Lecture 2

What we have seen so far

- Basic elements of C
- Comments: /**/, //
- #include <> - appending headers
- Functions
- semicolons ;
- Brackets (),

Program

```c
/* Description - not mandatory but polite*/
#include <stdio.h> //Preprocessor commands starting with a \# Note no semicolons ";", why?

int sum_ints(inta, intb); //Function prototypes, a promise to the compiler

int main() //The main function must be there
{
  int a=5, b=4;
  int c;
  c = sum_ints(a,b);
  printf("c=%d\n", c); // Print value of c, start new line
  return 0;
}

//Description of a function
int sum_ints(int a, int b)
{
  return a+b;
}
```

;

Semicolon ends the line

```
x1=(-b-sqrt(delta))/(2*a);
```

```
x1=(-b
-sqrt(
delta))
/
(2*a);
```

Computer Science I

What now?

- Simple data types with examples
- Arithmetic operations
- Precedence of operators
- Some fun with characters

Selected data types in C

- To declare variables
- Determine size in memory
- methods and operations possible
- Need format specifiers to print with *printf()*
- We limit ourself to integer, real numbers, characters, boolean and ... void
- The size of data types might vary with implementation, use *sizeof()*
- There is more ...

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:
  - %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

```c
#include <stdio.h>

int main()
{
   int a=5, b=4;
   printf("%d+%d=%d\n", a, b, a+b);
   printf("%d-%d=%d\n", a, b, a-b);
   printf("%d*%d=%d\n", a, b, a*b);
   printf("%d/%d=%d\n", a, b, a/b);
   printf("%d/%d=%d\n", b, a, b/a);
   printf("%d%%%d=%d\n", a, b, a%b);
}
```

- 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!

float, 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, double

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

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, double arithmetic

```c
#include <stdio.h>

int main()
{
  double a=5.0, b=4.0;
  printf("%lf+%lf=%lf\n", a, b, a+b);
  printf("%lf-%lf=%lf\n", a, b, a-b);
  printf("%lf*%lf=%lf\n", a, b, a*b);
  printf("%lf/%lf=%lf\n", a, b, a/b);
  printf("%lf/%lf=%lf\n", b, a, b/a);
}
```

- 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!

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

```
#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 (or 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 to **void** - it points to an addres, but does not specify the variable type - more later on
- keyword **void**
- 1B (?) - it has no size

```c
#include <stdio.h>

int main()
{
  printf("Storage␣size␣for␣void␣:␣%ld␣B␣\n", sizeof(void)); //should not work, works in gcc
  void a; //Not possible
  printf("␣%d␣\n", a);
}
```

Variables

Substitution:

```
int a,b; //declare two variables of type int
a=35; //a store value of 35
b=6; //b store 6
a=a+b;// perform addition in temporary space, copy on to a
```

An arithmetic expresion:

```
double x1; //declare a variable of type double
x1=(-b+sqrt(delta))/(2.0*a); //Perform RHS operation, write result
     to x1
```

Substitution

When using "=" sign variable on the LHS is assigned value of RHS. This does not necessarily means equality!
The RHS is calculated first and later the value is copied to the LHS.
Types of LHS and RHS should be the same.
Mixing of types should be avoided.

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

There is an explicit way to change the type: casting

```
double x1=6.28;
int a = 2
a = (int)x1; //loss of data, but no warning
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

```
-5 * 3 + 4 * 5. / 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
```