

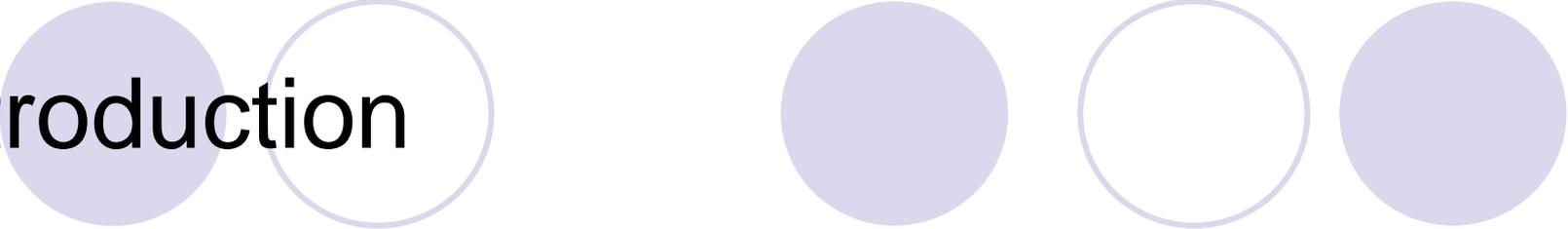
Computer Science

Control flow

Control Flow Statements in C language

- Introduction
- `if-else`
- `switch`
- `while`
- `for`
- `do-while`
- `break`
- `continue`
- `return`
- `goto`

Introduction



- **Control flow statements** specify the order in which computations are performed
- Different types
 - **Conditionals:** Take a decision among two or more options depending on the evaluation of a condition.
 - `if else` and `switch`
 - **Loops:** Iterations of operations (with condition evaluation)
 - `for`, `while` and `do-while`
 - **Jump:** They change unconditionally the order of execution.
 - `continue`, `break`, `return` and `goto`
 - **Labels:** Used to identify lines in a program.
 - `case`, `default` and `«label:»`

if-else statement (I)

```
if (expr) stat1;  
    else stat2;
```

- If `expr` is true then `stat1` is processed
- If `expr` is false, `stat2` is processed
- `expr` is true if its value is different than zero
- `else stat2;` is optional

if-else statement (II)

- `stat1` and `stat2` can be blocks of sentences between brackets

```
if (expr)
{
    /* Block of sentences 1 */
}
else
{
    /* Block of sentences 2 */
}
```

- Different `if-else` blocks can be grouped with brackets

if-else statement (III)

```
if (expr1)
{
    if (expr2)
        if (expr3) stat31;
        else stat32;
}
else stat2;
```

- stat31 is processed if **expr1, expr2 and expr3 are true**
- stat32 is processed if **expr1, expr2 are true and expr3 is false**
- stat2 is processed if **expr1 is false (without considering expr2 and expr3)**

if-else statement (IV)

- **Nested if-else statements**

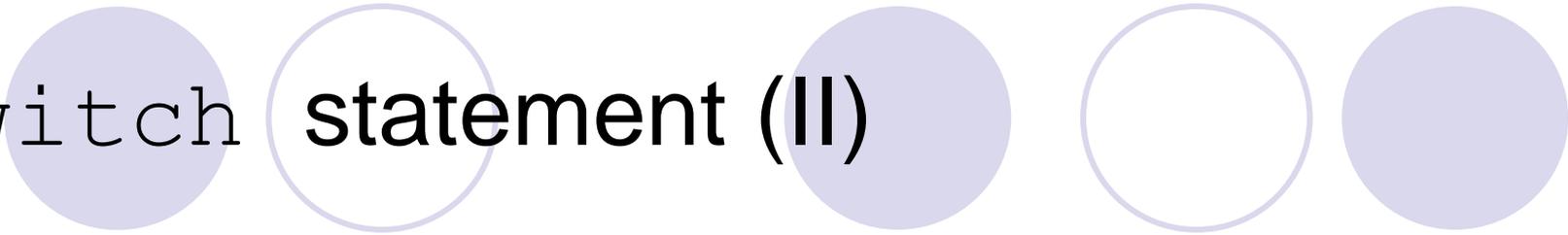
- Brackets determine priority among `if` and `else`
- Without brackets
 - Each `else` is associated with the closest `if`
 - Each block of statements is processed independently

```
if (expr1) stat1;
else if (expr2) stat2;
else if (expr3) stat3;
...
else if (exprN) statN;
else statN+1;
```

- `statN` is processed just if `exprN` is true
- `statN+1` is processed just if none of the previous statements have been processed

switch statement (I)

```
switch (expr)
{
    case const-expr1:
        /* Statement block 1 */
        break;
    case const-expr2:
        /* Statement block 2 */
        break;
    ...
    case const-exprN:
        /* Statement block N */
        break;
    default:
        /* Statement block N+1 */
        break;
}
```



switch statement (II)

- `switch` is a multi-way decision test whether an expression matches a number of *constant integers*
 - Brackets are needed
 - case number is unlimited
 - `default` is optional
 - `break` causes an immediate exit from the `switch`
- `expr` is evaluated and comparison with `const-expr` in each case starts
 - If any matches, all statements are executed until a *break* or the end of the `switch`
 - If none matches `default` statements are executed (if they exist) until a *break* or the end of the `switch`

switch statement (III)

```
#include <stdio.h>
int main ()
{
    char grade = 'B';

    switch(grade)
    {
        case 'A' :
            printf("Excellent!\n" );
            break;
        case 'B' :
        case 'C' :
            printf("Well done\n" );
            break;
        case 'D' :
            printf("You passed\n" );
            break;
        case 'F' :
            printf("Better try again\n" );
            break;
        default :
            printf("Invalid grade\n" ); }

    return 0;
}
```

while statement



```
while (expr) stat;
```

```
while (expr)
{
    stat; /* block of statements */
}
```

- If `expr` is true, `stat` is processed
- After execution `expr` is evaluated again
- If false, exit from the `while`
- WARNING: if `expr` doesn't change its value, an infinite loop can be created

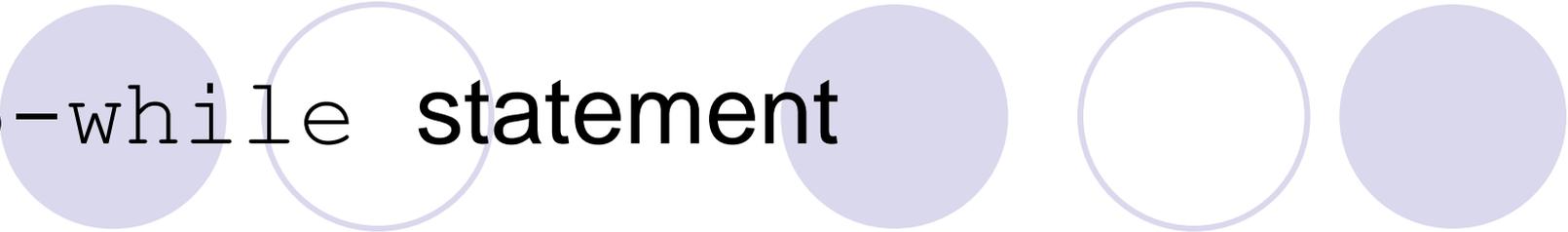
for statement

```
for (init_expr; cond_expr; update_expr) statement;
```

```
for (init_expr; cond_expr; update_expr)
{
    statement; /* Statement block */
}
```

- `init_expr` is an expression that assigns values to one or more variables
- `cond_expr` evaluates an expression: if true statement is processed. If false loop is finished
- `update_expr` are statements that are processed after statement. Typically update the value of the control variable

Example: `for (i=0; i<n; i++) printf("i= %d", i);`



do-while statement

```
do statement;  
while (expr);
```

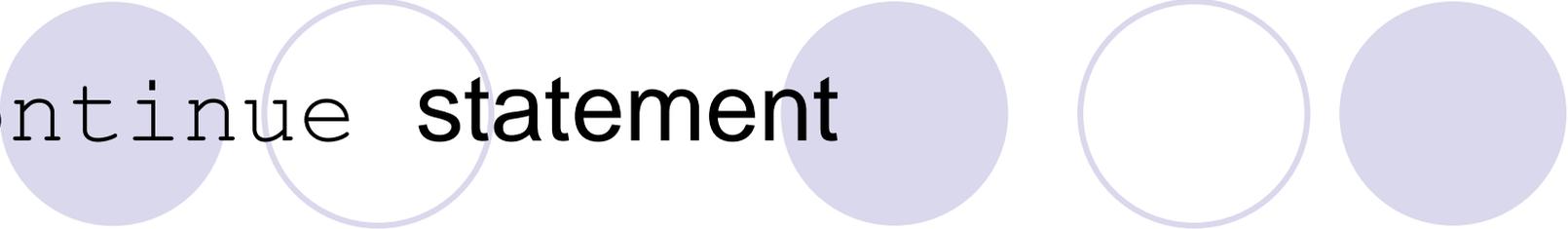
```
do  
{  
    statement ; /* Block of statements */  
} while (expr);
```

- After executing `statement`, `expr` is evaluated, and, if true, `statement` is executed again.
- If `expr` is false, exit from the loop.
- **WARNING:** If `expr` does not change its value within the loop, an infinite loop can be created.

break statement



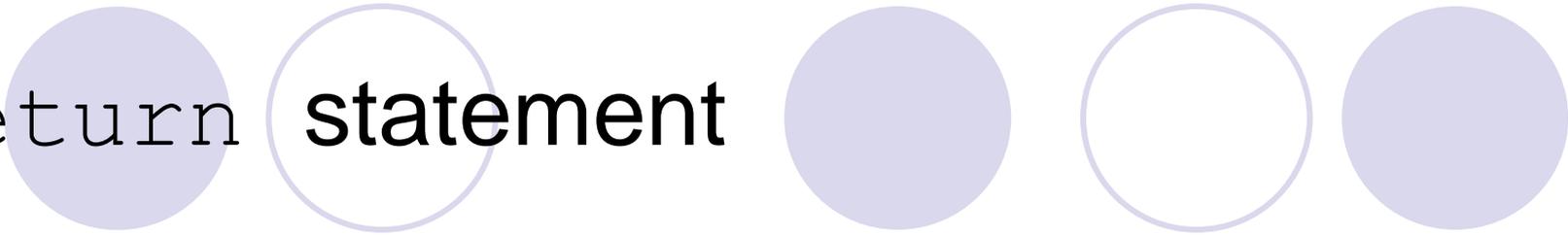
- `break` allows to exit immediately from the execution of `switch`, `while`, `do-while`, `for`, independently of any other condition.
- In nested loops, `break` exits just from the inner loop in which is placed.



continue statement

- `continue` forces a new iteration in the loop, ignoring the following statements until the end of the loop
 - With `while` and `do-while`, jumps to condition evaluation
 - With `for`, jumps to update and condition
- In nested loops, `continue` exits just applies to the inner loop where is placed

return statement



- `return` ends a function, returning control to the point of the program where it was called

```
return expr
```

- The value of `expr` will be returned to the program
- It must be of the type declared in the function
- Function end bracket «`}`» is equivalent to `return` without `expr`, and it is used with functions that does not return any value (equivalent to `return 0`)

goto statement

- goto is an unconditional jump

- **ABSOLUTELY NOT RECOMMENDED**

```
...  
label:  
...  
goto label;  
...
```

- «label:» is a line identifier.
- It can be in any part of the program