Lecture 2

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Types – C++ Primer Chapter 2

Coding Conventions for Exercises

- Indent your code consistently
- Start a new line for each statement
- Naming conventions
  - Own object names: LikeThis
  - variables: likeThis
  - Class members: likeThis_
  - Macros: LIKE_THIS
  - Integer types: i,j,k
  - Floating point: x,y,z
- Comment your code, e.g. in declaration when variable is initialized

Useful Tipps

- **plan-edit&comment-compile-debug-test**
- Compile cleanly at high warning levels
  - Wall
  - Know compiler options: g++, cl
- Use an automated build system
  - Make, cmake
- Use a version control system (bigger codes)
  - git
- Invest in code reviews
  - Debuggers: gdb, ddd
  - Valgrind
  - Profilers: gprof

Arithmetic Types

- **Integral Types**
  - bool, char, wchar_t, char16_t, char32_t, short, int, long, long long
  - wchar_t type is intended for characters from an extended character set; char type fits in 8 bits, e.g. Latin-1 or ASCII.
  - signed and unsigned
- **Floating-Point Types**
  - float: 6 significant digits
  - double, long double: 10 significant digits
- **Bool type**
  - may hold only two values: true or false

Literal Constants

- **Integer Literals**
  - What means 20, 024, 0x14?
  - unsigned: 128u, long 1L (assign a negative number to unsigned!)
- **Floating-Point Literals**
  - 0. , 0e0 , .001f, 1E-3F (type of constants 1 and 1.0?)
- **Boolean and Literals**
  - Bool: true, false
  - Character: ´a´
  - string: “a“

Definitions

literal constant: A value such as a number, a character, or a string of characters.

- value cannot be changed
- Literal characters are enclosed in single quotes, literal strings in double quotes.

Escape sequences

escape sequence: Alternative mechanism for representing characters.

- Usually used to represent nonprintable characters such as newline or tab.
- An escape sequence is a backslash followed by a character, a three-digit octal number, or a hexadecimal number.
- Escape sequences can be used as a literal character (enclosed in single quotes) or as part of a literal string (enclosed in double quotes).
- Examples: \n, \t, \\, \b

Variables

Object: A region of memory that has a type. A variable is an object that has a name.

Declaration: Asserts the existence of a variable, function, or type defined elsewhere in the program.

- Some declarations are also definitions; only definitions allocate storage for variables.

Definition: Allocates storage for a variable of a specified type and optionally initializes the variable.

Names may not be used until they are defined or declared!

Variables

type specifier: Part of a definition or declaration that names the type of the variables that follow.

Identifier: A name.

- A nonempty sequence of letters, digits, and underscores that must not begin with a digit.
- Identifiers are case-sensitive: Upper- and lowercase letters are distinct.
- Identifiers may not use C++ keywords.

Type-checking

statically typed: Term used to refer to languages such as C++ that do compile-time type checking.

C++ verifies at compile-time that the types used in expressions are capable of performing the operations required by the expression.

type-checking: Process by which the compiler verifies that the way objects of a given type are used is consistent with the definition of that type.

Variable Initialization

variable initialization: Rules for initializing variables and array elements when no explicit initializer is given.

- For class types, objects are initialized by running the class's default constructor. If there is no default constructor, then there is a compile-time error: The object must be given an explicit initializer.
- For built-in types, initialization depends on scope. Objects defined at global scope are initialized to 0; those defined at local scope are uninitialized and have undefined values.

Scope (C++14std 3.3)


```
std::string s1 = "hello";

int main()
{
      std::string s2 = "world";

      std::cout << s1 << " " << s2 << std::endl;

      int s1 = 42;

      std::cout << s1 << " " <<  s2 << std::endl;
      return 0;
}
```

Scope

- **Scope**: A portion of a program in which names have meaning. C++ has several levels of scope:
  - global— names defined outside any other scope.
  - class— names defined by a class.
  - namespace— names defined within a namespace.
  - local— names defined within a function.
  - block— names defined within a block of statements, that is, within a pair of curly braces.
  - statement— names defined within the condition of a statement, such as an if, for, or while.
  - Scopes nest. For example, names declared at global scope are accessible in function and statement scope.

References

Reference: An alias for another object.

Defined as follows: type &id = object;

- Defines id to be another name for object. Any operation on id is translated as an operation on object.
- There is no way to rebind a reference to a different object
- Nonconst reference may be attached only to an object of the same type as the reference itself

const reference: A reference that may be bound to a const object, a nonconst object, or the result of an expression.

- A const reference may not change the object to which it refers

Pointers I

Pointer: An object that holds the address of an object.

Example: int * p;

Values used to initialize or assign to a pointer:

- A constant expression with value 0 or better nullptr
- An address of an object of an appropriate type
- The address one past the end of another object
- Another valid pointer of the same type

Pointers II

- Pointers are iterators for arrays
- **void*:** A pointer type that can point to any nonconst type.
  - Only limited operations are permitted on void* pointers:
  - They can be passed or returned from functions and they can be compared with other pointers.
  - They may not be dereferenced.

Operators

*** operator:** Dereferencing a pointer yields the object to which the pointer points.

Assigning to the result of a dereference assigns a new value to the underlying object.

Example: int * p; *p = 2;

& operator: The address-of operator.

Yields the address in memory to which it is applied.

Pointers vs. references


```
int i = 1, j = 2;
int *pi = &i, *pj = &j;
pi = pj;

int &ri = i, &rj = j;
ri = rj;
```

- Comparing pointers and references
  - References always refer to an object
  - Assigning to a reference changes the underlying object

Pointers and the const qualifier


```
const double* cptr;
*cptr = 42;

int ierr = 0;
int *const curErr = &ierr;
curErr = curErr;

const double pi = 3.14;
const double* const pi_ptr = &pi;
```

- Pointers to const objects
  - Pointers that think they are const
- const pointers
- const pointers to const objects

Typedef

Typedef: Introduces a synonym for some other type.

Form: *typedef type synonym;* defines synonym as another name for the type named type.

Alternative (C++11): using synonym = type;

Purposes:

- Hide implementation of a given type and emphasize instead the purpose for which the type is used
- Streamline complex type definitions, making them easier to understand
- Allow a single type to be used for more than one purpose while making the purpose clear each time the type is used

decltype (C++14std 7.1.6.3)

- Type specifier that deduces the type of a variable or an expression
- Example:
  - const int ci = 0;
  - decltype(ci) x = 0; // x has type const int

auto (C++14std 7.1.6.4)

- Type specifier that deduces the type of a variable from its initializer
- Example:
  - auto i = 10; // i is an int

Class Types

- **Class**: C++ mechanism for defining data types.
  - Classes are defined using either the class or struct keyword.
  - Classes may have data and function members.
  - Access labels for members are public, protected, or private.
  - By default, members in a class defined using the class keyword are private; members in a class defined using the struct keyword are public.
  - Remember to put semicolon at end of class definition!

Arrays

Array: Data structure that holds a collection of unnamed objects that can be accessed by an index.

Example: int arr[5];

dynamically allocated: An object that is allocated on the program's free store.

Objects allocated on the free store exist until they are explicitly deleted.

free store (heap): Memory pool available to a program to hold dynamically allocated objects.

Security problem: buffer overflow

Operators

[] operator: The subscript operator takes two operands: a pointer to an element of an array and an index.

- Its result is the element that is offset from the pointer by the index.
- Indices count from 0.
- The subscript operator returns an lvalue.

++ operator: When used with a pointer, the increment operator "adds one" by moving the pointer to refer to the next element in an array.

Example: ++i;

New and delete

new expression: Allocates dynamic memory.

- We allocate an array of n elements as follows: new type[n];
- new returns a pointer to the first element in the array.

delete expression: A delete expression frees memory that was allocated by new:

- delete [] p;
- p must be a pointer to the first element in a dynamically allocated array.
- The bracket pair is essential: It indicates to the compiler that the pointer points at an array, not at a single object.