Today

- Constants & Data types
- Simple input/output
- Conditionals

Integer constants
- short
- int
- unsigned int
- long int
- unsigned long int

String Constants
- “deneme bir iki”
- “deneme bir iki”\“uc dort bes”
- “deneme \” bir iki”
Character Constants

- ‘a’, ‘1’, ‘%’, ...
- ‘\’
- ‘c’ vs “c”

Floating Points

- float
- double
- long double

Type Conversion

<table>
<thead>
<tr>
<th>char, short</th>
<th>int</th>
<th>long</th>
<th>float</th>
<th>double</th>
<th>long double</th>
</tr>
</thead>
</table>

Automatic Type Conversion Rules

* Advice: Avoid automatic type conversion!

Arithmetic Operators & Precedence

- C uses infix notation: a + b * c
- prefix notation: + a * b c
- postfix notation: a b c * +

<table>
<thead>
<tr>
<th>Operator</th>
<th>Type</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ -</td>
<td>Unary</td>
<td>Right to left</td>
</tr>
<tr>
<td>* / %</td>
<td>Binary</td>
<td>Left to right</td>
</tr>
<tr>
<td>+ -</td>
<td>Binary</td>
<td>Left to right</td>
</tr>
</tbody>
</table>
Increment, Decrement Operators

- \( ++a, --a \)
  vs
- \( a++, a-- \)

Compound Assignment Operators

- \( \text{var op= expr} \)
- \( +=, -=, *=, /=, %= \)

Some examples

- \( i += j = k; \)
- \( i = j += k; \)

Output

- \( \text{printf(format string, var1, var2, ...)} \)
  - Format string contains:
    - \( d, i \): integers
    - \( f \): float, double
    - \( e \): float, double in exponential notation
    - \( c \): character
    - \( s \): string
Input

- `scanf(format string, var1, var2, ...)`
  - var1, var2, ...: addresses of memory locations!
  - Format string contains:
    - d,i: integers
    - f: float, double
    - e: float, double in exponential notation
    - c: character
    - s: string

Simple Macros

- For long and/or frequent constants:
  - `#define PI 3.14159265`
- For long and/or frequent calculations:
  - `#define Area(Radius) (4*PI*Radius*Radius)`
  - ... `a = 10.0 + Area(2.0);`

Side Effects

- Due to expressions that change variables:
- Example:
  - `i = (i++) ? (i+1) : (i-1);`
  - `v + v++ + v + v++`

Example

- A gasoline ('benzin') and diesel engine versions of the same car model consume different amounts of petrol: `p_g, p_d` (in liters per km), usually `p_g > p_d`. These two different versions of the same car model have different prices: `c_g, c_d` (usually, `c_g < c_d`).

- Write a program that gets the values `p_g, p_d, c_g, c_d` as well as the price of 1 liter gasoline and 1 liter diesel from the user and calculates in how many kilometers the price difference these two versions is amortized.
Examples

• main()
  {
  float me = 1.1;
  double you = 1.1;
  if(me==you)
    printf("Me & You");
  else
    printf("You & Me");
  }

Examples

• main()
  {
    static int var = 5;
    printf("%d ",var--);
    if(var)
      main();
  }

Examples

• main()
  {
    int i=-1,j=-1,k=0,l=2,m;
    m=i++&&j++&&k++||l++;
    printf("%d %d %d %d
"),i,j,k,l,m);
  }

Examples

• main()
  {
    int i=10;
    i=!i>14;
    printf("i=%d",i);
  }

Examples

• #define square(x) x*x
  main()
  {
    int i;
    i = 64/square(4);
    printf("%d",i);
  }

Examples

# include <stdio.h>
#define a 10
main()
{
    int i;
    printf("%d",i++ + ++i);
}

Examples

• void main()
  {
    int i=5;
    printf("%d",i++ + ++i);
  }
Today

- Relational (<, <=, >, >=, ==, !=)
- Logical Operators (&&, ||)
- Changing the flow of the program
  - Conditional statements
  - Conditional expressions
- Repetitions
  - Iterative statements
    - while
    - do-while
    - for

Relational Operators

- < <= > >= == !=
- False means 0 (zero)
- True means anything that is not False (i.e., non-zero)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Type</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ - ++ --</td>
<td>Unary</td>
<td>Right to left</td>
</tr>
<tr>
<td>* / %</td>
<td>Binary</td>
<td>Left to right</td>
</tr>
<tr>
<td>+ -</td>
<td>Binary</td>
<td>Left to right</td>
</tr>
<tr>
<td>&lt; &lt;= &gt; &gt;=</td>
<td>Binary</td>
<td>Left to right</td>
</tr>
<tr>
<td>== !=</td>
<td>Binary</td>
<td>Left to right</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>= *= /= %/= += -=</td>
<td>Binary</td>
<td>Right to left</td>
</tr>
</tbody>
</table>

Example: \[ a = b + c <= d + e == c - d \]

Logical Operators

- && || !

<table>
<thead>
<tr>
<th>Operator</th>
<th>Type</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ - ++ --</td>
<td>Unary</td>
<td>Right to left</td>
</tr>
<tr>
<td>* / %</td>
<td>Binary</td>
<td>Left to right</td>
</tr>
<tr>
<td>+ -</td>
<td>Binary</td>
<td>Left to right</td>
</tr>
<tr>
<td>&lt; &lt;= &gt; &gt;=</td>
<td>Binary</td>
<td>Left to right</td>
</tr>
<tr>
<td>== !=</td>
<td>Binary</td>
<td>Left to right</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>= *= /= %/= += -=</td>
<td>Binary</td>
<td>Right to left</td>
</tr>
</tbody>
</table>

Changing the flow of the program

- if statements
  
```c
if(expr)
{
    ....
    printf(“a is bigger”);
}
else if(a < b)
    printf(“b is bigger”);
else
    printf(“a = b”);
```

```c
if(a > b)
{
    ....
    printf(“a is bigger”);
}
else
```
Changing the flow of the program

- Common mistake with if statements
- `if(a = 10) { ... }
- `if(a == 10); { ... }

Conditional Expression Operator

- Conditional expression:
  - `Expr ? True-expr : False-expr`
  - `int a = x > 10 ? 1 : 0;
- Right-to-left associative.
  - `X = c ? a : d ? e : f;
- Precedence:
  - `c ? X = a : X = b`
  - ‘?’ and ‘:’ bracket the expression. True-expr can have operators of any precedence without parentheses.
  - The False-expr part has lower precedence than all operators except ‘=’ and ‘,’.

Nested if statements

- `if(...) 
  if(...) 
  {....}
else
  {....}

Changing the flow of the program

- Multi-way conditionals: switch statements

```java
switch(expr)
{
    case value-1:
        ....
    break;
    case value-2:
        ....
    break;
    default:
        ....
    break;
}
```
Example

- main()
  {
    int i=3;
    switch(i)
    {
      default: printf("zero");
      case 1: printf("one");
             break;
      case 2: printf("two");
             break;
      case 3: printf("three");
             break;
    }
  }

- main()
  {
    int i=1;
    switch(i)
    {
      default: printf("zero");
      case 1: printf("one");
             case 2: printf("two");
             break;
      case 3: printf("three");
             break;
    }
  }

Repetitions

- while loop
  
  *Initialization;
  while( expr )
    statement;

  *Initialization;
  while( expr )
  {
    statement;
    statement;
  }

Example

- Write a C program that classifies a given character into one of the following:
  - Number
  - Uppercase letter
  - Lowercase letter
  - Operator
  - Whitespace

Example

- Factorial
  
  int N, fact = 1;
  scanf("%d", &N);
  while( N > 0 )
  {
    fact *= N--;
  }

- Bad examples:
  
  while( x = 1 )
  {
    x = getchar();
  }

  x = 0.0;
  while( x != 1.0 )
  {
    x += 0.005;
  }
Repetitions

• do-while loop
  
  Initialization;
do
  statement
while( expr );
  statement;

  Initialization;
do
  {  
    statement;
    statement;
    statement;
  } while( expr );

Example

• Factorial

  int N, fact = 1;
scanf("%d", &N);
do
  {  
    fact *= N--;
  } while( N > 0 );

Repetitions

• for loop

  Initialization;
  for( expr1; expr2; expr3 )
  statement

  Initialization;
  for( expr1; expr2; expr3 )
  {
    statement;
    statement;
    statement;
  }

  for( j = 0; j < N; j++)
    printf("j: %d\n", j);

  for( i=0, j=0;
    i < 0 & j > N; i++, j--);

  for( ; ; i++)
  {
    if( i > 0 ) return 0;
    
  break;

  
  void

  Stop the loop/iteration and continue with the statement after the loop.
  
  while( 1 )
  {
    c = getchar();
    if( c == EOF)
      break;
    printf( c );
  }

  while( 1 )
  {
    c = getchar();
    if( c == EOF)
      break;
  
  }
continue;

- Skips the remaining statements in the loop and continues with the “loop head”.
- Usable with while, for and do-while

```c
while(...) {
    ... continue;
    ...
}
```

```c
Sum = 0;
for(i=0; i<N; i++)
{
    if( i%2 == 0 )
        continue;
    sum = sum + i;
}
```

**Nested Loops**

- You can have loops within loops:

```c
for(i=0; i<N; i++)
{
    for(j=0; j<N; j++)
    {
        ...
    }
}
```

**Example**

- Write a program that, given \((a,b,c,d)\), finds all the points \((x,y,z)\) that satisfy the following equation:
  \[- ax + by + cz + d = 0 \]

**Example**

- Write a C code that multiplies two numbers without using *, / or %.