Crash Course into

C/C++

Prof. Dr. Renato Pajarola

C Language

- Low-level programming language
- General purpose imperative language
  - procedures and structures as only way to structure code and data types
- Syntax similar to Java
- Typed language with derived data types
  - but not strongly as explicit casting of types is possible

```
int
short            pointers
long             arrays
float     →      structures
double           unions
char             …
…
```

ANSI-C

- Standard language with set of libraries for I/O, string handling, character operations, math functions etc.
- Simple and compact language
  - independent of machine architecture
  - efficient compilation into native machine code
  - small required run-time library
  - source code portable
  - no GUI
- C++ extension provides object-oriented language features

C++

- C++ is a general purpose programming language with a bias towards systems programming that
  - is a better C
  - supports data abstraction
  - supports object-oriented programming
  - supports generic programming
- Java derived much of its syntax from C and C++
  - but Java has fewer low-level facilities
  - typically compiled into intermediate bytecode for Java VM

Program Structure

- Definitions and implementations are separated into *header* (.h/.hpp) and *source* (.c/.cc/.cpp) files
- Encapsulation and modularization is strongly encouraged by grouping code into header/source file pairs
  - header file contains all declarations of global variables, type definitions, data structures, procedures, objects and methods
  - actual implementation of procedures and methods is in the source file

Quicksort Routines

- Header file

```
/*
 *  quicksort.h
 */

void sort_array(float list[], int l, int r);

int partition_array(float list[], int l, int r);
```

- Source file

```
/*
 *  quicksort.cpp
 */

#include "quicksort.h"

void sort_array(float list[], int l, int r)
{
  int pivot_index;

  if (l < r) {
    pivot_index = partition_array(list, l, r);
    sort_array(list, l, pivot_index);
    sort_array(list, pivot_index+1, r);
  }
}
```

```
int partition_array(float list[], int l, int r)
{
  float tmp, pivot;
  int i;
  int j;

  pivot = list[l];

  i = l;
  j = r;
  while (1) {
    while (list[j] > pivot)
        j--;
    while (pivot > list[i])
        i++;

    if (i < j) {
        tmp = list[i];
        list[i] = list[j];
        list[j] = tmp;

      /* skip these two elements */
      j--;
      i++;
    } else
      return j;
  }
}
```

Main Quicksort

- Program must contain one `main()` function which is called at process startup

```
/*
 *  main.cpp
 */

#include <unistd.h>
#include <stdlib.h>
#include <iostream>

#include "quicksort.h"

#define ARRAY_SIZE 8

using namespace std;
```

```
int main(void)
{
  int i;
  float numbers[ARRAY_SIZE];

  /* initialize array of random float numbers */
  srandom(getpid());
  for (i = 0; i < SIZE; i++)
     numbers[i] = random();

  /* quicksort the array */
  sort_array(numbers, 0, SIZE-1);

  /* output result */
  for (i = 0; i < SIZE; i++)
     cout << "Number " << i << ": " << numbers[i] << endl;

  return 0;
}
```

External Code

- External functionality is imported via header files
  - #include <header_file>
- #include is a preprocessor directive preparing the source files for compilation
  - unistd.h declares the getpid() system function
  - stdlib.h declares the random number generator functions
  - iostream declares the standard C++ I/O streams
- #define is a preprocessor directive for symbolic constants

```
/*
 *  main.cpp
 */

#include <unistd.h>
#include <stdlib.h>
#include <iostream>

#include "quicksort.h"

#define ARRAY_SIZE 8
```

Namespaces

- C++ includes a number of standard classes and libraries
  - e.g I/O streams, strings or containers (STL)
  - standard C headers included as <`c_name`> instead of <`_name.h`>
- Namespaces used to limit scope of symbols to specific blocks of code
  - generally to avoid naming collisions
  - `namespace std –` space the C++ standard library resides in
- Declare namespace usage within scope of source code or for individual elements

```
using namespace std;     // imports all standard C++ library
                         // symbols into the current scope

using std::cout;         // import iostream cout only
```

Compiling and Linking

- At compile time only the header information is needed
  - only the function and variable definitions need to be verified
  - `cc -c quicksort.cpp` generates quicksort.o object file
    - include file directories can be specified with compiler flags
- At link time the actual object files and/or libraries are needed
  - object and libraries are merged and linked into one binary executable
  - `cc -o sort main.o quicksort.o` generates executable
    - standard libraries are linked automatically
    - extra libraries are indicated with compiler flags

Control Flow

- Sequence of statements terminated by `;`
  - definitions, assignments, procedure calls
  - blocks of statements within `{ }`
- Selection of code blocks
  - `if`, `else if`, `else` and `switch` statements
- Loops over statement blocks
  - `while`, `do` and `for` iterations
- Recursive calls of procedures

Data Types and Variables

- Declaration of variables by type generally at beginning of code block
  - `float numbers[ARRAY_SIZE];`
- Range of numeric types is machine dependent
  - `int` and `float` are typically 4 bytes on 32- or 64-bit systems
  - can check with `sizeof(<type>)`
  - use `#include <sys/types.h>` or `<inttypes.h>` for fixed size numerical types `uint8_t`, `int16_t`, `uint32_t`
- C++ strings are ASCII characters and modifiable
  - `string test = "Hello";`
  - `test += " World";`

Arithmetic

- Arithmetic expressions are based on implicit type conversion
  - starts with `int` → continues with truncated computations

```cpp
#include <iostream>

using namespace std;

int main(void)
{
  int fahr, celsius;

  /* Fahrenheit-Celsius */
  fahr = 57;
  celsius = (5 / 9 * (fahr - 32));
  cout << "Fahrenheit: " << fahr << "  Celsius: " << celsius;

  /* Celsius-Fahrenheit */
  celsius = 23;
  fahr = (9 / 5 * celsius) + 32;
  cout << "Celsius: " << celsius << "  Fahrenheit: " << fahr;
}
```

```
Convert Fahrenheit to Celsius

Fahrenheit: 57  Celsius: 0
Celsius: 23  Fahrenheit: 55
```

Functions

- Functions are identified via return value and parameters
  - `void sort_array(float list[], int l, int r);`
  - not part of any class → global functions
- Must be defined before being used
  - just procedure header without code body
- Arguments are call-by-value
  - functions receive a copy of the actual parameters
  - original cannot be modified inside function

References

- Call-by-reference can be enforced by ‘&’
  - if passed by reference, function can modify original variable
  - `void raiseSalary(Employee &e, int amount);`
  - normal behavior in Java on objects
- C++ references also work on basic types

```
void swap(int &a, int &b) {
  int tmp = a;
  a = b;
  b = tmp;
}
```

  - use references in C++ when function needs to modify parameters

Class Headers

- The class definition only contains the declaration of members and methods
  - implementation is separated in the source file
- Classes have public and private sections

```
class Employee {
public:
   Employee();
   Employee(string input);
   string getName() const;
private:
   string name;
};
```

  - protected – access by members and friends of derived classes

Class Implementation

- Methods are implemented in the source file
  - methods are prefixed by the class name and :: for correct class association

```
Employee::Employee {
  name = "Muster";
}
Employee::Employee(string input) {
  name = input;
}
string Employee::getName() const {
  return name;
}
```

Objects

- In C++ variables hold values not references
    - definition of variables causes memory to be allocated and a constructor to be called

```
Employee admin;
```

    - object is constructed using default constructor
        - causes only uninitialized reference in Java
- Assignment of variables causes copy of value
    - similar to `clone` in Java
    - no two variables for the same object
        - need to use pointers for that
- Object variable can only hold one particular type

Inheritance

- C++ syntax similar to Java
  - use of `: public` instead of `extend` to denote inheritance

```
class Manager : public Employee {
public:
  Manager(string nm, int salary, string dept);
  virtual void print() const;   // dynamically bound
private:
  string department;
};
```

- Unless specified with `virtual`, methods cannot be dynamically bound

Superclass Methods

- Invokation of superclass constructor done outside of constructor code body

```
Manager::Manager(string nm, int salary, string dept)
: Employee(nm, salary)     // initialization list
{
  department = dept;
}
```

- Reference to superclass via ::operator

```
void Manager::print() const {
  Employee::print();       // call superclass method
  cout << department << endl;
}
```

Polymorphism

- C++ variable of type `T` holds objects only of this type
  - variables hold value not reference to object
- Polymorphism requires use of pointer variable type `T*`
  - `T *p;` can point to `T` or any subclass of `T`

```
Employee *e = new Manager("Steve", 100000, "HW");
```

- Dynamic binding supported via pointers only

```
vector<Employee*> staff;
...
for (i = 0; i < staff.size(); i++)
  staff[i]->print();
```

Pointers

- Variables hold values

```
float x;
x = 0.5;
```

writes the float representation of 0.5 into the 4 bytes of variable x

- Pointers declared by '*' indicate memory addresses

```
float *px;
px = &x;
```

is a memory address of a float

address given by the '&' operator

- Pointers are dereferenced again by '*' to get value

```
float y;
y = *px + 1.0;
```

Pointers as Reference

- Similar to object variables in Java, pointers can be set to NULL and initialized with new

```
Employee *staff = NULL;  // always initialize !
Employee *chief = new Employee("Steve Jobs");
staff = chief;  // two variables pointing to the same
delete chief;   // leaves staff dangling
```

- To access object, point must be dereferenced

```
string boss = (*chief).getName();
```

  - or use the arrow operator ‘->’

```
string boss = chief->getName();
```

Arrays

- Defined as `type name[dimension]`
  - start index is at 0
- Access via `name[expression]`
  - where expression is an integer expression
    - implicit type cast converts any expression to integer index
  - array bounds are not implicitly checked
- Represents continuous block of memory
  - number of used bytes is `dimension * sizeof(type)`
  - variable `name` indicates start of array’s memory
    - `name` is in fact a memory address (pointer)

Pointer-Array Equivalence

- `NULL` indicates a void pointer
  - not pointing to any valid memory address (=0)
  - `int *pnum = NULL;` initialize pointers to `NULL` for safety
- Allocation via new and delete[]
  - malloc, calloc, free in standard C
- Array variables are pointers

```
char string[5];      pointer to fixed sized array
char *pc;            arbitrary pointer to a char
pc = string;
```

- Array indexing is dereferencing

```
pc = &string[2];     point to third element in array
*pc = string[3];     copy value to location at pc
```

Books

- The C++ Programming Language, *by Bjarne Stroustrup*, Addison Wesley, 2000
- The C Programming Language – ANSI C, *by Brian Kernighan and Dennis Ritchie*, Prentice Hall, 1988