# Session 4

## Variables and memory

- Each time we declare a variable of a given type (e.g int), the compiler allocates sufficient space in memory to store values specific to that particular type of variable
- Computer's memory is organized into bytes. Each variable will occupy some number of bytes in memory
- Variables of different types (e.g short, float, long, etc) may require different ammounts of memory to be allocated

---

## Integer types

---

**Signed Integer types**

- We have five basic types that we can use in order to store signed integer values.
- By signed we refer to positive and negative values
- Table with Signed Integer types:

| Type name | Number of bytes |
|---|---|
| `signed char` | 1 |
| `short` | 2 |
| `int` | 4 |
| `long` | 4 |
| `long long` | 8 |

- Some examples for variables of mentioned types:

```
short shoeSize;
int houseNumber;
long long starCount;
```

- Note: short, long, and long long can be written as:
  - `short int`
  - `long int`
  - `long long int`
  - However, these types are almost always written in thei abbreviated forms. Type int can also be written as `signed int`, but you won't see this often.

## Unsigned Integer Types

- Some kinds opf data are always positive
    - number of children in a class
    - a man's age
- For each type that stores signed integers, there is a corresponding type that stores unsigned integers
- The unsigned type occupies the same amount of memory as the signed type
- Each unsigned type name is essentially the signed type name prefixed with the keyword unsigned
- Table with unsigned integer types:

| Type name | Number of bytes |
|---|---|
| `unsigned char` | 1 |
| `unsigned short` | 2 |
| `unsigned int` | 4 |
| `unsigned long` | 4 |
| `unsigned long long` | 8 |

- Occupying the same number of bytes, the number of different values that can be represented is fixed.
- Using an unsigned type doesn't provide more values than the corresponding signed type, but it does allow numbers to be represented that are twice the magnitude
    - For example, signed char can represent values between -128 and 127 while unsigned char can occupy values between 0 and 255.
- Example of unsigned integer variable declarations:
    - `unsigned int count`
    - `unsigned long population`

---

## Floating point types

---

- Floating point numbers hold values that are written with a decimal point, so you can represent fractional as well as integral values
- Examples of floating-point numbers
    - `1.6`
    - `0.0000009`
    - `234234.434`
    - `100.0`
- Floating-point numbers are often expressed as decimal values multiplied by some power of 10, where the power of 10 is called the exponent.

- Example of expressing floating-point values:

| Value | With an exponent |
|---|---|
| `1.6` | `0.16 x 10^1` |
| `0.00008` | `8.0 x 10^-5` |
| `7655.899` | `0.7655899 x 10^4` |
| `100.0` | `1.0 x 10^2` |

**Floating-Point Variables**

- Floating-point variable types only store floating-point numbers
- There are three types fo floating point numbers:

| Type name | Number of bytes | Decimal Digits Precision |
|---|---|---|
| `float` | 4 | 7 |
| `double` | 8 | 15 |
| `long double` | 12 | 18 |

- Example of floating-point variables declaration:
  - `float radius`
  - `double biggest`
- Values of type float are known as single precision floating-point numbers.
- In order to specify that a number is of type float, we have to add `f` after its value:
  - `float area = 24.56f`;
- If a floating-point value does not have the `f` at the end, the compiler will interpret it as a value of type `double`

**Conversion specifiers for Floating-Point Variables**

- In order to display floating point variables we use `%f` as conversion specifier
  - e.g `printf("%f divided by %f is %f", 10.0, 4.0, 10.0/4.0)`

**Controlling the number of decimals in the output**

- If a division contains a lot of decimals in the output (e.g 34.45454555555555) we can decide how many we want to be displayed (for example we only want to display 3 decimals, thus obtaining 34.454)
- This can be done using the following syntax of the conversion specifier:
  - `printf("%.nf", xx.xxx)`
    - `n` is the number of decimals we want to display

- `xx.xxxx` is the decimal number we want to format
- example: `printf("%.3f",34.4545455555555)` will display `34.454`

#### Floating point exercises

1. Given a land of 25 square kilometers splitted into 12 equal smaller areas, what is the surface of each smaller area?

```
#include <stdio.h>

int main()
{
    float landArea = 25.0f;
    float numberOfSlices = 12;
    float sliceSurface = landArea / numberOfSlices;
    printf("If we split a land of 25 square km into 12 pieces, each piece
will have %f square km", sliceSurface);
    return 0;
}
```

### Accepting input from user

- In order to read data entered by user in the terminal, we can use the `scanf` function
- This function also require us to include the `stdio.h` header:
  - `#include <stdio.h>`
- Syntax: `scanf("<format_specifier>", &<variable_name>)`
  - `<format_specifier>` can be any of %f or %d depending on what kind of data we are expecting (floating point numbers, decimal numbers, characters, etc)
  - `<variable_name>` is the name of the variable which will hold the data entered by the user.
  - example:

```
    float diameter;
    scanf("%f", &diamester)
```

#### Advanced exercises using floating point numbers

1. Create a program which calculates the area and the circumference of a circle. The program will read the diameter value from the terminal
   - Solution:

```
#include <stdio.h>

int main()
{
    float radius = 0.0f;
```

```
    float diameter = 0.0f;
    float circumference = 0.0f;
    float area = 0.0f;
    float Pi = 3.14159265f;

    printf("Input the diameter of the circle: ");
    scanf("%f", &diameter);
    radius = diameter / 2.0f;
    circumference = 2.0 * Pi * radius;
    area = Pi * radius * radius;

    printf("\nThe circumference is %.2f", circumference);
    printf("\nThe area is %.2f \n", area);

    return 0;
}
```

2. Write a program to compute the area and the permiter of a rectangle. The width and the height are read from the terminal.
    - Solution:

```
#include <stdio.h>

int main()
{
    float width = 0.0f;
    float height = 0.0f;
    float area = 0.0f;
    float perimeter = 0.0f;

    printf("Enter the width of the rectangle: \n");
    scanf("%f", &width);
    printf("Enter the height of the rectangle: \n");
    scanf("%f", &height);

    perimeter = 2 * width + 2 * height;
    area = width * height;
    printf("Rectangle perimeter: %.2f\n", perimeter);
    printf("Rectangle area: %.2f", area);
    return 0;
}
```

3. Write a program which reads from the keyboard the number of the days and returns the number of years, weeks and days.

- Note: ignore leap years
- Example:

  - Input: 1329 days
  - Output: 3 years 33 weeks and 3 days

- Solution:

```
    #include <stdio.h>

    int main()
    {
        printf("Please enter the number of days: \n");
        int totalNumberOfDays = 0;
        scanf("%d", &totalNumberOfDays);

        int years = totalNumberOfDays / 365;
        int remainingDays = totalNumberOfDays % 365;
        int weeks = remainingDays / 7;
        int days = remainingDays - (weeks * 7);
        printf("%d days is: %d years and %d weeks and %d
days",totalNumberOfDays, years, weeks, days);

        return 0;
    }
```

4. Write a C program to calculate the distance between the two points. Note: x1, y1, x2, y2 are all floating point values.

- Solution:

```
#include <stdio.h>
#include <math.h>
int main()
{
    float x1;
    float y1;
    float x2;
    float y2;
    printf("Please enter the x coordinate of the first point: ");
    scanf("%f",&x1);
    printf("Please enter the y coordinate of the first point: ");
    scanf("%f",&y1);
    printf("Please enter the x coordinate of the second point: ");
    scanf("%f",&x2);
    printf("Please enter the y coordinate of the second point: ");
    scanf("%f",&y2);

    float distance = sqrt(pow((x2-x1), 2) + pow((y2-y1),2));

    printf("The distance between Point A(%.2f,%.2f) and Point B(%.2f,%.2f)
is %.2f", x1,y1, x2,y2, distance);
```

```
    return 0;
}
```

## Homework exercises

1. Create a program which reads two floating point numbers from the keyboard and computes their product
2. Create a program which reads two floating number and returns the result of their division with 3 decimal digits.
3. Create a program which reads two numbers, x and y and returns x to the power y. Make use of the `pow` function from the `math.h` library.
4. Create a program which reads two numbers and returns their average.
5. Create a program which reads a number from the keyboard and returns its square root. Make use of the `pow` function from the `math.h` library.
6. Create a program which reads the number of seconds from the keyboard and returns the number of hours, minutes and seconds in the following format: `x seconds are y hours z minutes and w seconds` where `x` represents the initial number of seconds (Read from the keyboard), `y` is the number of hours, `z` is the number of minutes and `w` is the number of seconds