Lecture 2 Examples

October 21, 2019

1 Data types

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).

- Used to declare variables or define functions.
- Determine size the variable occupies in memory.
- Need format specifiers to print with *printf()*.

We will discuss some (of many) data types. Integers, Floats, Characters and bools.

1.1 int

Used for storing integers (2, 6, 676, -1000 are integers and 4.5 is not). Use **int** keyword to define. Nowdays ints occupy 4B.

Format specifiers for printf(): * %d %i - signed integer (%li %ld for long). * %o - Octal integer. * %x %X - Hex integer. * %u - unsigned integer.

Let's start with the following example:

In [52]:
```
#include <stdio.h>

int main()
{
    printf("Storage size for int : %ld B \n", sizeof(int)); // Prints the size of int

    int a = 20;
    printf( "in decimal representation a=%d \n", a);
    printf( "in octal representation a=%o \n", a);
    printf( "in hexadecimal representation a=%x \n", a);
}
```

```
Storage size for int : 4 B
in decimal representation a=20
in octal representation a=24
in hexadecimal representation a=14
```

There is a limit to the size of integer that can be stored before the 4B of memory designed to store the integer overflows. For 32 integers (4B is 32 b - as in bits) is int is in the range: (-2147483647 - 1, 2147483647)

```
In [11]: #include <stdio.h>

         int main()
         {
             printf("Storage size for int : %ld B \n", sizeof(int));

             int a = 320*350*360*555;
             printf( "a =%d \n ", a);
         }
```

```
/tmp/tmpd1jn53ux.c: In function main:
/tmp/tmpd1jn53ux.c:7:24: warning: integer overflow in expression [-Woverflow]
     int a = 320*350*360*555;
                         ^


Storage size for int : 4 B
a =902763520
```

- Note: We habe been warned by the compiler that our operation will overflow the int! It does not always has to be so outspoken!
- Note2: The acctual value should be: 22377600000

1.1.1 Operations:

We can perform arithmetic operations on integers. We cann add (+), substract (-), multiply (*) and divide (/). We can also request remainder of the divison with %. See an example:

```
In [1]: #include <stdio.h>

        int main()
        {
            int a, b; //uninitialized variables, have any value
            a = 5;
            b = 6;

            printf( "a=%d b=%d\n ", a, b);
            printf( "%d+%d = %d\n ", a, b, a+b);
            int c = a - b;
            printf( "%d-%d = %d\n ", a, b, c);

            c = a*b;
            printf( "%d*%d = %d\n ", a, b, c);
            printf( "%d/%d = %d\n ", a, b, a/b); //risk of loosing data
```

```
            printf( "1/2=%d\n", 1/2);

            a = 10;
            b = 7;
            printf( "%d %% %d = %d\n ", a, b, a%b);
        }
```

```
a=5 b=6
 5+6 = 11
 5-6 = -1
 5*6 = 30
 5/6 = 0
 1/2=0
10 % 7 = 3
```

You might have noticed that division gives strange results. That is 5/6=0! The reason is that the integer only stores the integral value of a number and for this case it would be 0. **So rmember 1/2 is 0**! since both 1 and two are integers.

1.2 Floats and doubles

Are used to store floating point numbers, such as 14.35 or 5.4 or any other. We will get to know two: * **floats** that occupy 4B * **doubles** that take 8B

To print them use: * %lf for doubles and %f for floats * %e or %E for scientyfic notation usefull for small and large values

```
In [25]: #include <stdio.h>

         int main()
         {
             printf("Storage size for int : %ld B \n", sizeof(double));
             double a = 45887654.87863587358635987365897;
             printf("a= %lf \n", a);
             printf("a= %e \n", a);
             double b = 0.00000000000000002374274278;
             printf("a= %lf \n", b);
             printf("a= %e \n", b);
         }
```

```
Storage size for int : 8 B
a= 45887654.878636
a= 4.588765e+07
a= 0.000000
a= 2.374274e-15
```

Similarly to ints floating point numbers allow for addition (+), substraction (-), multiplication (*) and division (/):

```
In [2]: #include <stdio.h>

        int main()
        {
            double a = 4.56, b=9.345;

            printf( "a=%lf b=%lf\n", 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", a, b, a/b);
        }
```

```
a=4.560000 b=9.345000
4.560000+9.345000 = 13.905000
4.560000-9.345000 = -4.785000
4.560000*9.345000 = 42.613200
4.560000/9.345000 = 0.487961
```

1.3 Data casting

Now that we know two data types, we can try to change the way data is treated by means of casting.

Consider the following:

```
In [10]: #include <stdio.h>

         int main()
         {
             int a = 4, b=9;

             printf( "%d %d %d\n", a, b, a/b);
             printf( "%d %d %lf\n", a, b, (double)a/b); // cast a into a double
             printf( "%d\n", 1/2);
             printf( "%lf\n", 1./2.);
         }
```

```
4 9 0
4 9 0.444444
0
0.500000
```

What happens there?

1) So in line 7 we try to print a result of division of two ints. Since both operands are of type int the result is assumed to be of the same type. The operation cuts out any decimal part and only the integral part is left. That is why we see a zero.

2) In line 8 we perform an explicit cast by adding (double) to one of the variables (we write (double)a/b) this informs the complier that the result should be treated as a double. This way we see a rpoper result.
3) In lines 9 and 10 we print the result of 1/2, please note that adding a '.' changes the type from int to double! so 1 - an integer, but 1. - a double!

1.4 Characters

Our next data type are characters. Those are used to store just that, characters. Those are 1B in size and can be interpreted as integers or symbols. Have a look here: ASCI table for an ASCI table, that is a translator from integer to a character kind of thing.

(ASCI - American Standard Code for Information Interchange)

In [48]:
```c
#include <stdio.h>

int main()
{
    char a = 'b';
    printf("%c\n", a);
    printf("%d\n", a);

    a = 52;
    printf("%c\n", a);
    printf("%d\n", a);

    a = 32;
    printf("%c\n", a);
    printf("%d\n", a);
}
```

```
b
98
4
52

32
```

1.5 bool

Used to represent logical value of *true* and *false*. Introduced into C in C99 standard. in order to use *stdbool.h* needs to be included. Use %d format specifier to print.

The example below illustrates declaration of a boolean variable, assigment and printing.

In [38]:
```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 );
}
```

```
Storage size for char : 1 B
a=1
a=0
```

1.6 Void

Void is a special type that represents *nothing*. That is a lot, I hope you agree? It is used to distinguish functions that return no value and therfore are *void*. Variables of this type can not be declared! We will be having a look at *void* type when we get to work with functions and pointers.

In [40]:
```
#include <stdio.h>

void main(){
    printf( " Storage size for void : %ld B \n" , sizeof ( void ) ) ;
    //void a ; // variables of type void are not allowed
    //printf ( " %d \n" , a ) ;
}
```

```
 Storage size for void : 1 B


[C kernel] Executable exited with code 30
```

1.7 Data assigment, data loss and casting

Mixing of types should be, if possible avoided. If necessary it should be done with care.
Here we interpret a doble as an in tne than back again. As a result we suffer data loss since the output vale is not what it originally was. It is possible that your compiler will complain with a warning.

In [41]:
```
#include <stdio.h>

int main(){
    double x1 = 6.28;
    int a = 2;
    printf("%d %lf \n", a, x1);
    a = x1; // loss of data since a =6!
    printf("%d %lf \n", a, x1);
    x1 = a;
    printf("%d %lf \n", a, x1);
}
```

```
2 6.280000
6 6.280000
6 6.000000
```

And here we do the same, but "more consciously", i.e we perform a data cast we have seen earlier.

```
In [42]: #include <stdio.h>

         int main(){
             double x1 = 6.28;
             int a = 2;
             a = ( int ) x1; // loss of data , but no warning
             printf("%d %lf \n", a, x1);
             x1 = (double) 2/3; // x1 is not zero since I use a cast
             printf("%d %lf \n", a, x1);
         }
```

```
6 6.280000
6 0.666667
```

1.8 Precedence of operators

It is simple. More less what you know from your math class about the order of wxecution. So brackets () before multiplication, sumation/substraction and so on. There can be some new concepts on the way though.

```
In [44]: #include <stdio.h>

         int main(){
             double a = 6, b = 9;
             double c = (a+b) / a + b / (a * b);
             //double c = (a+b) / a + b / a * b; // it is different!
             printf("%lf \n", c);
         }
```

```
2.666667
```

1.9 Increment / decrement

C offers some operators that are good to know. Incrementation ++ and decrementation -- operators increase or decrease *int* value by 1. Consider the folloing:

```
In [45]: #include <stdio.h>

         int main(){
```

```
        int a = 1;
        printf("%d \n", a);
        ++a;
        printf("%d \n", a);
        ++a;
        printf("%d \n", a);
        ++a;
        printf("%d \n", a);
        ++a; // a = a + 1;
        printf("%d \n", a);

        --a;
        printf("%d \n", a);
        --a;
        printf("%d \n", a);
        --a;
        printf("%d \n", a);
        --a;
        printf("%d \n", a);
        --a;
        printf("%d \n", a);
        --a;
        printf("%d \n", a);
    }
```

```
1
2
3
4
5
4
3
2
1
0
-1
```

and:

```
In [46]: #include <stdio.h>

    int main(){
        int a = 1;
        printf("%d \n", a);
        a++;
        printf("%d \n", a);
        a++;
```

```
        printf("%d \n", a);

        a--;
        printf("%d \n", a);
        a--;
        printf("%d \n", a);
    }
```

```
1
2
3
2
1
```

There are two versions of those operators and apparently the two do the same. But there is a difference. The ++a is called prefix and a++ the postfix. The first is performed first and than passed for possible assigment. The second is first passed for assigment and only than incremented. Consider the two examples and note the resulting values:

```
In [47]: #include <stdio.h>

    int main(){
        int a = 1;
        int b = ++a; // first increment than copy
        printf("%d %d\n", a, b);
    }
```

```
2 2
```

```
In [48]: #include <stdio.h>

    int main(){
        int a = 1;
        int b = a++; // first copy than increment
        printf("%d %d\n", a, b);
    }
```

```
2 1
```

We note that it is better to use the prefix version sice it only invikes coping the value, while the postfix requiers first to create a copy that is passed and than the increamentation, which in case of composite data types can be expensive.

1.10 Compound Assignment Operators += -= *= /= Onother type of operators are the compound operators. Those are used to replace operation with the assigment. Such as a = a + b. Consider the following:

```
In [49]: #include <stdio.h>
```

```
int main(){
    int a = 1;
    int b = 2;
    a += b; // a = a + b;
    printf("%d %d\n", a, b);
}
```

```
3 2
```

In [50]:
```
//%cflags:-lm

#include <stdio.h>

int main(){
    double a = 3.14;
    double b = 2.73;
    a *= b; // a = a * b;
    printf("%lf %lf\n", a, b);
}
```

```
8.572200 2.730000
```