IP of the Max-Planck-Institute for Physics of Complex Systems (mpipks),
Nöthnitzer Straße 38, 01187 Dresden

MPI PKS C++ Lecture

Bennet Becker

2019-09-09

MPI PKS

Program Structure

Control Structure

Control Structure i

- a simple statement is each individual instruction of a program. like expressions seen before.
- statements are always terminated with a semicolon (most common error source), and executed in the same order in which they appear in a program*.
- not limited to such linear sequences of statements
- $\rightarrow$ control-flow statements. they require a generic (sub)statement as part of its syntax. single statements, terminated with semicolon or compound-statement, enclosed with curly brackets

Control Structure ii

```
// single statement
statement;

// compound statement
{
        statement1;
        statement2;
        ...
        statementN;
}
```

Selection statements - if and else i

```
// simple version: single statement
if (condition) statement;

if (condition)
        statement1;
else
        statement2;
```

Selection statements - if and else ii

```
// with compount statement and alternatives
if (condition) {
        statement1;
        statement2;
        ...
        statementN;
}
```

Selection statements - if and else iii

```
if (condition) {
        statement11;
        statement12;
        ...
        statement1N;
} else {
        statement21;
        statement22;
        ...
        statement2N;
}
```

Selection statements - if and else iv

```
if (condition) {
        statement11;
        statement12;
        ...
        statement1N;
} else if (condition2) {
        statement21;
        statement22;
        ...
        statement2N;
} else {
        statement31;
        statement32;
        ...
        statement3N;
}
```

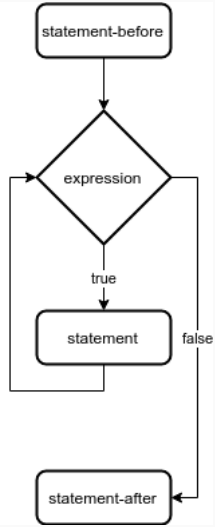
Selection statements - switch

```
// switch with many multiple options
switch (varibale) {
        case value1:
                statement1;
                break;
        case value2:
                statement2;
                break;
        ...
        case valueN:
                statementN;
                break;
        default:
                statement;
}
```

break is important. because switch jumps to according `case value:` and executes all statement until end or next break (which can be useful if used intentionally).

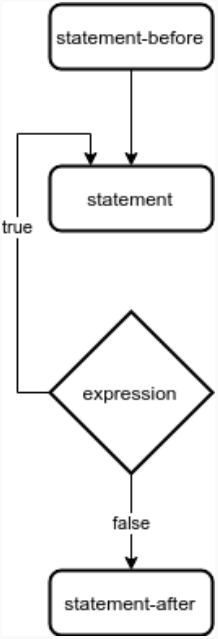
Iteration statements - while

```
while (expression) {
        statements;
}
```



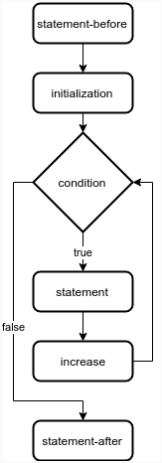
Iteration statements - do-while

```
do {
        statements;
} while (expression);
```



Iteration statements - for loop

```
for (initialzation; condition;
 ↪  increase) {
        statement;
}
```



Iteration statements - Range based for loop

- `string str; for (char c : str){ }`
- `int arr[]; for(int i : arr){ }`
- `for (auto i : var){ }`

Jump statements i

- `break` : break/abort a loop even if continuation condition is still fulfilled
- `continue` : skip rest of this loop iteration and continue with next iteration
- `goto` : **NON EXISTENT!**

Jump statements ii

Jump statements iii

TASK: Greatest common divisor

Write a programm, that for two positive Numbers, calculates thier Greatest common divisor.
Begin like this

```
#include <iostream>
#include <cstdlib>

int main(int argc, const char** argv){
        int n1, n2;

        std::cout << "Enter two numbers: ";
        std::cin >> n1 >> n2;

        //... your code here...

        return 0;
}
```

Possible solutions

```
#include <iostream>
#include <cstdlib>


int main(int argc, const char** argv){
        int n1, n2;

        std::cout << "Enter two numbers: ";
        std::cin >> n1 >> n2;

        while(n1 != n2)        {
                if(n1 > n2) {
                        n1 -= n2;
                } else {
                        n2 -= n1;
                }
        }

        std::cout << "GCD = " << n1;
        return 0;
}
```

Possible solutions

```
#include <iostream>
#include <cstdlib>

#define ABS(x) x < 0 ? x * -1 : x

int main(int argc, const char** argv){
        int a, b, h;

        std::cout << "Enter two numbers: ";
        std::cin >> a >> b;

        if (a == 0){
                std::cout << "GCD = " << ABS(b);
                return 0;
        }
        if (b == 0) {
                std::cout << "GCD = " << ABS(a);
                return 0;
        }

        do {
                h = a % b;
                a = b;
                b = h;
        } while (b != 0);

        std::cout << "GCD = " << ABS(a);
        return 0;
}
```

Possible solutions

```
#include <iostream>
#define EVEN(x) (x % 2) == 0

int main(int argc, const char** argv){
        int a,b;
        int d;

        std::cout << "Enter two numbers: ";
        std::cin >> a >> b;

        while(EVEN(a) && EVEN(b)){
                a >>= 1; // a = a / 2
                b >>= 1; // b = b / 2
                d++;
        }
        while(a != b){
                if(EVEN(a)){
                        a >>= 1;
                } else if (EVEN(b)){
                        b >>= 1;
                } else if (a > b) {
                        a = (a - b) >> 1;
                } else {
                        b = (b - a) >> 1;
                }
        }

        std::cout << (a << d); // a * 2 ^ d
        return 0;
}
```

Functions

Functions i

- Functions allow to structure programs in segments of code to perform individual tasks

```
1       type_t1 func_name(type_t2 param1, type_t3 param2,
        ↪  ...){
2       type_t1 val;
3       // ...
4
5       return val;
6       }
```

Functions ii

- `type_t1` is the type of the value returned by the function
- `func_name` identifier by which the function is called
- `param1` - `paramN` : parameters (with type). as many as needed

- Functions with no type. -> `void function(type parm1, ...) { }`
- Function with no parameters -> `type function(void) { }` or `type function() { }`
- Default paramenters ->
  `type function(type1 param1, type2 param2 = value) { }`
  (parameter `param2` can now be omitted on function-call)
- parentheses required on function call! to differentiate a function from a variable

main() - Function

- return value of `main()`
  - main returns `int`
  - value `0` is interpreted by the environment as that the program ended successfully
  - but `main` may also return other values. some environments give access to that value to the caller in some way, although this behavior is not required nor necessarily portable between platforms
  - The values for main that are guaranteed to be interpreted in the same way on all platforms are:
    - `0` : The program was successful
    - `EXIT_SUCCESS` : The program was successful (same as above). This value is defined in header `#include <cstdlib>` .
    - `EXIT_FAILURE` : The program failed. This value is defined in header `#include <cstdlib>` .

Functions

- Declaring functions by writing them without body:
  `type_t1 func_name(type_t2 param1, type_t3 param2, ...);`
  - functions as well as variables can not be used before declared
  - So you can declare functions before defining them to overcome scoping difficulties

TASK: least-common-multiple

Rewrite the GCD-Program to use a function. Use this function to calculate the least-common-multiple.

```
#include <iostream>

// ...

int main (int argc, const char** argv){
        int a,b;

        std::cout << "Enter two numbers: ";
        std::cin >> a >> b;

        std::cout << "GCD = " << /* ... */ << std::endl
                  << "LCM = " << /* ... */ << std::endl;
}

// ...
```

Possible Solution i

```
#include <iostream>

int gcd(int, int);
int lcm(int, int);

int main (int argc, const char** argv){
        int a,b;

        std::cout << "Enter two numbers: ";
        std::cin >> a >> b;

        std::cout << "GCD = " << gcd(a,b) << std::endl
                  << "LCM = " << lcm(a,b) << std::endl;
}
```

Possible Solution ii

```
int gcd(int a, int b){
        if (a == 0){
                return std::abs(b);
        }
        if (b == 0) {
                return std::abs(a);
        }

        do {
                h = a % b;
                a = b;
                b = h;
        } while (b != 0);

        return std::abs(a);
}

int lcm(int a, int b){
        return std::abs(a * b / gcd(a,b));
}
```