ECE2800J Mid RC

Tianle Xue

Global College - Shanghai Jiaotong University

10/31/2025

Table of Contents

General Tips

1. Try to get all the easy points (Review the slides carefully)
2. Try not to leave anything blank, so that we can find some points to give you
3. Do come to paper checking, for ECE280, usually you can find some points to be added.
4. Review homework especially project 2. At least know what is ”List” and ”Tree” in project...

Table of Contents

SHANGHAI JIAO TONG UNIVERSITY

Const Data Variable

Basic syntax:

```
const data_type variable_name = value;
```

Properties: The value of a const variable cannot be changed after initialization.

```
const int MAX_SIZE = 100;
MAX_SIZE = 200; // Error
```

Const with References

- Reference must be initialized with an object
- Non-const reference must be initialized with an lvalue
- Const reference can be initialized with an rvalue

```
const int N = 10;
const int NN; // illegal. Should have initial value.
const int& ref1 = N; // legal
const int& ref2 = 6; // legal
int &ref = 5; //illegal.
const int cInt = 0;
int &ref = cInt; // illegal
const int& cref = cInt; //legal
```

Const with Reference

When calling big structures, we can use const to optimize time:

```
struct BigStruct{
  //many things...
};
void f1(BigStruct& T);
void f2(const BigStruct& T); // This is quicker when passing T.
    But T should not be changed!
```

Const with Pointers

A pointer stores the address of an object. Two candicates for the const qualifier: the pointer itself (the address), and the object it points to.

```
const int* ptr2Intc; // the integer cannot be modified
int const* ptr2cInt; // the integer cannot be modified
int* const cPtr2Int; // the address cannot be modified
```

Exercise

```
// Think about the following:
// Which are illegal?
int a = 1, b = 2;
const int c = 3;
const int d = 4;
const int* p1 = &c;
const int* p2 = &a;
int* const p3 = &b;
int* const p4 = &d;
const int* const p5 = &d;
p1 = &a;
*p1 = 6;
p3 = &a;
*p3 = 7;
```

Answer

```
// Think about the following:
// Which are illegal?
int a = 1, b = 2;
const int c = 3;
const int d = 4;
const int* p1 = &c; // legal
const int* p2 = &a; // legal
int* const p3 = &b; // legal
int* const p4 = &d; // illegal
const int* const p5 = &d; // legal
p1 = &a; //legal
*p1 = 6; //illegal, p1 is pointer to const int, cannot modify
    through pointer.
p3 = &a; //illegal, p3 is const pointer, cannot change
*p3 = 7; // legal
```

Const with Function Parameters

For function parameters, it is recommended to pass by const reference if the parameter is not modified inside the function. The type compatibility is as follows:

- const type & to type & is not allowed.
- type & to const type & is allowed.
- const type * to type * is not allowed.
- type * to const type * is allowed.

From non-const to const is allowed.

Exercise

```
void const_reference_test(const int &r) {}
void reference_test(int &r) {}
void pointer_test(int *p) {}
int main() {
  int a = 0;
  const int b = 0;
  int *p = &a;
  const int *cp = &a;
  // Which of the following function calls are valid?
  const_reference_test(a); // valid
  const_reference_test(b);// valid
  const_reference_test(*p);// valid
  const_reference_test(*cp);// valid
  reference_test(a);//valid
  reference_test(b);//illegal, b is const int, cannot be tranform
      to int&.
```

Exercise

```

    reference_test(*p); //legal, *p is lvalue
    reference_test(*cp);// illegal,*cp is const int lvalue, cannot
        be int&
    pointer_test(p);//legal
    pointer_test(cp);// illegal
}
```

Typedef

typedef is used to create an alias for a data type. It is often used to simplify the declaration of complex data types, just like using reference variables.

```
typedef int* int_ptr;
// int_ptr is an alias for int*
typedef const int_ptr const_int_ptr;
// the defined alias can be used in other typedef
typedef const int* const_int_ptr;
// All in one line
```

Table of Contents

Procedure Abstraction

Concepts

- Abstraction is a process of emphasizing the separation of ”what” and ”how”. It helps programmers to use a function without knowing how it is implemented.
- It only provides the details that are relevant to the user, and hide the unnecessary details.

```
// In the header file, the function is declared
int add(int a, int b);
// In the cpp file, the function is defined/implemented
int add(int a, int b) {
  return a + b;
}
```

Properties

- **Local:** The implementation of an abstraction is independent of any other abstraction implementation.

```
// Here the user doesn't need to know how multiply is
    implemented
int square(int a) {
    return multiply(a, a);
}
```

- **Substitutable:** The implementation of an abstraction can be replaced with another implementation if the interface is the same and the implementation is correct.

```
// Here the implementation can be replaced by another correct
    implementation
int multiply(int a, int b) {
    return a * b; // can be replaced by return b*a
}
```

Specification Comments

Besides the type signature, a function should also have specification comments. There're usually three types of comments:

- **REQUIRES**: Preconditions that must hold, if any.
- **MODIFIES**: Variables that are modified, if any.
- **EFFECTS**: What the procedure does given legal inputs.

```
int positiveAdd(int a, int b);
// REQUIRES: a > 0, b > 0
// MODIFIES: None
// EFFECTS: Returns the sum of a and b
```

NOTE that functions with no REQUIRES clause is complete, while functions with them are partial.

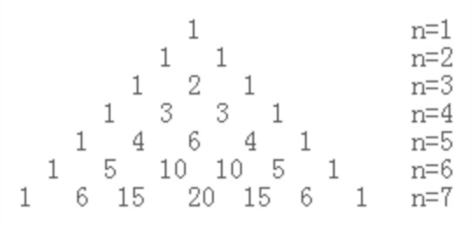
Table of Contents

Recursion Example

```
int factorial(int n){
    if (n == 0){
        return 1; // BASE CASE, recursion must have an end
    }
    return n * factorial(n- 1); // recursive step
}
```

Exercise 1

Figure out the value of i th row and j th column (i and j start from 0)



```
            1                       n=1
          1   1                     n=2
        1   2   1                   n=3
      1   3   3   1                 n=4
    1   4   6   4   1               n=5
  1   5   10  10  5   1             n=6
1   6   15  20  15  6   1           n=7
```

Exercise 2

Given a set of characters and a positive integer k, print all possible strings of length k that can be formed from the given set.
(https://www.geeksforgeeks.org/recursion-practice-problems-solutions/)
Example: set[ ] = 'a', 'b' , k = 3 Output: aaa aab aba abb baa bab bba bbb

Exercise 3

Leetcode Solution is good!
do it by yourself to gain bettr understanding of recursion.
https://leetcode.com/problems/merge-two-sorted-lists/description/
https://leetcode.com/problems/swap-nodes-in-pairs/description/
https://leetcode.com/problems/reverse-linked-list/description/

Tips for Exam

In the exercise, you are required to write code by hands, which means you don't have compiler or "AI" to help you debug and no test case are provided.

There are some general tips.

1. Don't forget to add { } ; ,
2. Consider all the boundary condition of the function, when to end the recursion?
3. Try to consider all the condition.
4. Make sure you write on the given underline in the exam paper.
5. Write on the draft paper first if you have time, because any code outside the given line is invalid. If you write on the exam paper without thinking carefully....

```
if (list_isEmpty(list)) // boundary condition
    .....
else if (fn(list_first(list)))
    .....
else
    .....
```

Function Pointers

Motivation

- Save time and make your code elegant
- Min and max assigne to compareHelp
- Treat functions as variables.

Grammar

Grammar

```
int (*foo)(int,int);
foo = max;
foo(5,3);
```

Function Pointer

Use function as function parameters:

```

void bubbleSort(int *arr,int n,bool (*cmp)(int,int)){
  for (int i=0;i<n;i++){
    for(int j=0;j<n-i-1;j++){
      if(cmp(arr[j],arr[j+1])){
        std::swap(arr[j],arr[j+1]);
      }
    }
  }
}
```

If you pass in bool cmp(inta, intb){returna<b;}. The array will be sorted in ascending order.
If you pass in bool cmp(inta, intb){returna>b;}. The array will be sorted in descending order.

Question

Can we do:

```
int max_nonconst(int a, int b) { return a > b ? a : b; }

int max_const(const int a, const int b) { return a < b ? a : b; }

int main(void) {
  int (*fp1)(const int, const int) = max_nonconst; \\ OK
  const int (*fp2) (int, int) = max_nonconst;\\ OK
  int (*fp3)(int, int) = max_const;\\ Not-OK
}
```

Table of Contents

Function Call Mechanism

- Stack: a set of objects which is modified as last in first out. (Think about a pile of plates, we can only put a new plate on the top or remove the plate at the top.)
- Call Stack: a stack that stores the order of function calls.

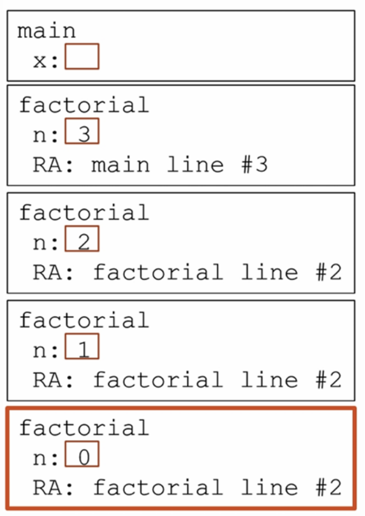
Steps

- When a function f() is called, its activation record is added to the top of the stack;
- When the function f() returns, its activation record is removed from the top of the stack;
- f() may have called other functions.
  These functions create corresponding activation records.
  These functions must return (and destroy their corresponding activation records) before f() can return.

Example

```
int f(int n) {
  // REQUIRES: n >= 0
  // EFFECTS: return n!
  if(n == 0 || n == 1) {
    // stop case
    return 1; // 1! = 0! = 1, trivially.
  } else {
      return n * f(n - 1);
      // We can refer to the math relationship:
      // n! = n * (n - 1)!
  }
}
```

Example


main
x:
factorial
n: 3
RA: main line #3
factorial
n: 2
RA: factorial line #2
factorial
n: 1
RA: factorial line #2
factorial
n: 0
RA: factorial line #2

Function Call Mechanism

- Give a recursive function which get some variable as arguments. Write an call stack of that function.
- Give a recursive function which get some variable as arguments and the function will output its argument variables. Write the output of the function.(Be careful of the output format, or you may lose some credits)

Exercise

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

void printDown(int n) {
    if (n > 0) {
        cout << n << "␣";
        printDown(n - 1);
    }
}
int main() {
    cout << "Output␣of␣printDown(3):␣";
    printDown(3);
    cout << endl;
    return 0;
}
```

Answer

Output of printDown(3): 3 2 1

Table of Contents

Enum

Enum can act as index.

```
enum Direction { up, down, left, right }; // 0 1 2 3
Direction dir = up; // 0
const int dx[4] = {0, 0, -1, 1};
const int dy[4] = {1, -1, 0, 0};
void move(Direction dir, int& x, int& y) {
  x += dx[dir];
  y += dy[dir];
}
```

References

[1] Haoyang, Wu. VE280-25sp Midterm RC

[2] Yiheng, Li. VE280-25sp Midterm RC

[3] Tongfei, Zou. VE280-25su Midterm RC

[4] Weikang, Qian. Lecture Slides.

[5] ZhongQiang, Ren. Lecture Slides.