Game Engine Programming

GMT Master Program
Utrecht University

Dr. Nicolas Pronost

Course code: INFOMGEP
Credits: 7.5 ECTS
Lecture #1

Part I: Introduction to C++
Introduction to C++

• Extension of C language
  – Bjarne Stroustrup (Bell Labs, 80’s) ‘C with classes’

• Open programming language
  – No owner, no central website, no official documentation except the ISO standard (1998)
  – Code compiled for a specific platform
  – Recent compilers have a high conformity with the standard but
    • A valid C++ code may not compiled if it uses advanced features not implemented in the compiler
    • An invalid C++ code may compiled with non rigorous compilers
Compilers

• Open source
  – GCC, Open Watcom ...

• Commercial products
  – Borland, Microsoft, SGI, Sun ...

• The standard specifies only the language (syntax) and its library
  – Compiler specific versions of network management, multi-task, UI, graphics ...
  – Compatibility / portability issues
What’s inside?

• Low-level manipulation of data
  – Pointer, memory usage ...

• Higher modeling functionalities
  – Reference, exception, class, template ...

• Programming techniques
  – OO, procedural and generic

• Suitable for large programs with high performance requirements
C++ vs. other languages

• Java
  – Compiled (vs. interpreted), separated declaration and definition, memory management

• C#
  – Multiple inheritance, separated declaration and definition, lower-level control

• Which language to use?
  – In industry 90% of the decision from financial issues
And in game engines?

• C++ is still an industry standard
  – Many games are programmed in C++ or use (prior) libraries written in C++

• Mostly, game companies use C++ for building their games
  – Object lifetime and memory management is often necessary
  – C++ allows for both high- and low-level coding
  – A lot of libraries and code is available

• Java is rarely used for games
  – But a lot of development is going on for Java3D, jMonkey engine and Java Scene Graphs

• C# is used in combination with XNA to produce Xbox games, Flash technology in casual games etc.
Lecture #1

Part II: C++ basics
Game Over!

```cpp
#include <iostream>

using namespace std;

int main(int argc, char* argv[]) {
    // This program prints Game Over!
    cout << "Game Over!" << endl;
    return 0;
}
```
The `#include <iostream>` directive loads the iostream library used for printing and reading data from the keyboard.

Comments are introduced by `//` (one line) or by `/*` and `*/` (multi-lines).

The `using namespace std;` simply means that we will directly use functions/objects from the package called “std”.

Game Over!
Game Over!

- The parameter `argc` gives the number of arguments (including the name of the program) and `argv` gives them in an array
  - `argc` and `argv` are optional
- The `cout` instruction prints data in the standard output (console)
- The returned value of the main program is
  - 0 if the program terminates normally
  - Non-zero for abnormal termination
Primitive types

• C++ has 5 primitive types
  – int, float, double, char and bool (true / false)
• C++ has no String class
  – Use array of ‘char’ or STL string (next lecture)
• In many libraries, ‘NULL’ is defined as macro for ‘0’ to increase readability
Using variables

• Normal variable

```c
int a;
```

• Reference to a variable (address of)

```c
int a = 2;
int & b = a; // reference
```

• Pointer to a variable (value pointed by)

```c
int a = 2;
int * c = & a; // pointer
```
Explicit casting

• To convert a value to a different type
• ⚠️ careful use as C++ does not generate compiler error

```
int x = 5; int y = 2; double z = 5.0;

double a = x / y; // a equals 2

double b = z / y; // b equals 2.5

double c = double(x)/double(y); // c equals 2.5
```
Operators

• Assignment to set a value to a variable
  \( = \) not the math equal and usually does not create compile-time error!

• Classical arithmetical operations
  \(-, +, -, /, \ast, \%\)

• Compound assignments
  \(+=, -= \ldots\)
  \(-\) value += increase means value = value + increase

• Increase, decrease
  \(-++ and --\)
  \(-a++ \iff a+=1 \iff a = a + 1\)
  • a++ returns the value before increment
  • ++a returns the value after increment
## Relational and logical operators

<table>
<thead>
<tr>
<th>operator</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![x]</td>
<td>Returns false if x is true and vice-versa</td>
</tr>
<tr>
<td>x &lt; y</td>
<td>Returns true if x is <strong>less than</strong> y</td>
</tr>
<tr>
<td>x &gt; y</td>
<td>Returns true if x is <strong>greater than</strong> y</td>
</tr>
<tr>
<td>x &lt;= y</td>
<td>Returns true if x is <strong>less than or equal to</strong> y</td>
</tr>
<tr>
<td>x &gt;= y</td>
<td>Returns true if x is <strong>greater than or equal to</strong> y</td>
</tr>
<tr>
<td>x == y</td>
<td>Returns true if x and y are <strong>equal</strong></td>
</tr>
<tr>
<td>x != y</td>
<td>Returns true if x and y are <strong>not equal</strong></td>
</tr>
<tr>
<td>x &amp;&amp; y</td>
<td>Returns true only if <strong>both</strong> x and y are <strong>true</strong></td>
</tr>
<tr>
<td>x ^^ y</td>
<td>Returns true if <strong>either</strong> x or y is <strong>true</strong> (not both)</td>
</tr>
<tr>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>
Control structures

• Conditional structure
  – the if-else statement

• Iteration structure
  – the while loop
  – the do-while loop
  – the for loop

• Jump structure
  – the break statement
  – the continue statement
  – the goto statement

• Selective structure
  – the switch statement
The if-else statement

• To execute a block only if a condition is fulfilled [otherwise execute another block]

```
if (condition) {block1;} [else {block2;}]
```

• Example

```
if (player_number > 0) {
    InitializeGameForPlayers(player_number);
    StartGame();
}
else WaitForMorePlayers();
```
The while loop

• To repeat a block while a condition is fulfilled

```java
while (condition) {block;}
```

• Example

```java
while (player_number <= 0) {
    player_number = GetMorePlayers();
}
```
The do-while loop

• Same as while loop except that the condition is evaluated after the execution of the block

```c
do {block;} while (condition);
```

• Example

```c
do {
    player_number = GetMorePlayers();
} while (player_number <= 0);
```
The for loop

• To repeat a block a certain number of times

```
for ([initialization]; condition; [statement]) {block;}
```

• Example

```
cout << "Respawn in 10 seconds: ";
for (int n = 10; n > 0; n--) {
    cout << n << " ";
    WaitOneSecond();
}
Respawn();
```
The brake statement

• To leave a loop even if the condition for its end is not fulfilled

• Example

```cpp
cout << "Respawn in 10 seconds: ";
for (int n = 10; n > 0; n--) {
    cout << n << " ";
    WaitOneSecond();
    if (NeedToAbord()) {
        cout << "countdown aborted!" << endl;
        break;
    }
}
Respawn();
```
The continue statement

- To skip the rest of the block causing the jump to the start of the next iteration
- Example

```cpp
// The code snippet

cout << "Respawn in 10 seconds: ";
for (int n = 10; n > 0; n--)
{
    cout << n << " ";
    if (NeedToSkipThatSecond()) continue;
    else WaitOneSecond();
}
Respawn();
```
The goto statement

• To make an absolute jump to another point in the program identified by a label
  – the label must be located in the current function

• Example

```cpp
cout << “Respawn in 10 seconds: ”;
int n = 10;
loop:
    cout << n << “ ”;
    n--;
if (n>0) goto loop;
Respawn();
```
The switch statement

• To check several possible **constant** values for an expression and execute blocks

• Example

```cpp
switch (option) {
    case 'a':
    case 'b':
    case 'c':
        cout << "Normal menu option" << endl;
        ExecuteOption(option);
        break;
    case '?':
        cout << "Help option" << endl;
        ShowHelp();
        break;
    default:
        cout << "Invalid option!" << endl;
}
```
Scope

• Variables are accessible in the block in which they are defined

```cpp
if (x == 12) {
   double z = 48.7;
}
cout << z << endl; // output?
```

```cpp
for (int i = 0; i < 10; i++) {
   cout << i << endl;
}
cout << i << endl; // output?
```
Standard Input / Output

- Using the C++ iostream library

```cpp
#include <iostream>
using namespace std;
```

- Print on the standard output (screen)

```cpp
cout << "Welcome " << PlayerName << endl;
```

- Read from the standard input (keyboard)

```cpp
int PlayerAge;
cout << "Please enter your age."
    cin >> PlayerAge;
```
Reading lines

- cin extraction stops reading as soon as it finds a blank space character
- Function getline to get the line in a string

```cpp
#include <iostream>
#include <string>
using namespace std;

int main () {
    string Quest;
    cout << "What is your quest?" << endl;
    getline(cin, Quest);
    cout << Quest << " is also my quest! Let's team up!";
    return 0;
}
```
Reading numerical values

• To perform extraction or insertion operations to convert strings to numerical values and vice-versa

```cpp
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

int main () {
    string inputString;
    int PlayerGold, PlayerSilver;
    cout << "How much gold and silver coins do you have?" << endl;
    getline(cin, inputString);
    stringstream(inputString) >> PlayerGold >> PlayerSilver;
    cout << "Can you give me " << PlayerGold / 2 << " gold coins?";
    return 0;
}
```
File Input / Output

• To read a file

```cpp
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main() {
    string line;
    ifstream myfile("GameSaved.txt");
    if (myfile.is_open()) { // accessing file?
        while ( !myfile.eof() ) { // parsing file
            getline(myfile,line); // reading line-by-line
            cout << line << endl;
        }
        myfile.close();
    } else cout << "Unable to open file";
    return 0;
}
```
File Input / Output

• To write a file

```cpp
#include <iostream>
#include <fstream>
using namespace std;

int main() {
    int PlayerLifes = 3;
    ofstream myfile("GameSaved.txt");
    if (myfile.is_open()) {
        myfile << “Game saved file” << endl;
        myfile << “Current lifes ” << PlayerLifes << endl;
        myfile.close();
    }
    else cout << "Unable to open file";
    return 0;
}
```
Functions

• A function is a group of statements that is executed when it is called from some point of the program

```plaintext
type name ([parameter1, parameter2, ...]) {block;}
```

– type is the type of the data returned by the function
– name is the identifier of the function
– parameters (data type followed by an identifier) act within the function as local variables
– block is the function's body
```cpp
#include <iostream>
using namespace std;

int subtraction (int a, int b) {
    int r;
    r = a - b;
    return r; // or return a - b;
}

int main() {
    int x = 5, y = 3, z;
    z = subtraction(7,2);
    cout << "The first result is " << z << '\n';
    cout << "The second result is " << subtraction(7,2) << '\n';
    cout << "The third result is " << subtraction(x,y) << '\n';
    z = 4 + subtraction (x,y);
    cout << "The fourth result is " << z << '\n';
    return 0;
}
```
void functions

- Functions with no parameters and/or no return type (procedures)

```c
void AVoidReturnFunction (int a) {
    int b = a + 1;
}

int AVoidParameterFunction (void) {
    int b = 1;
    return b;
}

void AVoidReturnAndParameterFunction () {
    int b = 1;
}
```
Modifying function

• Parameters are copies of the values but never the variables themselves
  – Modifications to them within the function will not have any effect on the values outside it
  – But if you want a modification, use a reference to the variable
#include <iostream>
using namespace std;

void PreviousAndNext (int x, int& prev, int& next) {
    prev = x-1;
    next = x+1;
}

int main () {
    int x=100; int y=15; int z=8;
    PreviousAndNext(x, y, z);
    cout << "Previous=" << y << ", Next=" << z;
    return 0;
}
Create data types

• Data structures
• Union of types
• Enumeration of types
• Definition of types
Data structures

• A data structure is a group of data elements (not necessarily of the same type) grouped together under one name

```c
struct structure_name {
    member_type1 member_name1;
    member_type2 member_name2;
    member_type3 member_name3; ...
} object_names;
```

• Examples

```c
struct PlayerState {
    bool alive;
    int amno;
};
PlayerState State1, State2;
```

```c
struct PlayerState {
    bool alive;
    int amno;
};
PlayerState State2, State3;
```
Data structures

- Manipulation of the members with the dot operator

```java
if (State1.alive && State2.alive && !State3.alive) {
    State1.amno += State3.amno / 2;
    State2.amno += State3.amno / 2;
    State3.amno = 0;
}
```

- Structures can be nested

```java
struct Player {
    float posx, posy;
    PlayerState state;
};
Player player1;
if (player1.posx == 0.0) player1.state.amno = 0;
```
Union of types

- Allow one same portion of memory to be accessed as different data types

```
union union_name {
    member_type1 member_name1;
    member_type2 member_name2;
    member_type3 member_name3; ...
} object_names;
```

- Example

```
union mix_t {
    double l;
    struct {
        int hi;
        int lo;
    } s;
    char c[4];
} mix;
```
Enumeration of types

- Create new data types to contain something different that is not limited to the values that fundamental data types may take

```c
enum enumeration_name {
    value1,
    value2, ...
} object_names;
```

- Example

```c
enum GameState {InMenu, Paused, Running};

GameState currentState = InMenu;
while (!playerReady) update();
currentState = Running;
```
Definition of type

• Definition of your own types based on other existing data types

```c
typedef existing_type new_type_name;
```

• Example

```c
typedef char C;
typedef unsigned int WORD;
typedef char field [50];

C mychar, anotherchar;
WORD myword;
field name;
```
End of lecture #1

Next lecture
Array, pointer, dynamic memory, string and OO basics