C++

PENG

Created: 20260817, Version: 20260831

Contents

1 Variables and Basic Types

1.1 Primitive Built-in Types

1.1.1 Arithmetic Types

Definition 1.1 (Primitive builtin types [1])**.**

- *arithmetic types:*
  - *integral types*
    - *characters:* `char`, `std::string`
    - *integers:* `int`*:* `signed` *or* `unsigned` *(*≥ 0*)*
    - *boolean values:* `bool`
  - *floating-point types:* `double`
- *void*

Remark 1.2. *Use* `double` *for floating-point computations.*

1.1.2 Type Conversions

Example 1.3 (Type conversions)**.**

```
bool b = 20; // 0 for false and nonzero values for true
int i = b;   // true for 1 and false for 0
```

Example 1.4 (Signed and unsigned)**.**

```
unsigned u = -1; // if 32-bit ints, 4294967295
int a = -1;
unsigned b = 1;
a * b // int value is converted to unsigned before addition
```

1.1.3 Literals

Definition 1.5 (Literals)**.**
Integer literals:

- *decimal*
- *octal:* `0-`
- *hexadecimal:* `0x-`
- *suffix:*
  - `u`*: unsigned*
  - `L`*: long*

Floating-point literals (`double` *by default):*

- *decimal point*
- `E: 1E-2`
- *suffix:* `f/F`*: float*

character and string literals: `"Hello"` = `"Hello\0"`

Definition 1.6 (Escape sequences)**.**

- *newline:* `\n`
- *horizontal tab:* `\t`
- *null:* `\0`

1.2 Variables

1.2.1 Variable Definitions

Remark 1.7.

- *Variables defined outside any function body are initialized to 0.*
- *Initialize every object of built-in type.*
- `std::cin >> int i;` *error, should be* `int i; std::cin >> i;`.

1.2.2 Variable Declarations

- File A define the variable `int ix;`. Other files: `extern int ix;`.
- All variables must be declared before being used.

1.2.3 Identifiers

Naming conventions:

- Variable name: lowercase
- Class name: uppercase

1.2.4 Scope of a Name

- Global scope: `::global_var`
- Block scope

Remark 1.8. *It is almost always a bad idea to define a local variable with the same name as a global variable that the function uses.*

1.3 Compound Types

1.3.1 References

Definition 1.9 (Reference)**.** *A reference defines an alternative name for an object.*

```
int ival = 0;
int &refVal = ival;
```

- A reference is an alias, not an object.
- The type of a reference and the object must match exactly.
- A reference can only refer to the object to which it was initially bound.

1.3.2 Pointers

Definition 1.10 (Pointer)**.** *A pointer holds the address of another object.*

```
int ival = 0;
int *pi = &ival;
*pi = 10;
int* p1, p2; // legal, but might be misleading. p2 is int.
```

- A pointer is an object. The type of the pointer and the object to which it points must match.
- Dereferencing a pointer yields the object to which the pointer points.
- States of address value stored in a pointer:
  - point to an object
  - null pointer, not bound to any object: `nullptr`

Remark 1.11 (Initialize all pointers)**.** *Define a pointer only after the object to which it should point, or initialize the pointer to* `nullptr`*.*

Example 1.12 (Pointers to pointers)**.**

```
int ival = 1;
int *pi = &ival;
int **ppi = &pi;
```

Example 1.13 (References to pointers)**.**

```
int i = 1;
int *p;
int *&r = p;
```

1.4 Qualifier

Definition 1.14. `const`*:*

- *Make a variable unchangeable.*
- *Must be initialized.*

Example 1.15. *Share a* `const` *object among multiple files:*

```
extern const i = 10; // file A
extern const i; // other files
```

1.4.1 Reference to const

Definition 1.16 (reference to const)**.** *Reference to const only restricts what we can do through that reference.*

Reference to const can refer to a nonconst object.

```
const int ci = 10;
const int &r1 = ci;
int i = 1;
const int &r2 = i;
const int &r3 = 2;
double d = 3.14;
const int &r4 = d;
```

1.4.2 Pointer to const

Definition 1.17 (Pointer to const: low-level)**.** *Pointer to const only affects what we can do with the pointer.*

Pointer to const can point to a nonconst object.

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

1.4.3 Top-Level const

Definition 1.18 (const pointer: top-level)**.** *const pointer must be initialized.*

```
int i = 10;
int *const pi = &i;
```

Remark 1.19. *Low-level* `const` *is never ignored when copying an object. Both objects must have the same low-level qualification or binding const to nonconst.*

```
const int i = 1;
const int *p1 = &i;
const int *const p2 = p1;
int *p3 = 0;
p1 = p3;
```

1.4.4 Constant Expressions

Definition 1.20 (constexpr)**.**

```
const int sz = get_size(); // not const expression
constexpr int sz = size(); // ok only if size is a constexpr function
const int *p = 0;     // pointer to const int
constexpr int *q = 0; // const pointer to int: top-level const
```

Remark 1.21.

- *Variables defined inside a function are not stored at a fixed address.*
- *The address of an object defined outside of any function is a constant expression.*

1.5 Dealing with Types

1.5.1 Type Aliases

Definition 1.22 (type alias)**.** `typedef double wages;`

```
typedef wages base, *p; // p for double*
base d = 3.14;
p p1 = &d, p2 = &d;
```

Definition 1.23 (pointer, const, type alias)**.**

```
typedef char *pstring;  // char*
const pstring cstr = 0; // char *const
const pstring *ps;      // char **const
```

1.5.2 The Auto Type Specifier

Definition 1.24 (auto)**.** *A variable that uses* `auto` *must have an initializer.*
The initializers for all the variables in the declaration must have consistent types.

```
auto i = 0, *p = &i;
auto sz = 0, pi = 3.14;
```

Example 1.25 (compound types, const, and auto)**.**

```
int i = 1;
const int ci = i;
auto a = ci;       // int, top level const is dropped
const auto b = ci; // const int
auto c = &i;       // int*
auto d = &ci;      // const int* (& of a const object is low-level const)
```

2 Defining Out Own Data Structures

```
struct Node
{
    int val;
    Node *next;
    Node(int x) : val(x), next(nullptr) {}
};

int main()
{
    Node *n1 = new Node(1);
    Node *n2 = new Node(2);
    n1->next = n2;
    while (n1 != nullptr)
    {
        std::cout << n1->val << std::endl;
        n1 = n1->next;
    }
    return 0;
}
```

References

[1] S.B. Lippman, J. Lajoie, and B.E. Moo. *C++ Primer*. Pearson Education, 2012. ISBN: 9780133053036.