

Elements of C in C++

Types and Control Statements

data types
if else statement
for loops

Simulations

random numbers
rolling a die

Functions and Pointers

defining a function
call by value and call by reference
pointers and references

- 1 **Types and Control Statements**
 - data types
 - if else statement
 - for loops
- 2 **Simulations**
 - random numbers
 - rolling a die
- 3 **Functions and Pointers**
 - defining a function
 - call by value and call by reference
 - pointers and references

MCS 360 Lecture 2
Introduction to Data Structures
Jan Verschelde, 25 August 2010

Introduction to Data Structures

Types and Control Statements

data types

if else statement
for loops

Simulations

random numbers
rolling a die

Functions and Pointers

defining a function
call by value and call
by reference
pointers and
references

1 Types and Control Statements

data types

if else statement

for loops

2 Simulations

random numbers

rolling a die

3 Functions and Pointers

defining a function

call by value and call by reference

pointers and references

We can classify numerical data types as

- `int` : a subset of the integer numbers,
- `double` : double precision floating-point.

The family of `int` types breaks up in 2 families:

- 1 `bool`, `char`, `wchar_t` and unsigned integers `size_t`,
- 2 `short int` (16), `int` (32), and `long int` (64 bits).

The first bit is normally used for the sign.

A floating-point number consists of a sign, fraction (or significand) and an exponent:

- `float`: 23 bits in fraction, 8 bits in exponent
- `double`: 52 bits in fraction, 11 bits in exponent
- `long double`: 63 bits in fractions, 15 bits in exponent

We can classify numerical data types as

- `int` : a subset of the integer numbers,
- `double` : double precision floating-point.

The family of `int` types breaks up in 2 families:

- 1 `bool`, `char`, `wchar_t` and unsigned integers `size_t`,
- 2 `short int` (16), `int` (32), and `long int` (64 bits).

The first bit is normally used for the sign.

A floating-point number consists of a sign,
fraction (or significand) and an exponent:

- `float`: 23 bits in fraction, 8 bits in exponent
- `double`: 52 bits in fraction, 11 bits in exponent
- `long double`: 63 bits in fractions, 15 bits in exponent

We can classify numerical data types as

- `int` : a subset of the integer numbers,
- `double` : double precision floating-point.

The family of `int` types breaks up in 2 families:

- ① `bool`, `char`, `wchar_t` and unsigned integers `size_t`,
- ② `short int` (16), `int` (32), and `long int` (64 bits).

The first bit is normally used for the sign.

A floating-point number consists of a sign, fraction (or significand) and an exponent:

- `float`: 23 bits in fraction, 8 bits in exponent
- `double`: 52 bits in fraction, 11 bits in exponent
- `long double`: 63 bits in fractions, 15 bits in exponent

formatting doubles

Types and
Control
Statements

data types

if else statement

for loops

Simulations

random numbers

rolling a die

Functions and
Pointers

defining a function

call by value and call

by reference

pointers and

references

```

#include<iomanip>
// include io manipulators
double x = 1.2345e+9;
double y = 456.789;

cout << "x = " << x << ", y = " << y;

x = 1.2345e+09, y = 456.789

cout << fixed << setprecision(2)
     << "x = " << x << ", y = " << y;

x = 1234500000.00, y = 456.79

cout << scientific << setprecision(4)
     << "x = " << x << ", y = " << y;

x = 1.2345e+09, y = 4.5679e+02

```

formatting doubles

Types and
Control
Statements

data types

if else statement
for loops

Simulations

random numbers
rolling a dieFunctions and
Pointersdefining a function
call by value and call
by reference
pointers and
references

```

#include<iomanip>
// include io manipulators
double x = 1.2345e+9;
double y = 456.789;

cout << "x = " << x << ", y = " << y;
x = 1.2345e+09, y = 456.789

cout << fixed << setprecision(2)
    << "x = " << x << ", y = " << y;
x = 1234500000.00, y = 456.79

cout << scientific << setprecision(4)
    << "x = " << x << ", y = " << y;
x = 1.2345e+09, y = 4.5679e+02

```

formatting doubles

Types and
Control
Statements

data types

if else statement

for loops

Simulations

random numbers

rolling a die

Functions and
Pointers

defining a function

call by value and call

by reference

pointers and

references

```

#include<iomanip>
// include io manipulators
double x = 1.2345e+9;
double y = 456.789;

cout << "x = " << x << ", y = " << y;
x = 1.2345e+09, y = 456.789

cout << fixed << setprecision(2)
    << "x = " << x << ", y = " << y;
x = 1234500000.00, y = 456.79

cout << scientific << setprecision(4)
    << "x = " << x << ", y = " << y;
x = 1.2345e+09, y = 4.5679e+02

```

Introduction to Data Structures

Types and Control Statements

data types

if else statement

for loops

Simulations

random numbers

rolling a die

Functions and Pointers

defining a function

call by value and call

by reference

pointers and

references

1 Types and Control Statements

data types

if else statement

for loops

2 Simulations

random numbers

rolling a die

3 Functions and Pointers

defining a function

call by value and call by reference

pointers and references

coding a grading scale

Grading scale from our course description:

90 – 100% = A, 80 – 89% = B, 70 – 79% = C, 60 – 69% = D,
0 – 59% = E.

Input from user: score (a percentage).

Output to screen: letter grade with as justification a
reference to the corresponding percentage range.

```
$ grading_scale
```

```
Enter a percentage : 89.3
```

```
Score 90 equals the grade A, as 90 >= 90%.
```

```
$ grading_scale
```

```
Enter a percentage : 70
```

```
Score 70 equals the grade C, as 70% <= 70 <= 79%.
```

```
$ grading_scale
```

```
Enter a percentage : 23
```

```
Score 23 equals the grade E, as 23 <= 59%.
```

```
$
```

coding a grading scale

Grading scale from our course description:

90 – 100% = A, 80 – 89% = B, 70 – 79% = C, 60 – 69% = D,
0 – 59% = E.

Input from user: score (a percentage).

Output to screen: letter grade with as justification a
reference to the corresponding percentage range.

```
$ grading_scale
Enter a percentage : 89.3
Score 90 equals the grade A, as 90 >= 90%.
$ grading_scale
Enter a percentage : 70
Score 70 equals the grade C, as 70% <= 70 <= 79%.
$ grading_scale
Enter a percentage : 23
Score 23 equals the grade E, as 23 <= 59%.
$
```

coding a grading scale

Grading scale from our course description:

90 – 100% = A, 80 – 89% = B, 70 – 79% = C, 60 – 69% = D,
0 – 59% = E.

Input from user: score (a percentage).

Output to screen: letter grade with as justification a
reference to the corresponding percentage range.

```
$ grading_scale
```

```
Enter a percentage : 89.3
```

```
Score 90 equals the grade A, as 90 >= 90%.
```

```
$ grading_scale
```

```
Enter a percentage : 70
```

```
Score 70 equals the grade C, as 70% <= 70 <= 79%.
```

```
$ grading_scale
```

```
Enter a percentage : 23
```

```
Score 23 equals the grade E, as 23 <= 59%.
```

```
$
```

ceil() applied to a double

Types and Control Statements

data types

if else statement

for loops

Simulations

random numbers

rolling a die

Functions and Pointers

defining a function

call by value and call

by reference

pointers and

references

```
#include<iostream>
#include<math.h>
```

```
using namespace std;
```

```
int main()
{
```

```
    double input;
```

```
    cout << "Enter a percentage : ";
```

```
    cin >> input;
```

```
    int score = ceil(input);
```

ceil(x) returns the smallest integer not less than x

a nested if else

Applying the grading scale to score:

```
char grade;

if(score >= 90)
    grade = 'A';
else if(score >= 80)
    grade = 'B';
else if(score >= 70)
    grade = 'C';
else if(score >= 60)
    grade = 'D';
else
    grade = 'E';

cout << "Score " << score
     << " equals the grade "
     << grade << ", as ";
```

the switch statement

The justification refers to the grading scale:

```
switch(grade)
{
    case 'A': cout << score << " >= 90%"; break;
    case 'B': cout << "80% <= "
                << score << " <= 89%"; break;
    case 'C': cout << "70% <= "
                << score << " <= 79%"; break;
    case 'D': cout << "60% <= "
                << score << " <= 69%"; break;
    default : cout << score << " <= 59%"; break;
}
cout << "." << endl;

return 0;
}
```

Observe the `break` statement.

breaking out of a loop

Recall the do-while in the greatest common divisor computation of x and y :

```
int r;  
do  
{  
    r = x % y;  
    x = y; y = r;  
} while(r != 0);
```

We leave the loop as soon as remainder becomes zero:

```
do  
{  
    r = x % y;  
    if(r == 0) break;  
    x = y; y = r;  
} while(true);
```

breaking out of a loop

Recall the do-while in the greatest common divisor computation of x and y :

```
int r;  
do  
{  
    r = x % y;  
    x = y; y = r;  
} while(r != 0);
```

We leave the loop as soon as remainder becomes zero:

```
do  
{  
    r = x % y;  
    if(r == 0) break;  
    x = y; y = r;  
} while(true);
```

Introduction to Data Structures

Types and Control Statements

data types
if else statement
for loops

Simulations

random numbers
rolling a die

Functions and Pointers

defining a function
call by value and call by reference
pointers and references

1 Types and Control Statements

data types
if else statement
for loops

2 Simulations

random numbers
rolling a die

3 Functions and Pointers

defining a function
call by value and call by reference
pointers and references

for loops

The do-while and while loops are good when the number of iterations is not known in advance.

To compute $s = \sum_{k=1}^n k$:

```
int s = 0;
for(int k=1; k<=n; k++)
    s = s + k;
```

is equivalent to

```
int s = 0;
int k = 1;
while(k <= n)
    s = s + (k++); // s = s + k; k = k + 1;
```

Important: $s = s + (k++) \neq s = s + (++k)$.

for loops

Types and
Control
Statementsdata types
if else statement
for loops

Simulations

random numbers
rolling a dieFunctions and
Pointersdefining a function
call by value and call
by reference
pointers and
references

The do-while and while loops are good when the number of iterations is not known in advance.

To compute $s = \sum_{k=1}^n k$:

```
int s = 0;
for(int k=1; k<=n; k++)
    s = s + k;
```

is equivalent to

```
int s = 0;
int k = 1;
while(k <= n)
    s = s + (k++); // s = s + k; k = k + 1;
```

Important: $s = s + (k++) \neq s = s + (++k)$.

Introduction to Data Structures

Types and Control Statements

data types
if else statement
for loops

Simulations

random numbers
rolling a die

Functions and Pointers

defining a function
call by value and call by reference
pointers and references

1 Types and Control Statements

data types
if else statement
for loops

2 Simulations

random numbers
rolling a die

3 Functions and Pointers

defining a function
call by value and call by reference
pointers and references

random numbers

Via the C library, we generate *pseudorandom* numbers.

Three steps:

- 1 include `stdlib.h`, the C standard library
- 2 `srand(s)` set the seed of the generator to `s`
Common practice: `s` is the current time.
For debugging: `s` is fixed so get same sequence.
- 3 `rand()` returns an integer in $0 \dots \text{RAND_MAX}$
Use modulo operator `%` to limit range of numbers.

For a random double in $[0,1]$:

```
int r = rand()  
double d = double(r)/RAND_MAX;
```

The `double(r)` converts `r` to a double.

random numbers

Via the C library, we generate *pseudorandom* numbers.

Three steps:

- 1 include `cstdlib`, the C standard library
- 2 `srand(s)` set the seed of the generator to `s`
Common practice: `s` is the current time.
For debugging: `s` is fixed so get same sequence.
- 3 `rand()` returns an integer in `0..RAND_MAX`
 Use modulo operator `%` to limit range of numbers.

For a random double in `[0,1]`:

```
int r = rand()
double d = double(r)/RAND_MAX;
```

The `double(r)` converts `r` to a double.

random numbers

Via the C library, we generate *pseudorandom* numbers.

Three steps:

- 1 include `cstdlib`, the C standard library
- 2 `srand(s)` set the seed of the generator to `s`
Common practice: `s` is the current time.
For debugging: `s` is fixed so get same sequence.
- 3 `rand()` returns an integer in `0 .. RAND_MAX`
Use modulo operator `%` to limit range of numbers.

For a random double in `[0,1]`:

```
int r = rand()  
double d = double(r)/RAND_MAX;
```

The `double(r)` converts `r` to a double.

random numbers

Via the C library, we generate *pseudorandom* numbers.

Three steps:

- 1 include `stdlib.h`, the C standard library
- 2 `srand(s)` set the seed of the generator to `s`
Common practice: `s` is the current time.
For debugging: `s` is fixed so get same sequence.
- 3 `rand()` returns an integer in $0 \dots \text{RAND_MAX}$
Use modulo operator `%` to limit range of numbers.

For a random double in $[0,1]$:

```
int r = rand()  
double d = double(r)/RAND_MAX;
```

The `double(r)` converts `r` to a double.

Introduction to Data Structures

Types and Control Statements

data types
if else statement
for loops

Simulations

random numbers
rolling a die

Functions and Pointers

defining a function
call by value and call by reference
pointers and references

1 Types and Control Statements

data types
if else statement
for loops

2 Simulations

random numbers
rolling a die

3 Functions and Pointers

defining a function
call by value and call by reference
pointers and references

rolling a die

To check if the pseudorandom numbers give a fair die, we run a simulation.

```
$ die_freq
Simulating the rolling of a die...
give a positive integer : 100000
```

```
Frequency Table of 100000 times rolling a die :
```

```
#0 : 16700
#1 : 16521
#2 : 16660
#3 : 16666
#4 : 16782
#5 : 16671
```

```
$
```

rolling a die

To check if the pseudorandom numbers give a fair die, we run a simulation.

```
$ die_freq  
Simulating the rolling of a die...  
give a positive integer : 100000
```

Frequency Table of 100000 times rolling a die :

```
#0 : 16700  
#1 : 16521  
#2 : 16660  
#3 : 16666  
#4 : 16782  
#5 : 16671
```

```
$
```

Types and
Control
Statements

data types

if else statement

for loops

Simulations

random numbers

rolling a die

Functions and
Pointers

defining a function

call by value and call

by reference

pointers and

references

defining a table

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

int main()
{
    int n;

    std::cout << "Simulating the rolling of a die...
    std::cout << "   give a positive integer : ";
    std::cin >> n;

    std::srand(time(0)); // set seed to time

    int freq[6] = {0, 0, 0, 0, 0, 0};
```

updating the table

Types and
Control
Statements

data types
if else statement
for loops

Simulations

random numbers
rolling a die

Functions and
Pointers

defining a function
call by value and call
by reference
pointers and
references

```
for(int i=0; i<n; i++)
{
    int d = std::rand() % 6;
    // std::cout << " " << d;
    freq[d] = freq[d] + 1;
}
std::cout << "\nFrequency Table"
          << " of " << n
          << " times rolling a die :\n";

for(int i=0; i<6; i++)
    std::cout << " #" << i << " : "
              << freq[i] << "\n";

return 0;
}
```

Introduction to Data Structures

Types and Control Statements

data types
if else statement
for loops

Simulations

random numbers
rolling a die

Functions and Pointers

defining a function
call by value and call
by reference
pointers and
references

1 Types and Control Statements

data types
if else statement
for loops

2 Simulations

random numbers
rolling a die

3 Functions and Pointers

defining a function
call by value and call by reference
pointers and references

defining a function

A function has a prototype and a definition.

Prototype of a d-sided die:

```
int die ( int d );  
// returns a number in the range 0..d-1
```

A definition of the function die:

```
int die ( int d )  
{  
    int roll = rand() % d;  
    return roll;  
}
```

`d` is a parameter of the function `die`

`roll` is a local variable of the function

defining a function

A function has a prototype and a definition.

Prototype of a d-sided die:

```
int die ( int d );  
// returns a number in the range 0..d-1
```

A definition of the function die:

```
int die ( int d )  
{  
    int roll = rand() % d;  
    return roll;  
}
```

`d` is a parameter of the function `die`

`roll` is a local variable of the function

defining a function

A function has a prototype and a definition.

Prototype of a d-sided die:

```
int die ( int d );  
// returns a number in the range 0..d-1
```

A definition of the function die:

```
int die ( int d )  
{  
    int roll = rand() % d;  
    return roll;  
}
```

`d` is a parameter of the function `die`

`roll` is a local variable of the function

Introduction to Data Structures

Types and Control Statements

data types
if else statement
for loops

Simulations

random numbers
rolling a die

Functions and Pointers

defining a function
call by value and call by reference
pointers and references

1 Types and Control Statements

data types
if else statement
for loops

2 Simulations

random numbers
rolling a die

3 Functions and Pointers

defining a function
call by value and call by reference
pointers and references

Types and
Control
Statementsdata types
if else statement
for loops

Simulations

random numbers
rolling a dieFunctions and
Pointersdefining a function
call by value and call
by reference
pointers and
references

Using the function `die` in the simulation:

```
for(int i=0; i<n; i++)  
{  
    int d = die(6);  
    freq[d] = freq[d] + 1;  
}
```

The parameter of `die` is *call by value*:

- the value is copied to the parameter of `die`;
- the value of the parameter cannot change.

Using the function `die` in the simulation:

```
for(int i=0; i<n; i++)  
{  
    int d = die(6);  
    freq[d] = freq[d] + 1;  
}
```

The parameter of `die` is *call by value*:

- the value is copied to the parameter of `die`;
- the value of the parameter cannot change.

update with a function

Types and Control Statements

data types
if else statement
for loops

Simulations

random numbers
rolling a die

Functions and Pointers

defining a function
call by value and call by reference
pointers and references

To update the frequency table with a function:

```
void update ( int *t, int d );  
// updates the table t with the die value d
```

- void indicates there is no return
- by *t we pass an address to the function

```
void update ( int *t, int d )  
{  
    t[d] = t[d] + 1;  
}
```

update with a function

Types and Control Statements

data types
if else statement
for loops

Simulations

random numbers
rolling a die

Functions and Pointers

defining a function
call by value and call by reference
pointers and references

To update the frequency table with a function:

```
void update ( int *t, int d );  
// updates the table t with the die value d
```

- void indicates there is no return
- by *t we pass an address to the function

```
void update ( int *t, int d )  
{  
    t[d] = t[d] + 1;  
}
```

call by reference

Types and Control Statements

data types
if else statement
for loops

Simulations

random numbers
rolling a die

Functions and Pointers

defining a function
call by value and call by reference
pointers and references

In the function `main()`:

```
int freq[6] = {0, 0, 0, 0, 0, 0};  
  
for(int i=0; i<n; i++)  
{  
    int d = die(6);  
    update(freq,d);  
}
```

Because `freq` is an *array*, the name `freq` refers to the address in memory of the first item in the array.

call by reference

Types and Control Statements

data types
if else statement
for loops

Simulations

random numbers
rolling a die

Functions and Pointers

defining a function
call by value and call by reference
pointers and references

In the function `main()`:

```
int freq[6] = {0, 0, 0, 0, 0, 0};  
  
for(int i=0; i<n; i++)  
{  
    int d = die(6);  
    update(freq,d);  
}
```

Because `freq` is an *array*, the name `freq` refers to the address in memory of the first item in the array.

Introduction to Data Structures

Types and Control Statements

data types
if else statement
for loops

Simulations

random numbers
rolling a die

Functions and Pointers

defining a function
call by value and call
by reference
**pointers and
references**

1 Types and Control Statements

data types
if else statement
for loops

2 Simulations

random numbers
rolling a die

3 Functions and Pointers

defining a function
call by value and call by reference
pointers and references

pointers

Types and
Control
Statements

data types

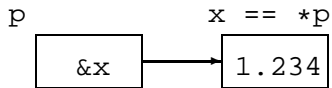
if else statement
for loops

Simulations

random numbers
rolling a dieFunctions and
Pointersdefining a function
call by value and call
by reference
pointers and
references

```
double x = 1.234;
double *p = &x;    // address of x
```

We have that p points to x :



```
cout << "x = " << x << ", via p : " << *p << endl;
double *q;
q = &x;
*q = *q + 1; // dereferencing q
cout << "x = " << x << ", via p : " << *p
      << ", via q : " << *q << endl;
```

$x = 1.234$, via p : 1.234

$x = 2.234$, via p : 2.234, via q : 2.234

pointers

Types and
Control
Statements

data types

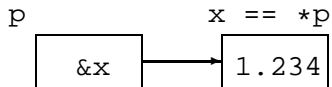
if else statement
for loops

Simulations

random numbers
rolling a dieFunctions and
Pointersdefining a function
call by value and call
by reference
pointers and
references

```
double x = 1.234;
double *p = &x;    // address of x
```

We have that p points to x :



```
cout << "x = " << x << ", via p : " << *p << endl;
double *q;
q = &x;
*q = *q + 1; // dereferencing q
cout << "x = " << x << ", via p : " << *p
      << ", via q : " << *q << endl;
```

```
x = 1.234, via p : 1.234
```

```
x = 2.234, via p : 2.234, via q : 2.234
```

pointer arithmetic

Arrays are consecutive items in memory:

```
double a[3] = {1.23, 2.34, 3.45};

for(int i=0; i<3; i++)
    cout << " " << a[i];
cout << endl;
```

is equivalent to

```
double *p;
p = a;
for(int i=0; i<3; i++)
    cout << " " << *(p++);
cout << endl;
```

pointer arithmetic

Arrays are consecutive items in memory:

```
double a[3] = {1.23, 2.34, 3.45};
```

```
for(int i=0; i<3; i++)  
    cout << " " << a[i];  
cout << endl;
```

is equivalent to

```
double *p;  
p = a;  
for(int i=0; i<3; i++)  
    cout << " " << *(p++);  
cout << endl;
```

references

Types and
Control
Statementsdata types
if else statement
for loops

Simulations

random numbers
rolling a dieFunctions and
Pointersdefining a function
call by value and call
by reference
**pointers and
references**

```
double y = 5.678;
double &z = y;
```

z is just another name for y

```
cout << "y = " << y << ", z = " << z << endl;
z = z + 1;
cout << "y = " << y << ", z = " << z << endl;
```

```
y = 5.678, z = 5.678
y = 6.678, z = 6.678
```

```
void swap ( double &a, double &b )
{
    double c = a;
    a = b; b = c;
}
```

references

Types and
Control
Statements

data types
if else statement
for loops

Simulations

random numbers
rolling a die

Functions and
Pointers

defining a function
call by value and call
by reference
pointers and
references

```
double y = 5.678;
double &z = y;
```

z is just another name for y

```
cout << "y = " << y << ", z = " << z << endl;
z = z + 1;
cout << "y = " << y << ", z = " << z << endl;
```

```
y = 5.678, z = 5.678
y = 6.678, z = 6.678
```

```
void swap ( double &a, double &b )
{
    double c = a;
    a = b; b = c;
}
```

Types and
Control
Statementsdata types
if else statement
for loops

Simulations

random numbers
rolling a dieFunctions and
Pointersdefining a function
call by value and call
by reference
pointers and
references

```
double y = 5.678;
double &z = y;
```

z is just another name for *y*

```
cout << "y = " << y << ", z = " << z << endl;
z = z + 1;
cout << "y = " << y << ", z = " << z << endl;
```

```
y = 5.678, z = 5.678
y = 6.678, z = 6.678
```

```
void swap ( double &a, double &b )
{
    double c = a;
    a = b; b = c;
}
```

Summary + Assignments

Types and Control Statements

data types
if else statement
for loops

Simulations

random numbers
rolling a die

Functions and Pointers

defining a function
call by value and call
by reference
pointers and
references

In this lecture we covered more of Chapter P.

Assignments:

- 1 Use the `pow` from `math.h` to add `a = pow(2, 31) = 231` and `b = 231`. Explain the outcome of this addition.
- 2 Declare `float x = 1.0;` and `double y = 1.0;` and do `x = x + 1.0e-10` and `y = y + 1.0e-10`. Explain the difference between the outcomes.
- 3 Print the ASCII table to screen, using the conversion `char(i)`, for all integers `i` ranging from 0 to 255.
- 4 Convert the for loops in the simulation of rolling a die into while loops.