`	4	$-2^{31} \sim 2^{31}-1$
	`unsigned long int`	`unsigned long`	4	$0 \sim 2^{32}-1$
`float`			4	
`double`			8	

Note: $2^7 = 128$, $2^{15} = 32768$, $2^{31} = 2147483648$

Complex are not available

No boolean types

Use 0=False and anything else(usually 1)=True

- Character literal
  - American Standard Code for Information Interchange (ASCII)
  - Printable:

single space	32
‘0’ - ‘9’	48 - 57
‘A’ - ‘Z’	65 - 90
‘a’ - ‘z’	97 - 122

  - Nonprintable and special meaning chars

`\n’	new line	10	`\t’	tab	9
`\\’	back slash	9	`\’’	single quote	39
`\0’	null	0	`\b’	back space	8
`\f’	formfeed	12	’\r’	carriage return	13
`\”’	double quote	34			

- String Literal
  - will be covered in Array section
  - String is a array of chars but ended by **`'\0'`**
  - String literal is allocated in a continuous memory space of Data Segment, so it can not be rewritten

Example: **"ABCD"**

A	**B**	C	D	'\0'	…

4 chars but takes 5 byte spaces in memory

Question: **"I am a string"** takes ? Bytes

Ans: 13+1 = 14 bytes

- Character literals & ASCII codes:

```
char x;
x='a';    /* x = 97*/
```

Notes:

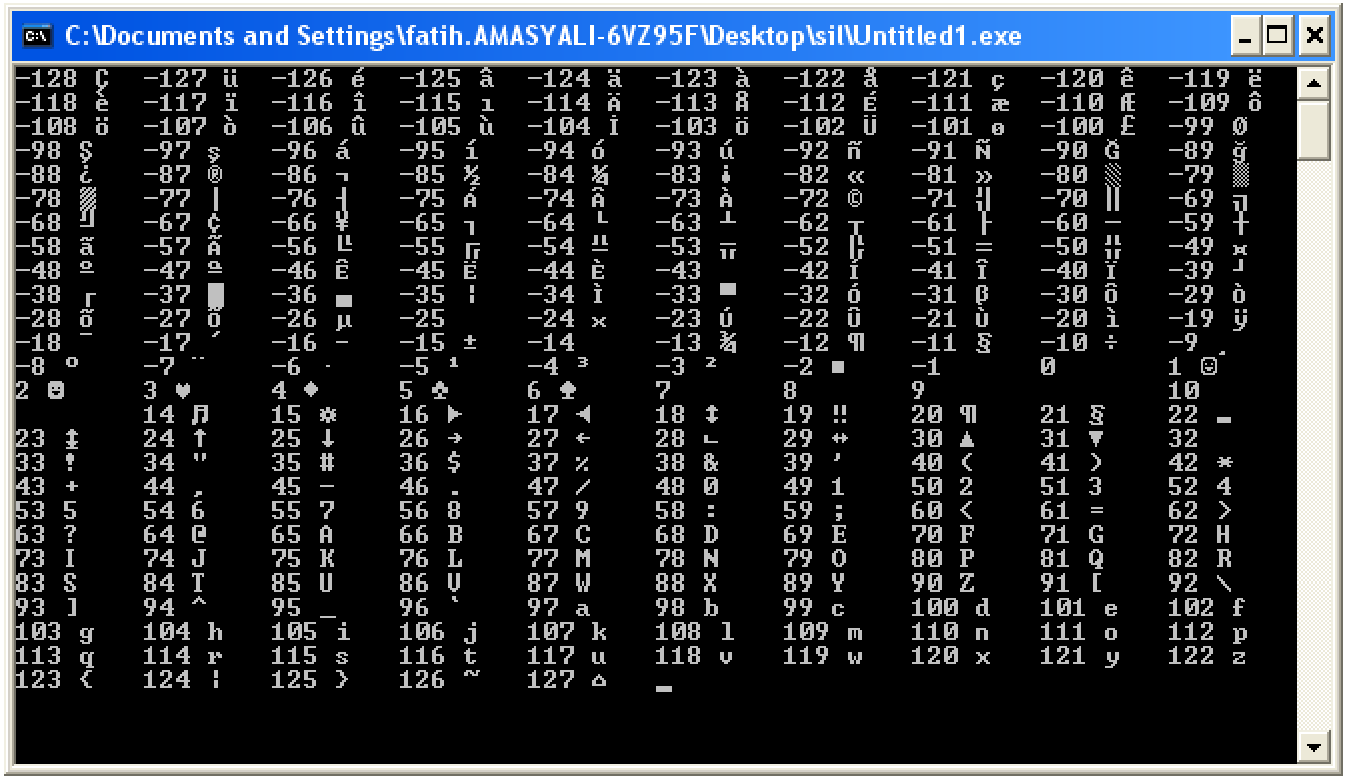
- 'a' and "a" are different; why?

  'a' is the literal 97

  "a" is an array of character literals, { 'a', '\0'} or {97, 0}

- "a" + "b" +"c" is invalid but 'a' + 'b' + 'c' = ? (hint: 'a' = 97 in ASCII)
- 'a' + 'b' + 'c' = 97 + 98 + 99 = 294

```
#include <stdio.h>
#include <conio.h>
int main()
{
    for (int i=-128;i<128;i++)
    printf("%d %c\t",i,i);
    getch();
    return(0);
}
```



C:\Documents and Settings\fatih.AMASYALI-6VZ95F\Desktop\sil\Untitled1.exe

```
-128 Ç  -127 ü  -126 é  -125 â  -124 ä  -123 à  -122 å  -121 ç  -120 ê  -119 ë
-118 è  -117 ï  -116 î  -115 ı  -114 Ä  -113 Å  -112 É  -111 æ  -110 Æ  -109 ô
-108 ö  -107 ò  -106 û  -105 ù  -104 İ  -103 Ö  -102 Ü  -101 ø  -100 £  -99 Ø
-98 Ş   -97 ş   -96 á   -95 í   -94 ó   -93 ú   -92 ñ   -91 Ñ   -90 Ğ   -89 ğ
-88 ¿   -87 ®   -86 ¬   -85 ½   -84 ¼   -83 ¡   -82 «   -81 »   -80 ░   -79 ▒
-78 ▓   -77 │   -76 ┤   -75 Á   -74 Â   -73 À   -72 ©   -71 ╣   -70 ║   -69 ╗
-68 ╝   -67 ¢   -66 ¥   -65 ┐   -64 └   -63 ┴   -62 ┬   -61 ├   -60 ─   -59 ┼
-58 ã   -57 Ã   -56 ╚   -55 ╔   -54 ╩   -53 ╦   -52 ╠   -51 ═   -50 ╬   -49 ¤
-48 º   -47 ª   -46 Ê   -45 Ë   -44 È   -43     -42 Í   -41 Î   -40 Ï   -39 ┘
-38 ┌   -37 █   -36 ▄   -35 ¦   -34 Ì   -33 ▀   -32 Ó   -31 ß   -30 Ô   -29 Ò
-28 õ   -27 Õ   -26 µ   -25     -24 ×   -23 Ú   -22 Û   -21 Ù   -20 ì   -19 ÿ
-18 ¯   -17 ´   -16 -   -15 ±   -14     -13 ¾   -12 ¶   -11 §   -10 ÷   -9 ¸
-8 °    -7 ¨    -6 ·    -5 ¹    -4 ³    -3 ²    -2 ■    -1      0       1 ☺
2 ☻     3 ♥     4 ♦     5 ♣     6 ♠     7       8       9               10
        14 ♫    15 ☼    16 ►    17 ◄    18 ↕    19 ‼    20 ¶    21 §    22 ▬
23 ↨    24 ↑    25 ↓    26 →    27 ←    28 ∟    29 ↔    30 ▲    31 ▼    32
33 !    34 "    35 #    36 $    37 %    38 &    39 '    40 (    41 )    42 *
43 +    44 ,    45 -    46 .    47 /    48 0    49 1    50 2    51 3    52 4
53 5    54 6    55 7    56 8    57 9    58 :    59 ;    60 <    61 =    62 >
63 ?    64 @    65 A    66 B    67 C    68 D    69 E    70 F    71 G    72 H
73 I    74 J    75 K    76 L    77 M    78 N    79 O    80 P    81 Q    82 R
83 S    84 T    85 U    86 V    87 W    88 X    89 Y    90 Z    91 [    92 \
93 ]    94 ^    95 _    96 `    97 a    98 b    99 c    100 d   101 e   102 f
103 g   104 h   105 i   106 j   107 k   108 l   109 m   110 n   111 o   112 p
113 q   114 r   115 s   116 t   117 u   118 v   119 w   120 x   121 y   122 z
123 {   124 |   125 }   126 ~   127 ⌂
```

Initialization

- If a variable is not initialized, the value of variable may be either 0 or garbage depending on the storage class of the variable.

```
int i=5;
float x=1.23;
char c='A';
int i=1, j,k=5;
char c1 = 'A', c2 = 97;
float x=1.23, y=0.1;
```

Memory Concepts

- Each variable has a name, address, type, and value

```
1)int x;
2)scanf(“%d”, &x);
3)user inputs 10
4)x = 200;
```

After the execution of (1) x	
After the execution of (2) x	
After the execution of (3) x	10
After the execution of (4) x	200

Previous value of x was overwritten

Sample Problem

Write a program to take two numbers as input data and print their sum, their difference, their product and their quotient.

Problem Inputs

```
float x, y;               /* two items */
```

Problem Output

```
float sum;                /* sum of x and y */
float difference;         /* difference of x and y */
float product;            /* product of x and y */
float quotient;           /* quotient of x divided by y */
```

Sample Problem (cont.)

- Pseudo Code:

```
Declare variables of x and y;
Prompt user to input the value of x and y;
Print the sum of x and y;
Print the difference of x and y;
Print the product of x and y;
If y not equal to zero, print the quotient of x divided by y
```

```
#include <stdio.h>
#include <conio.h>
int main()
{
    float x,y;
    float sum;
    printf("Enter the value of x:");
    scanf("%f", &x);
    printf("\nEnter the value of y:");
    scanf("%f", &y);
    sum = x + y;
    printf("\nthe sum of x and y is:%f",sum);
    printf("\nthe sum of x and y is:%f",x+y);
    printf("\nthe difference of x and y is:%f",x-y);
    printf("\nthe product of x and y is:%f",x*y);
    if (y != 0)
          printf("\nthe quotient of x divided by y is:%f",x/y);
    else
          printf("\nquotient of x divided by y does not exist!\n");
    getch();
    return(0);
}
```

function

- name
- list of argument along with their types
- return value and its type
- Body

inequality operator

Data Type Conversion

- Rule #1

  `char, short` → **`int`**

  `float` → **`double`**

- Rule #2 (double ← long ← unsigned ← int)
  - If either operand is **`double`**, the other is converted to **`double`**, and the result is **`double`**
  - Otherwise, if either operand is **`long`**, the other is converted to **`long`**, and the result is **`long`**
  - Otherwise, if either operand is **`unsigned`**, the other is converted to **`unsigned`**, and the result is **`unsigned`**
  - Otherwise, the operand must be **`int`**

Examples

Example: `c: char,  u: unsigned, i: int, d: double, f:float,`

`s: short,   l: long,`

Expression	Final Data Type	Explanation
`c – s / i`	`int`	`short→int, int/int, char→int, int-int`

`u * 3 – i`	`unsigned`	`int(3)→unsigned,`
		`unsigned*unsigned=unsigned,`
		`int→unsigned, unsigned-unsigned=unsigned`

`u * 3.0 – i`	`double`	`unsigned→double, double*double,`
		`int→double, double-double=double`

`c + i`	`int`	`char→int`
`c + 1.0`	`double`	`char→int (rule 1), int→double(rule 2)`
`3 * s * l`	`long`	`short→int, int*int, int→long, long*long`

Cast Operator

If a specific type is required, the following syntax may be used, called cast operator.

```
(type) expr
```

Example :

```
float f=2.5;
int x = (int)f + 1;
```

/* x is 3, Q: will f value be changed? */

Assignment

- a=b=c=1;
- Same as a=(b=(c=1));
- a=b=c=d+1;
- But cannot write a=b=c+1=d+1

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

- Syntax

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

`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`**

<u>Example</u>: **`q = q / (q+2) ⇒ q /= q+2`**

<u>More complicated examples</u>:

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

++ (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**

–Simple cases

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

Example:

```
i = 5;
i++; (or ++i;)
printf(“%d”, i) ⇒ 6
```

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

–Complicated cases

```
i = 5;                i   j
j = 5 + ++i;         6   11

i = 5;
j = 5 + i++;         6   10

i = 5;
j = 5 + --i;        4     9

i = 5;
j = 5 + i--;        4    10
```

Precedence of Operators

- You may have learned about this in the third grade:
- 1 + 2 * 3 has the value of 1 + (2 * 3)
- If we want the addition to be performed first, must parenthesize: (1 + 2) * 3.
- We say that * has a higher precedence than +.

Associativity of Operators

- What about operators at the same precedence level? For instance, * and / ?
- Is 12 / 6 * 2 equal to (12 / 6) * 2, or 12 / (6 * 2) ?
- It’s the first: these operators are *left associative* (as are most)
- Moral of story: I say parenthesize when in doubt.

Logical Operators

- The evaluation order for `&&` and `||` is guaranteed to be from left to right
- `a==1 && b!=2 || !c`
- `!(a==1 || b>=3) && c`
- `a>b == b>c`

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 (line 8)
  - Cannot print a **`char`** argument
  - Prints characters until **`NULL`** (**`'\0'`**) encountered
  - Single quotes for character constants (**`'z'`**)
  - Double quotes for strings **`"z"`** (which actually contains two characters, **`'z'`** and **`'\0'`**)

Example:

```
#include <stdio.h>
#include <conio.h>
int main()
{
   char character='A';
   char string[]="This is a string";
   const char *stringPtr ="This is also a string";
   printf("%c\n",character);
   printf("%s\n","This is also a string");
   printf("%s\n",string);
   printf("%s\n",stringPtr);
   getch();
   return(0);
}
```

Program Output

```
A
This is also a string
This is a string
This is also a string
```

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int *ptr;
    int x=1233,y;
    ptr=&x;
    printf("the value of ptr is %p\n",ptr);
    printf( "The address of x is %p\n\n", &x );
    y = printf( "This line has 28 characters\n" );
    printf( "%d characters were printed\n\n", y );
    printf( "Printing a %% in a format control string\n" );

            getch();
            return(0);
}
```

Program Output

```
The value of ptr is 0065FDF0
The address of x is 0065FDF0

This line has 28 characters
28 characters were printed

Printing a % in a format control string
```

```
int i=1256;
printf(“%d”,i);            4 characters 1256
printf(“%3d”,i);           4 characters 1256
printf(“%8d”,i);           8 characters ▲▲▲▲1256
printf(“%05d”,i);          5 characters 01256
printf(“%x”,i);            3 characters 788
printf(“%-5d”,i);          5 characters 1256▲


float buf=125.12;
printf(“%f”,buf);          125.120000
printf(“%.0f”,buf);        125
printf(“%7.2f”,buf);       ▲125.12
printf(“%07.2f”,buf);            0125.12


char buf[] = “hello, world”;
printf(“%10s”,buf);        hello, world
printf(“%-10s”,buf);       hello, world
printf(“%20s”,buf);        ▲▲▲▲▲▲▲▲hello, world
printf(“%20.10s”,buf);     ▲▲▲▲▲▲▲▲▲▲hello, wor
printf(“%-20.10s”,buf);    hello, wor▲▲▲▲▲▲▲▲▲▲
printf(“%.10s”,buf);       hello, wor
```

Example:

```
#include <stdio.h>
#include <conio.h>
int main()
{
   char x,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 );
   getch();
   return(0);
}
```

Program Output:

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

Example:

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int month1, day1, year1, month2, day2, 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 );
    getch();
    return(0);
}
```

Program Output:

```
Enter a date in the form mm-dd-yyyy: 11-18-2000
month = 11  day = 18  year = 2000

Enter a date in the form mm/dd/yyyy: 11/18/2000
month = 11  day = 18  year = 2000
```

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`** */

```
#include <stdio.h>
#include <conio.h>
int main()
{
    puts("deneme");
    putchar('A');
    char buf[128];
    gets(buf);
    printf( "buf is: %s",buf );
    int c;
    c = getchar();
    printf ("c is: %c, c is:%d",c,c);
    getch();
    return(0);
}
```

Program Output:

```
deneme
Asdf sdf sdf
buf is: sdf sdf sdf f
c is: f, c is:102
```

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:

```
max =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 */
```

Compound Statement

{

definitions-and-declarations (optional)

Statement-list

}

- Used for grouping as function body and to restrict identifier visibility
- Note: no semicolon after closing bracet
  - – But every statement in C must be followed by ;

The `if` Selection Structure (cont.)

- A decision can be made on any expression.
    - zero - **false**
    - nonzero - **true**
  - Example:

    `(3 – 4)` is **true**

Selection Structure: `if/else`

- **`if`**/**`else`**
  - **`if:`** only performs an action if the condition is **`true`**
  - **`if/else:`** Specifies an action to be performed both when the condition is **`true`** and when it is **`false`**
- Pseudocode:

```
If (student’s grade is greater than or equal to 60)
   Print “Passed”
else
   Print “Failed”
```

  - Note spacing/indentation conventions
- C code:

```
if ( grade >= 60 )
   printf( "Passed\n");
else
   printf( "Failed\n");
```

The `if/else` Selection Structure

- Ternary conditional operator (**`?:`**)
  - Takes three arguments (condition, value if **`true`**, value if **`false`**)
  - Creates an if/else ***expression***. Recall that expressions are computations that yield a single value.
  - Our pseudocode could be written:

```
printf( "%s\n", grade >= 60 ? "Passed" :
  "Failed" );
```

  - Or it could have been written:

```
grade >= 60 ? printf( “Passed\n” ) :
  printf( “Failed\n” );
```

The `if/else` Selection Structure

- Compound statement:
  - Set of statements within a pair of braces
  - Example:

```
if ( grade >= 60 )
   printf( "Passed.\n" );
else {
   printf( "Failed.\n" );
   printf( "You must take this course again.\n" );
  }
```

  - Without the braces,

```
if ( grade >= 60 )
   printf( "Passed.\n" );
else
   printf( "Failed.\n" );
printf( "You must take this course again.\n" );
```

the statement

```
printf("You must take this course again.\n");
```

would be executed under every condition.

Equality (==) vs. Assignment (=)

- Dangerous error
  - Does not ordinarily cause syntax errors
  - Any expression that produces a value can be used in control structures
  - Nonzero values are **`true`**, zero values are **`false`**

    Example: using **`==`**:

    ```
    if ( payCode == 4 )
       printf( "You get a bonus!\n" );
    ```

    - Checks **`paycode`**, if it is **`4`** then a bonus is awarded

    Example: replacing **`==`** with **`=`**:

    ```
    if ( payCode = 4 )
       printf( "You get a bonus!\n" );
    ```

    - This sets **`paycode`** to **`4`**
    - **`4`** is nonzero, so expression is **`true`**, and bonus awarded no matter what the **`paycode`** was

  - Logic error, not a syntax error

Examples

Ex_1:

```
  if (i=1) y = 3;
```

⇒ **y = 3 is always executed**
this is not the same as

```
  if (i==1) y = 3;
```

Ex_2:

```
  if (i!=0) y=3;
```

⇒ `if (i) y=3;`

Ex_3:

```
if (i==0) y=3;
```

⇒ `if (!i) y=3;`

Examples

```
if (i>2)
    if (j==3)
       y=4;
    else
     y=5;
```

≠

```
if (i>2) {
    if (j==3)
          y=4;
}
else
     y=5;
```

=

```
if (i>2)
     if (j==3)
        y=4;
     else
        ;
else
       y=5;
```

```
if (a>b)
     c = a;
else
     c = b;


⇒c=(a>b)?a:b
```

```
if (x==5)
     y = 1;
else
     y = 0;


⇒ y = (x==5);
```

```
if (x<6)
    y = 1;
else
    y = 2;


⇒ y = 2-(x<6);
⇒ or y = 1+(x>=6);
```

The Essentials of Repetition

- Loop
  - Group of instructions computer executes repeatedly while some condition remains **`true`**
- Counter-controlled repetition
  - Definite repetition: know how many times loop will execute
  - Control variable used to count repetitions
- Sentinel-controlled repetition
  - Indefinite repetition
  - Used when number of repetitions not known
  - Sentinel value indicates "end of data“

Essentials of Counter-Controlled Repetition

- Counter-controlled repetition requires
  - The name of a control variable (or loop counter)
  - The initial value of the control variable
  - A condition that tests for the final value of the control variable (i.e., whether looping should continue)
  - An increment (or decrement) by which the control variable is modified each time through the loop

Example:

```
int counter = 1;             /* initialization */
while ( counter <= 10 ) {    /* repetition condition */
   printf( "%d\n", counter );
   ++counter;                /* increment */
}
```

- The statement

```
int counter = 1;
```

  - Names **counter**
  - Declares it to be an integer
  - Reserves space for it in memory
  - Sets it to an initial value of **1**

Repetition Structure: while

```
#include <stdio.h>
#include <conio.h>
int main()
{
  int counter, grade, total, average;
  /* initialization phase */
  total = 0;
  counter = 1;

  /* processing phase */
  while ( counter <= 10 ) {
    printf( "Enter grade: " );
    scanf( "%d", &grade );
    total = total + grade;
    counter = counter + 1;
  }
  average=total/5;
  /* termination phase */
  printf( "Class average is %d\n", average);
  getch();
  return(0);
}
```

Program Output:

```
Enter grade: 98
Enter grade: 76
Enter grade: 71
Enter grade: 87
Enter grade: 83
Enter grade: 90
Enter grade: 57
Enter grade: 79
Enter grade: 82
Enter grade: 94
Class average is 81
```

```
#include <stdio.h>
#include <conio.h>
int main()
{
   int counter, grade, total;
   float average;
   /* initialization phase */
   total = 0;
   counter = 1;
   printf( "Enter grade, -1 to end: " );
   scanf( "%d", &grade );
   while ( grade != -1 ) {
         total = total + grade;
         counter = counter + 1;
         printf( "Enter grade, -1 to end: " );
         scanf( "%d", &grade );
   }    /* termination phase */
   if ( counter != 0 ) {
      average = ( float ) total / counter;
      printf( "Class average is %.5f", average );
   }
   else
       printf( "No grades were entered\n" );
   getch();
   return(0);
}
```

```
Enter grade, -1 to end: 45
Enter grade, -1 to end: 12
Enter grade, -1 to end: 1
Enter grade, -1 to end: 78
Enter grade, -1 to end: 9
Enter grade, -1 to end: -1
Class average is 24.16667
```

Repetition Structure: `for`

- **`for`** loops syntax
- **for (initialization ; loopContinuationTest ; increment)**
  statement

  Example: Prints the integers from one to ten

```
for ( counter = 1; counter <= 10; counter++ )
   printf( "%d\n", counter );
```

No semicolon (`;`) after last expression

- For loops can usually be rewritten as **`while`** loops:

```
initialization;
while ( loopContinuationTest ) {
   statement;
   increment;
}
```

- Initialization and increment
  - Can be comma-separated list of statements

  Example:

```
for ( i = 0, j = 0;  j + i <= 10; j++, i++)
    printf( "%d\n", j + i );
```

The `for` Structure (cont.)

- Arithmetic expressions
  - Initialization, loop-continuation, and increment can contain arithmetic expressions. If **`x`** equals **`2`** and **`y`** equals **`10`**

```
for ( j = x; j <= 4 * x * y; j += y / x )
```

is equivalent to

```
for ( j = 2; j <= 80; j += 5 )
```

- Notes about the **`for`** structure:
  - "Increment" may be negative (decrement)
  - If the loop continuation condition is initially **`false`**
    - The body of the **`for`** structure is not performed (i.e. pre-test)
    - Control proceeds with the next statement after the **`for`** structure

The `for` Structure (cont.)

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int sum = 0, number;
    for ( number = 2; number <= 100; number += 2 )
        sum += number;
    printf( "Sum is %d, Final number is %d\n", sum,number );
   getch();
   return(0);
}
```

Program Output:

2 + 4 + 8 + … +100 = 2550

```
Sum is 2550, Final number is 102
```

Repetition Structure: `do/while`

- The **`do/while`** repetition structure
  - Similar to the **`while`** structure
  - do/while is a "post-test" condition. The body of the loop is performed at least once.
    - All actions are performed at least once
  - Format:

```
do {
   statement;
} while ( condition );
```

Repetition Structure: do/while

```
int main()
{
    int counter=0;
    do {
    printf( "%d  ", counter );
    } while (++counter <= 10);
    printf( "Final counter is %d\n", counter);
    getch();
    return(0);
}
```

Program Output:

```
0  1  2  3  4  5  6  7  8  9  10  Final counter is 11
```

```
  } while (++counter <= 10);
  } while (counter++ <= 10);
```

```
 0  1  2  3  4  5  6  7  8  9  10  11  Final counter is 12
```

Multiple-Selection Structure: `switch`

- **`switch`**
  - Useful when a variable or expression is tested for all the values it can assume and different actions are taken
- Format
  - Series of **`case`** labels and an optional **`default`** case

```
switch ( value ){
   case '1':
       actions
   case '2':
       actions
   default:
       actions
   }
```

  - **`break;`** exits from structure

```c
/* Fig. 4.7: fig04_07.c
 Counting letter grades */
#include <stdio.h>

int main()
{
 int grade;
 int aCount = 0, bCount = 0, cCount = 0, dCount = 0,
                 fCount = 0;

   printf(  "Enter the letter grades.\n"  );
   printf(  "Enter the 'X' character to end input.\n"  );

   while ( ( grade = getchar() ) != 'X' ) {

      switch ( grade ) {    /* switch nested in while */

         case 'A': case 'a':  /* grade was uppercase A */
            ++aCount;         /* or lowercase a */
            break;

         case 'B': case 'b':  /* grade was uppercase B */
            ++bCount;         /* or lowercase b */
            break;

         case 'C': case 'c':  /* grade was uppercase C */
            ++cCount;         /* or lowercase c */
            break;

         case 'D': case 'd':  /* grade was uppercase D */
            ++dCount;         /* or lowercase d */
            break;

         case 'F': case 'f':  /* grade was uppercase F */
            ++fCount;         /* or lowercase f */
            break;

```

1. **Initialize variables**
2. **Input data**
3. **Use switch loop to update count**

```c
         case '\n': case' ':  /* ignore these in input */
            break;

         default:        /* catch all other characters */
            printf( "Incorrect letter grade entered." );
            printf( " Enter a new grade.\n" );
            break;
      }
   }

   printf( "\nTotals for each letter grade are:\n" );
   printf( "A: %d\n", aCount );
   printf( "B: %d\n", bCount );
   printf( "C: %d\n", cCount );
   printf( "D: %d\n", dCount );
   printf( "F: %d\n", fCount );

   return 0;
}
```

4. Print results

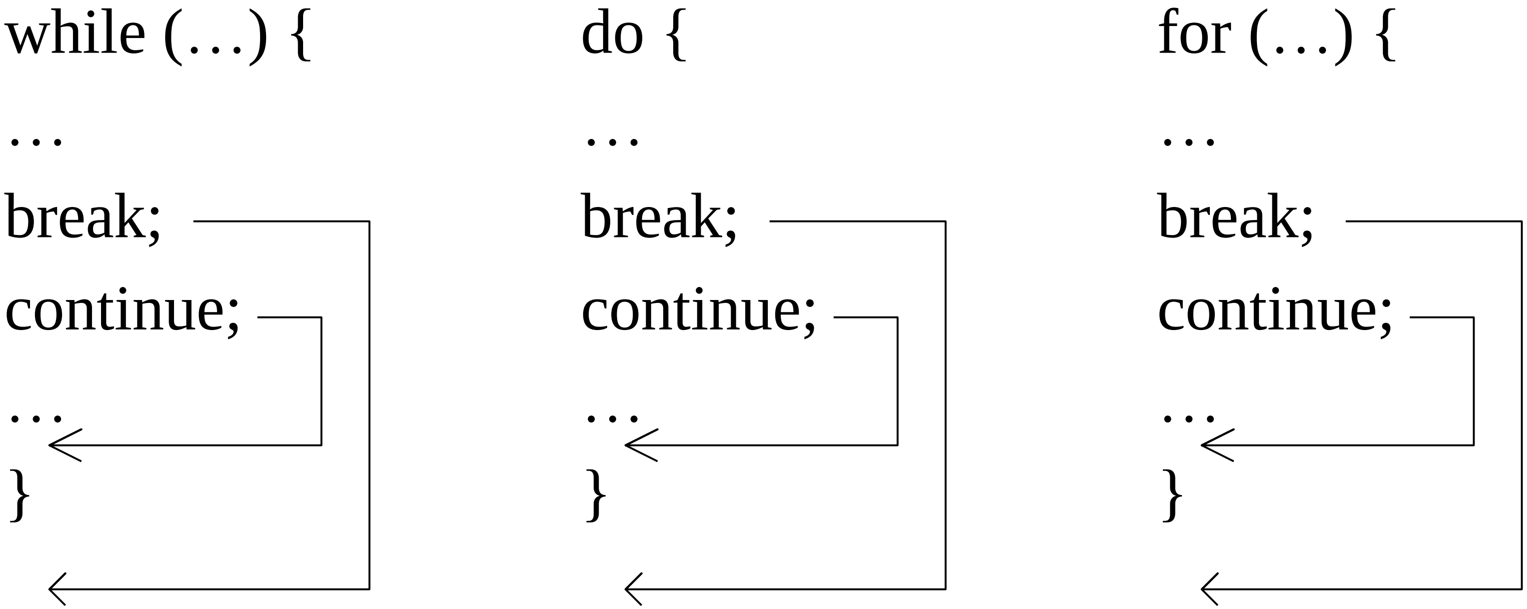
Program Output:

```
Enter the letter grades.
Enter the ‘X’ character to end input.
A
B
C
C
A
D
F
C
E
Incorrect letter grade entered. Enter a new grade.
D
A
B
X
Totals for each letter grade are:
A: 3
B: 2
C: 3
D: 2
F: 1
```

Switch example

```
switch( month ) /* month given as integer(1-12) */
{
      case 1: case 3: case 5: case 7: case 8:
      case 10:
      case 12:
            printf( “31 days.\n” );
            break;
      case 2:
            printf( “28 days.\n” );
            break;
      default;
            printf( “30 days.\n” );
}
```

break & continue



```
while (…) {
…
break;
continue;
…
}
```

```
do {
…
break;
continue;
…
}
```

```
for (…) {
…
break;
continue;
…
}
```

Conclusion:

Break: stops execution of loop and continues after it

Continue: stops execution of the loop and lets it continue from the start again (as long as the condition remains true!)

50 karakterlik veya bir satırlık bir giriş beklendiğini varsayarsak;

```
for (cnt=0; cnt<50; cnt++)
{
      c=getchar();
      if (c==’\n’)
                break;
      else
                …
}
```

```
int main(int argc, char *argv[])
{
char digit;
int num=0;
while ((digit=getchar()) != '\n')
    {
    if (isdigit(digit) == 0)
          continue;
    num = num*10;
    num = num + (digit - '0');
    }
printf("%d",num);
system("PAUSE");
return 0;
}
```

4te42rgfs6
girildiğinde ekrana
4426 yazar

Variation: **rect2.c**

Print an m by n rectangle of asterisks

```
input width and height

for each row
{
   for each column in the current
     row
   {
     print an asterisk
   }
  start next row

}
```

```c
#include <stdio.h>

/*  Print an m-by-n rectangle of
    asterisks */
int main()
{
  int rowc, colc, numrow, numcol;

  printf("\nEnter width: ");
  scanf("%d", &numcol);
  printf("\nEnter height: ");
  scanf("%d", &numrow);

  rowc = 0;
  while (rowc < numrow)
  {
    for (colc=0; colc < numcol; colc++)

    {
      printf("*");
    }

    printf("\n");
    rowc++;
  }
return 0;
}
```

Variation: **rect3.c**

```
Print an m by n rectangle of
asterisks


input width and height


for each row
{
   for each column in the current
     row
   {
      print an asterisk
   }


  start next row
}
```

```c
#include <stdio.h>
/* Print an m-by-n rectangle of
   asterisks */
int main()
{
  int rowc, colc, numrow, numcol;

  printf("\nEnter width: ");
  scanf("%d", &numcol);
  printf("\nEnter height: ");
  scanf("%d", &numrow);

for (rowc=0; rowc < numrow; rowc++)
  {
    colc = 0;
    while (1)
    {
      printf("*");
      colc++;
      if (colc == numcol)
       { break; }
    }
    printf("\n");
  }
  return 0;
}
```

Referance

- Ioannis A. Vetsikas, Lecture notes
- Dale Roberts, Lecture notes