Lecture 2

data types

What we have seen so far

- Basic elements of C
- Comments: /* */ - multi line, // - single line
- #include <> or " " - appending headers
- Functions
- semicolons ;
- Brackets (), {}

Program

```
1  /* Description - not mandatory but polite*/
2  #include <stdio.h> //Preprocessor commands starting with a \# Note
       no semicolons ";".
3
4  void fun2(); // declaration of fun 2, definition below
5
6  void fun1() // definition of fun 1
7  {
8    printf("Hello␣from␣1!\n")
9  }
10
11 int main() //The main function must be there
12 {
13   fun1();
14   printf("Hello␣from␣main!\n", c); // \n starts new line
15   fun2();
16   return 0;
17 }
18
19 void fun2() // definition of fun2
20 {
21   printf("Hello␣from␣2!\n")
22 }
```

Data types

Everything lives in computer's memory

Our program processes data, and everything it does or uses is stored in computer's memory. Basic data types allow for declaration of variables and allocation of necessary resources (space in memory).

For today:

- Simple data types with examples.
- Arithmetic operations.
- Precedence of operators.
- Some fun with characters.
- Printing of variables with *printf()*

Selected data types in C

- Used to declare variables or define functions.
- Determine size the variable occupies in memory.
- Need format specifiers to print with *printf()*.
- We limit our interest to integers, real numbers, characters, boolean and ... void types
- The size defined by a specific data type might vary with implementation (32b vs 64b), use *sizeof()*
- There is more ...

Integers

int, unsigned int, long int, long long int, unsigned long long int

- 4 6 842 -6 2 1024 - integers
- 4.8 3.141592 1.- not integers - mind the dot
- keyword **int**
- 2 or **4** Bytes (B)
- $-2,147,483,648 - 2,147,483,647$
- Format specifiers for *printf():*
  - *%d %i - signed integer (%li %ld for long).*
  - *%o - Octal integer.*
  - *%x %X - Hex integer.*
  - *%u - unsigned integer.*
- *012 in octal representation*
- *0x10 in hexadecimal representation*

int

```c
#include <stdio.h>

int main()
{
   printf("Storage size for int : %ld B \n", sizeof(int));
   int a=032;
   int b=0x23;
   int c=23;
   int d=-1;
   printf("a=%d, b=%d, c=%d, d=%d\n", a, b, c, d);
   printf("a=%x, b=%x, c=%x, d=%x\n", a, b, c, d);
   printf("a=%o, b=%o, c=%o, d=%o\n", a, b, c, d);
   printf("a=%u, b=%u, c=%u, d=%u\n", a, b, c, d);
}
```

int arithmetic

```
1  #include <stdio.h>
2
3  int main()
4  {
5    int a=5, b=4;
6    printf("%d+%d=%d\n", a, b, a+b);
7    printf("%d-%d=%d\n", a, b, a-b);
8    printf("%d*%d=%d\n", a, b, a*b);
9    printf("%d/%d=%d\n", a, b, a/b);
10   printf("%d/%d=%d\n", b, a, b/a);
11   printf("%d%%%d=%d\n", a, b, a%b);
12 }
```

- a + b addition
- a - b subtraction
- a * b multiplication
- a / b division
- a % b remainder of division

note 1: To print % one needs %% !

note 2: The result has the same type *int* as arguments!

Real floating-point

float, double, long double

4.8 3.141592 1.0 0.5 -3.14 2. - real numbers

float

- keyword **float**
- 4B (single precision!)
- $1.2 \cdot 10^{-38}$ to $3.4 \cdot 10^{38}$
- 6 decimal places
- Format specifiers:
  - %f

double

- keyword **double**
- 8B (double precision!)
- $2.3 \cdot 10^{-308}$ to $1.7 \cdot 10^{308}$
- 15 decimal places
- Format specifiers:
  - %lf

- %e %E - Scientific notation.
- %g %G - Similar as %e or %E.

float and double

```c
#include <stdio.h>
#include <math.h> // the math library

int main()
{
  printf("Storage size for float : %ld B \n", sizeof(float));
  printf("Storage size for double : %ld B \n", sizeof(double));
  float a = 4.0 * atan(1.0); // This is PI
  double b = 4.0 * atan(1.0); // This is PI
  printf("a=%f, a=%e, a=%E, a=%g, a=%G\n", a, a, a, a, a);
  printf("b=%f, b=%e, b=%E, b=%g, b=%G\n", b, b, b, b, b);
  a = 6.02214085774e23;
  b = 6.02214085774e23;
  printf("a=%f, a=%e, a=%E, a=%g, a=%G\n", a, a, a, a, a);
  printf("b=%f, b=%e, b=%E, b=%g, b=%G\n", b, b, b, b, b);
}
```

float and double arithmetic

```
1  #include <stdio.h>
2
3  int main()
4  {
5    double a=5.0, b=4.0;
6    printf("%lf+%lf=%lf\n", a, b, a+b);
7    printf("%lf-%lf=%lf\n", a, b, a-b);
8    printf("%lf*%lf=%lf\n", a, b, a*b);
9    printf("%lf/%lf=%lf\n", a, b, a/b);
10   printf("%lf/%lf=%lf\n", b, a, b/a);
11 }
```

- a + b addition
- a - b subtraction
- a * b multiplication
- a / b division

note 1: The result has the same type *float* or *double* as arguments!

Characters
char

- 'a' 'b' 'c' '1' etc.
- or hexadecimal ASCI code, e.g.: '\x15' '\x9c'
- keyword **char**
- 1B
- "a" is not 'a' ! "a" is 'a' and '\0' - null sign
- Format specifiers: %c

char

```c
#include <stdio.h>

int main()
{
   printf("Storage size for char : %ld B \n", sizeof(char));
   char a = 'a';
   printf("a=%c \n", a);
   printf("a=%d \n", a);
   a = '\x15';
   printf("a=%c \n", a);
   printf("a=%d \n", a);
   a = "R";
   printf("a=%c \n", a);
   printf("a=%d \n", a);
   a = 'R';
   printf("a=%c \n", a);
   printf("a=%d \n", a);
   a = '\0';
   printf("a=%c \n", a);
   printf("11 %c 11 %c 11 %c 11 \n", '\0', '\x00', '\x30');
   printf("11 %d 11 %d 11 %d 11 \n", '\0', '\x00', '\x30');
}
```

bool

To be full of bool? To be false or true?

- Represents logical value
- introduced in C99
- need #include <stdbool.h>
- *true* or *false*
- keyword **bool** or **_Bool**
- 1B or same as int (platform dependant)
- Has no format specifier, use %d (see example)

bool

```c
#include <stdio.h>
#include <stdbool.h>

int main()
{
   printf("Storage size for char : %ld B \n", sizeof(bool));
   bool a = true;
   printf("a=%d\n", a);
   a = false;
   printf("a=%d\n", a);

   //For curious students:
   a = true;
   printf("a=%s \n", a ? "true" : "false");
   a = false;
   printf("a=%s \n", a ? "true" : "false");
}
```

void

Specifies no value available

- Functions with no return are **void**
- Functions that accept no arguments accepts **void**
- There can be a pointer (what is a pointer?) to **void** - it points to an address, but does not specify the variable type - more later on
- keyword **void**
- 1B (?) - it has no size

```
1 #include <stdio.h>
2
3 int main()
4 {
5   printf("Storage size for void : %ld B \n", sizeof(void)); //should
        not work, works in gcc
6   void a; //Not possible
7   printf(" %d \n", a);
8 }
```

Operations on variables

Substitution:

```
1 int a,b; //declare two variables of type int
2 a=35; //assigment: a store value of 35
3 b=6; //b store 6
```

An arithmetic expresion:

```
1 a=a+b;// perform addition in temporary space, and assign the result
      to a
```

Assignment

When using '=' sign variable on the Left of '=' is assigned the value on the Right. This does not necessarily means equality!
The Value on the Right is calculated first and later the value is copied to the Left. Types on the Left and Right should be the same and type mixing should be avoided.

```
1 double x1=6.28;
2 int a = 2
3 a = x1; //loss of data since a=6!
4 x1=a;
```

There is an explicit way to change the type: casting

```
1 double x1=6.28;
2 int a = 2
3 a = (int)x1; //loss of data, but no warning
4 x1=(double)2/3; //x1 is not zero
```

Precedence of operators

The ones we know so far

1. () brackets
2. + - uary plus/minus: (-1)
3. * / % binary operator a*b
4. - + binary operator a+b

```
1 -5 * 3 + 4 * 5. / 2.
2 ((-5)*3)+(4*5)/2.
```

Increment/decrement operators

Increment / decrement operators are unary operators that change the value of a variable by 1.

They can have postfix or prefix form

```
a++ //postfix
a--
++a //prefix
--a
```

```
int a = 1;
int b = a++; // stores 1+a (which is 2) to a
             // returns the value of a (which is 1)
             // After this line, b == 1 and a == 2
a = 1;
int c = ++a; // stores 1+a (which is 2) to a
             // returns 1+a (which is 2)
             // after this line, c == 2 and a == 2
```