Introduction to C

Programming Workshop in C (67316)
Fall 2017
Lecture 2
26.10.2017

C – 32 Keywords only

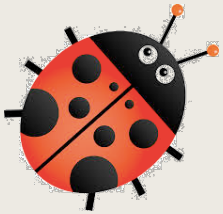
Numeric data types in C

```
char c = 'A';
short s = 0;
int x = 1;
long y = 9;
```

```
unsigned char c = 'A';
unsigned short s = 0;
unsigned int x = 1;
unsigned long y = 9;
```

```
float x = 0.0;
double y = 1.0;
```

```
unsigned char x = 0;
x = x - 1; // x = 255
```



overflow

Operator Types in C

- Arithmetic operators: + - * / %
- Increment and decrement operators ++ --
- Relational operators: < <= > >= == !=
- Logical operators: && || !
- Bitwise operators: & | ^ << >> -
- Assignment operators: += -= *= /= %= ...

- can be binary (such as +), unary (such as ++), or ternary

Arithmetic operators

- **The order of evaluation is as in algebraic expressions:**
  - **brackets first, followed by * and /, followed by +**
  - **from left to right**

Operator	Meaning	Examples
+	addition	`int x = y + 3;`
-	subtraction	`int x = y - 3;`
*	multiplication	`float z = x * y;`

/	division	`float x = 3 / 2; // = 1`
		`float y = 3.0 / 2; // = 1.5`
		`int z = 3.0 / 2; // = 1`

%	remainder	`int x = 3 % 2;`

Truncation and type casting

```
int x = 75;
int y = 100;
float z = x / y; // = 0.0
float z = (float) x / y; // = 0.75
int z = (float) x / y; // = 0
```

cast x to float

truncation towards zero

Increment and decrement operators

- **++ and -- operators are shortcuts:**

`x++;`	`x = x + 1;`	
`y = x++;`	`y = x; x = x + 1;`	x is evaluated **before** it is increment
`y = ++x;`	`x = x + 1; y = x;`	x is evaluated **after** it is incremented

Statements

Statements - conditional

```
if (expression) {
 // ... (single statement or block)
} else if (expression) {
 // ...
} else {
 // ...
}

switch (integer value) { ... }
// find by yourself later
```

Statements - conditional

A programmer goes to the grocery store and his wife tells him:

- "Buy a gallon of milk, and if there are eggs, buy a dozen."

So he buys everything, and drives back to his house. Upon arrival, his wife angrily asks him:

- "Why did you get 13 gallons of milk?"

The programmer says:

- "There were eggs!"

Statements - loops

```
// for( initial ; test condition ; update step )
int i, j; // in ANSI C you can't declare inside the for loop!
for (i=0, j=0; (i<10 && j<5); i++, j+=2) {
   // ...
}
```

```
while (condition) {
   // ...
}
```

```
do {
   // ...
} while (condition);
```

```
break; // exit loop
continue; // begin next iteration
```

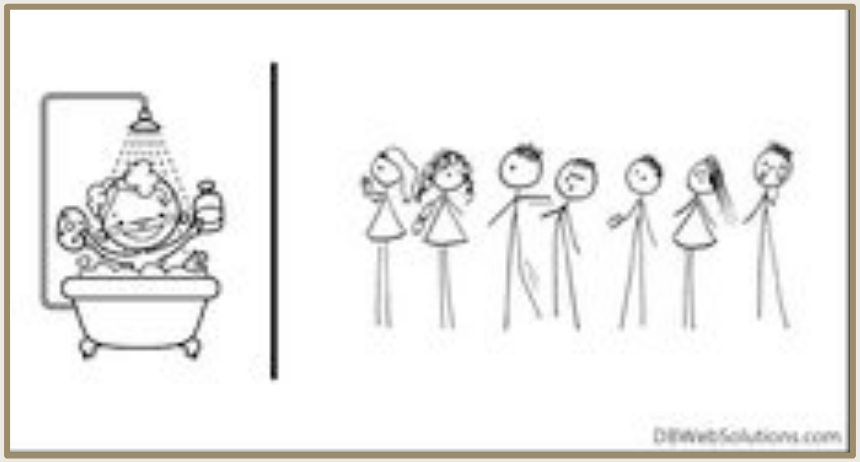
Statements - loops

The Programmer got stuck in the shower, why ?

Because the instructions on the shampoo bottle said

`Lather, Rinse, Repeat`

```
while (shampoo) {
    Lather;
    Rinse;
    Repeat;
}
```



Loop program

```
#include <stdio.h>
int main()
{
   int i;      // declares i as an integer
   int j = 0; // declares j as an integer,
              // and initializes it to 0

   for( i = 0; i < 10; i++ )
   {
      j += i; // shorthand to: j = j + i
      printf("%d %d %d\n", i, j, (i*(i+1))/2);
   }
   return 0;
}
```

Running…

```
> gcc -Wextra loop.c -o loop
> loop
```

```
0 0 0
1 1 1
2 3 3
3 6 6
4 10 10
5 15 15
6 21 21
7 28 28
8 36 36
9 45 45
```

Input/Output

Character Input/Output

```
#include <stdio.h>
int main()
{
    int c;
    while( (c = getchar()) != EOF )
    {
        putchar(c);
    }
    return 0;
}
```

gets a character from stdin

#define macro

```
#include <stdio.h>
#define NUM_OF_LINES 10
int main()
{
    int n = 0;
    int c;
    while(((c=getchar()) != EOF) &&
          (n < NUM_OF_LINES) )
    {
        putchar(c);
        if( c == '\n' )
            n++;
    }
    return 0;
}
```

AND operator

How many iterations are performed?

General Input/Output

```
#include <stdio.h>
int main()
{
   int n;
   float q;
   double w;

   printf("Please enter an int, a float
           and a double\n");
   scanf("%d %f %lf", &n, &q, &w);
   printf("I got: n=%d, q=%f, w=%lf", n, q, w);

   return 0;
}
```

Functions

Functions

C allows to define functions

Syntax:

Return type

Parameter declaration

```
int power( int a, int b )
{
  // ...
  return 7;
}
```

Return statement

Procedures

Functions that return `void`

```
void power( int a, int b )
{
  // ...
  return;
}
```

Return w/o value (optional)

Example – printing powers

```c
#include <stdio.h>

int power( int base, int n )
{
   int i, p;

   p = 1;
   for( i = 0; i < n; i++ )
   {
      p = p * base;
   }
   return p;
}
```

```c
int main()
{
   int i;

   for( i = 0; i < 10; i++ )
   {
      printf("%d %d %d\n",
         i,
         power(2,i),
         power(-3,i) );
   }
   return 0;
}
```

Functions Declaration

```
void funcA()
{
    ...
}
void funcB()
{
    funcA();
}
void funcC()
{
    funcB();
    funcA();
    funcB();
}
```

```
void funcA()
{
    ...
}
void funcB()
{
    funcC();
}
void funcC()
{
    funcB();
}
```

"Rule 1": A function "knows" only functions which were declared above it.

Error:
funcC is not known yet.

Forward Declaration

Amendment to “Rule 1” : use **forward declarations**

```
void funcC(int param);
void funcA()
{
}
void funcB()
{
   funcC(7);
}
void funcC(int param)
{
}
```

Functions declaration

Declaration tells the compiler function **name and return type**

```
// the following 3 declarations are legit:
int foo(int a); // return int accepts int
int foo(int);   // return int accepts int
int foo();      // return int accepts unspecified
                //                    parameters
int main() {
   foo(5);
   return 0;
}
int foo(int a) { // actual definition of `foo`
   return a;
}
```

Functions declaration

```
int foo(int);    // return int accepts int
void foo(int);   // return void accepts int
```

error: conflicting types for 'foo'

```
int main() {
    foo(5);
    return 0;
}
int foo(int a) { // actual definition of `foo`
    return a;
}
```

Functions declaration

```
int foo(int);       // return int accepts int
int foo(int, int); // return int accepts int, int
```

error: conflicting types for 'foo'

```
int main() {
   foo(5);
   return 0;
}
int foo(int a) { // actual definition of `foo`
   return a;
}
```

NO function overloading

A function may have several declarations,
but only one definition

→ The following code **will not compile**

```
int foo(int a) {return a;}
int foo(int a) {return a;}
error: redefinition of 'foo'

int main() {
  foo(5);
  return 0;
}
```

Functions declaration

```
int foo(); // return int accepts unspecified
           //                    parameters

int main() {
   foo(5, 6, 7); // strange, but this is OK
   return 0;
}
int foo(int a) { // actual definition of `foo`
   return a;
}
```

What are the maximal values of standard variables?

```
#include <stdio.h>

int power( int base, int n );

int main()
{
  // Basic primitive types:
  printf("Size of char %lu\n", sizeof(char));
  printf("Size of short %lu\n", sizeof(short));
  printf("Size of int %lu\n", sizeof(int));
  printf("Size of long %lu\n", sizeof(long));
  return 0;
}

int power( int base, int n ) {
   int i, p = 1;
   for( i = 0; i < n; i++ ) {
      p = p * base;
   }
   return p;
}
```

```
power(2, 8*sizeof(char))-1
```

What are the maximal values of standard variables?

```
#include <stdio.h>

int power( int base, int n );

int main()
{
  // Basic primitive types:
  printf(”Max char value %d\n", power(2, 8*sizeof(char)) - 1);
  printf(”Max short value %d\n", power(2, 8*sizeof(short)) - 1);
  printf(“Max int value %d\n", power(2, 8*sizeof(int)) - 1);
  printf(“Max long value %d\n", power(2, 8*sizeof(long)) - 1);
  return 0;
}

int power( int base, int n ) {
   int i, p = 1;
   for( i = 0; i < n; i++ ) {
      p = p * base;
   }
   return p;
}
```

What are the maximal values of standard variables?

```
#include <stdio.h>

long power( int base, int n );

int main()
{
  // Basic primitive types:
  printf(”Max char value %d\n", power(2, 8*sizeof(char)-1) - 1);
  printf(”Max short value %d\n", power(2, 8*sizeof(short)-1) - 1);
  printf(“Max int value %d\n", power(2, 8*sizeof(int)-1) - 1);
  printf(“Max long value %d\n", power(2, 8*sizeof(long)-1) - 1);
  return 0;
}

long power( int base, int n ) {
   int i,
   long p = 1;
   for( i = 0; i < n; i++ ) {
      p = p * base;
   }
   return p;
}
```

one bit for sign