

Introduction to Programming and Computational Physics

Lecture 3

Operators
Selection

Operators

double **sqrt** (double) (-> math.h, math.c)
output input

arithmetic operators (basic C library)

int + (int, int)	float + (float, float)
int / (int, int)	float / (float, float)
void = (&int, int)	void = (&float, float)

what if...

int a; float b;	<i>usually</i> a is promoted to float
float c = a+b;	and + (float, float) called

Increment and decrement operators

<code>n++</code>	}	The same as <code>n=n+1</code>	<code>++ (&int)</code>
<code>++n</code>			

<code>n--</code>	}	The same as <code>n=n-1</code>
<code>--n</code>		

The expression `++n` increments `n` *before* its value is used, while `n++` increments `n` *after* its value has been used.

If `n` is 5, then

`x = ++n` sets `x` to 6 but,

`x = n++` sets `x` to 5

In both cases `n` becomes 6

Assignment operators

$x+=2$

The same as **$x=x+2$**

$x-=y$

The same as **$x=x-y$**

$x*=y+1$

The same as **$x=x*(y+1)$**

$x/=y$

The same as **$x=x/y$**

Relational operators

hierarchy



>	>=	<	<=
==		!=	

The result can be “true” or “false” and the returned value is 1 (true) or 0 (false)

`int >(int, int)`

```
int a = 10, b = 20;
```

```
int r1 = a<b; // r1 takes value 1
```

```
int r2 = a>b; // r2 takes value 0
```

```
int r3 = a==b; // r3 takes value 0
```

```
int r4 = a!=b; // r4 takes value 1
```

Relational and logical operators

hierarchy ↓	! (NOT)	<code>int !(int)</code>
	> >= < <=	
	== !=	} relational
	&& (AND)	<code>int &&(int, int)</code>
	 (OR)	<code>int (int, int)</code>

The value "0" is assumed as false, any other value as true.

```
int a = 0, b = 10;
```

```
int r1 = a&&b; // r1 takes value 0
```

```
int r2 = a||b; // r2 takes value 1
```

```
int r3 = !b; // r3 takes value 0
```

```
int r4 = !!b; // r4 takes value 1
```

```
int r5 = a<=b || !b      int r5 = ((a<=b) || (!b))
```

Truth table

Ex1	Ex2	Ex1 & Ex2	Ex1 Ex2	!Ex1
F	F	F	F	T
F	T	F	T	T
T	F	F	T	F
T	T	T	T	F

Conditional structures

The C language has two conditional structures:

- **if - else (...else if)**
- **switch**

if – else

```
if (a>b) printf(“%f is greater than %f”,a,b);
```

Brackets are not needed for a single statement but probably helpful to avoid mistakes.

```
if (a>b) //no semicolon here!!  
{  
    printf(“%f is greater than %f”,a,b);  
    ...  
}
```

Combining relational and logical operators...

```
if (a<b&&b<c) ...           if((a<b)&&(b<c))...
```

if – else

```
if (a>b)
{
    max_ab = a;
    printf("a is the maximum and its value is %f",a);
}
else
{
    max_ab = b;
    printf("b is the maximum and its value is %f",b);
}
```

if – else

```
if (a=b)
    printf("a and b are the same");
else
    printf("a and b are different");
```

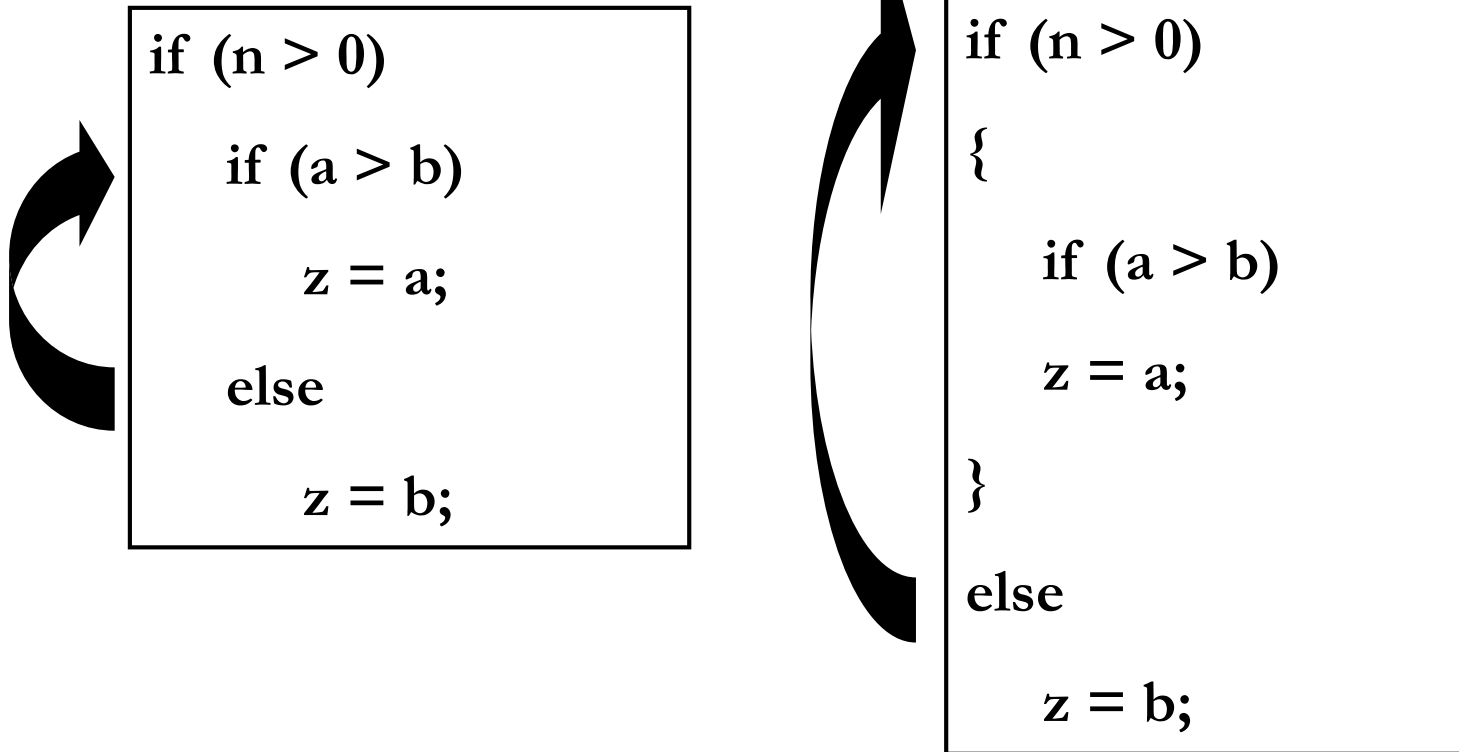
WRONG !!!

It will execute the statement if b is not zero

```
if (a==b)
    printf("a and b are the same");
else
    printf("a and b are different");
```

CORRECT !!!

if – else (nested)



Because the **else** part of an **if-else** is optional, there is an ambiguity when an **else** is omitted in a nested **if** sequence. The C language associates the **else** to the closest **if**. The usage of additional `{ }` could be anyway suggested to avoid confusion

if – else ... else if

When the number of alternatives is bigger than 2 the **if-else** can be extended by using the **else if** option

```
if (expression)
    statement ;
else if (expression)
    statement ;
else if (expression)
    statement ;
else
    statement ;
```

The expressions are evaluated in sequence. The last **else** is optional

```
if (n>0)
    printf("n is positive");
else if (n==0)
    printf("n is null");
else
    printf("n is negative");
```

Switch

```
switch(expression)
{
    case const1 :
        statement ;

    ...

    break;

    case const2 :
        statement ;

    ...

    break;

    default :
        statement ;

    ...

}
```

The expression must be an integer or a char

The (optional) **break** command brings outside the **switch** block

The statements following the (optional) **default** command are executed when none of the **case** is fulfilled

Switch

```
#include <stdio.h>

int main()
{
    int x;
    printf("\nEnter a number between zero and three: ");
    scanf("%d",&x);

    switch(x)
    {
        case 0:
            printf("you wrote zero\n\n");
            break;
        case 1:
            printf("you wrote one\n\n");
            break;
        case 2:
            printf("you wrote two\n\n");
            break;
        case 3:
            printf("you wrote three\n\n");
            break;
        default:
            printf("your number is not between zero and three\n\n");
            break;
    }
    return 0;
}
```