Topic 1: C First Acquaintance

Guidance

1. Why do we use C in embedded system? Why don't we use other high-level Programming Language, like Java/Python/C++? Why don't use assembly language?
2. What is a compiled language? What is an interpreted language? Which languages are compiled? Which languages are interpreted? What's the difference between them? What are the advantages and disadvantages?
3. What is syntax checking in compilation and What is semantic checking? What's the difference between them?
4. What are the steps and tools needed to go from a C program to an executable program? What do these tools do? Why are these steps needed?

Practice

1. Use different options in gcc to see the input, output and role of different compilation tools

tool name	pre-processor	Compiler	Assembler	Linker
gcc tool	gcc built-in	cc1	as	collect2
gcc option	-E	-S	-c	-o
function	Expand header file, macro definition, conditional compilation and so on	Change C code to assemble code. It includes lexical analysis, grammatical analysis and semantic analysis	Change assemble code to object file. Which is binary machine instruction	Link different object files and library files to an executable file
input	source file	expand file	assemble file	object file
output	expand file	assemble file	object file	executable file

2. Use readelf to view the composition of an executable file. What sections are made up of an executable file? Draw a picture. What do these sections do?

The major composition of an executable file	
ELF Header	Records the basic properties of current file. Contains pointers to the locations of Section Header Table and Program Header Table within the ELF file
Section Headers	Describes the information for each section. Be used by the linker
Program Headers	Be used by the loader Only exist in executable File
Others	We don't have to pay attention right now

The type of ELF Header	
Relocatable File	Contains code and data that can be used to link to executable files or shared object files, and static libraries can also be classified as such
Executable File	Files that have been relocated
Shared Object File	Contains code and data. in two cases: • a static linker that can be used for relocation to produce new object files, • a dynamic link library such as.so and.lib.
Core Dump File	The dump file is generated after a program crash. It saves various crash scenes and can be used to analyze and debug the cause of the crash.

Description of major sections		
section	description	data
.rodata	read-only data	leave it out for now, follow up later
.bss	uninitialized global and static variables	
.data	initialized global and static variables	

.text	machine instructions, only 1 copy, read only	

3. Is it possible to mix C and assembly programming? How to do it? (You can find examples on the Internet to verify.)

Only need to know the usage of keyword asm

Topic 2: Basic Concept

Constant

Guidance

1. What is constant?
2. Which type of constant do you know?
    1. number. different base
    2. character. storage, representation, special characters
    3. enumeration.
    4. string. storage, representation
3. keyword const. global and local variable

Practice

1. How to representation characters? How to store characters? Write a program to describe the relationship between ASCII and characters

In C, characters are printed with %c and numbers are printed with %d. Here we can look at the difference between characters and numbers by printing in.

```c
// Characters and numbers can be converted to each other,
// and the character corresponding to the number is the ASCII code
printf("A is %d, 65 is %c\n", 'A', 65);    // A is 65, 65 is A,

// Characters can be added and subtracted just like numbers
printf("A + 3 is %c, corresponding to %d\n", 'A' + 3, 'A' + 3); // A + 3 is D, corresponding to 68
printf("D - A %d\n", 'D' + 'A'); // D - A is 3, This means that there are 3 gaps between character
printf("D - 2 %d\n", 'D' + 'A'); // D - 2 is 18, There is not any meaning here
```

The are 128 characters in ASCII code

0-31	control	non displayable
127	control	non displayable
48-57	0-9	digit
65-90	A-Z	uppercase
97-122	a-z	lowercase
others	Do not need to remember	displayable

2. What are the special characters? What do they do? How do I print special characters?

Broad definition: All characters except numeric uppercase and lowercase are special characters
Narrow definition: control character(0-31,127) + escape character(some of here is also control character)

3. How to define and represent different base in C? Write code to illustrate

```c
int binary_number = 0b1111;
int octal_number = 017;
int hex_number = 0xf;

printf("%d %d %d %d\n", binary_number, octal_number, decimal_number, hex_number); // 15 15 15 15
printf("%o %x %X\n", decimal_number, decimal_number, decimal_number); // 17 f F
```

4. How to define and use enumeration? Write code to describe

- Enumeration can help coder better understand code and enhance the readability of code

```
switch(day)
{
    case 1:    // it is confusing
        printf("Today is Monday\n");
        break;
    default:
        break;
}
```

- The usage is very flexible

```
enum weeks
{
    TEST_1,
    TEST_2,
    TEST_3 = 5,
    TEST_4,
    TEST_5 = TEST_2
};

printf("The default start value is %d, The next one increases by 1\n", TEST_1, TEST_2);
printf("The value can be changed %d, The next one increases by 1\n", TEST_3, TEST_4);
printf("The value can be changed %d like this form\n", TEST_5);
```

- It can also be combined with a typedef to define a type (leave it out for now,follow up later)

5. Why do we say string is constant? Illustration

- Compare the output of a and b with objdump -h to see which part is 7 bytes larger

```
a.
char *p;
p = "jerry";
printf("%s\n", p);

b.
char *p;
p = "jerry12345";
printf("%s\n", p);
```

- To check the address.Leave it for you

6. Can we change the value of a const variable? How to do?

It's different for global and local variables.
global variables declare by const are stored in .rodata, they cannot be modified anyway.
The corresponding local variable is stored on the stack and can be modified by pointer.

Variable

Guidance

1. Composition of variable
2. Data Type
3. local and global variable

Practice

The Linux tools objdump and readelf can help you know more about that how a variable be stored in memory

1. What are the rules for constitute the variables?

- The variable name consists of letters, digits and underscores, starting with a letter or an underscore. Normally a variable starting with a letter used for normal user, and starting with underscore used for system base functions.

```
int a, a_1, a1, a1_;    // valid
int 2a, 3_a, a#, a?;    // invalid
```

You can find here, There are really many functions which name is start with underscore
https://github.com/torvalds/linux/blob/master/include/linux/socket.h

- There are 32 reserved words in C programming language, the variable name defined cannot be the same with the reserved word.

```c
int for = 2, case = 3, if = 0;    // invalid, for/case/if are reserved words
```

- Variable names are case sensitive.

```c
int abc = 2, ABC = 3;           // variable abc and ABC are different
```

- Variable names should be as meaningful as possible Which can improve code readability

```c
// We can got more information from ip_bitmap and arp_outout than aa and bb
int ip_bitmap, arp_outout, aa, b;
```

2. What are the types of variables? What's the difference between them?

Misunderstanding

3. How to define and use variables? Assign and Use

- The variable need be define before to use it
- We can give the value when define it, The format is = ;
- We also can define it without value assign.The format is ;
- We need set the variable on left side and value on right side when do the assignment. =
- The variable can be used after it be assigned a value

4. What is the nature of a variable name? Illustration

Variable Name is only exist when the program was a C code. Then it will become a address.

5. Where do computer store local and global variable in memory? Topic 1->Practice->question

Refer to program "The address of constant and variable.c"

6. Why do we need static to declare a variable?

- When it used as a global variable, the scope of the variable changes to the file which define it, that is mean, the variable cannot be used in a source file other than the current file.

- When it used as a local variable, the variable is stored in the global variable table. The lifetime of the variable does not depend on the call stack.

7. What is different between local and global variable when it declare by static? And Where the computer store them in memory? Topic 1->Practice->question

Refer to program "The address of constant and variable.c"

Operators and Expressions

Guidance

Pointers On C.pdf -> Chapter 5

Practice

1. What are the rules of expression operation?

You can understand it in your own way and don't need to memorize it. In particular, the priority problem can be solved by parenthesis if you are unsure when programming.
This helps to improve the readability of the code, but it is necessary to know some priorities, otherwise adding too many brackets will affect the readability of the code.
This question is more subjective, you can judge for yourself, I give a few of my habits, as an example. It's not mandatory here.

```
// We know that the monadic operator takes precedence over the binocular operator,
// so we don't need parentheses here
if(!condition_a && *condition_b);

// The same is the monadic operator here, you can not add, but add parentheses,
// can make the code more readable
if((condition_a & condition_b) && (condition_c == NUM) || condition_d);
```

2. Pointers On C.pdf 5.8 question 5

```c
int is_leap_year = 0, year = 0;

is_leap_year = ((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0)
```

3. Pointers On C.pdf 5.8 question 6

side effect:There are some operators change the behavior of operands, and we call this change a side effect
The operators who have side effect:increment/decrement operator, assignment and compound assignment operator
Attention: Increment/Decrement operator is not recommended to use unless there is only one statement in the expression

```c
#include <stdio.h>
#define SQUARE(a) (a) * (a)

int main(void)
{
    int a, b;

    a = b = 1;
    printf("%d\n", a++ + ++b);                  // hard to memorize

    a = b = 1;
    printf("%d\n", ++a + ++b);                  // hard to memorize

    a = b = 1;
    printf("%d %d\n", SQUARE(a++), SQUARE(++b));    // Depending on the compiler, the result is uncertain

    return 0;
}
```

4. Pointers On C.pdf 5.8 question 7

Computers can't think as flexibly as people, and they need to follow the rules of operators.
Both operators have the same priority and are executed from left to right.

```c
1 <= a <= 10;        // Be equivalent to b = 1 <= a; b <= 10; b is non 0
1 <= a && a <= 10    // It should be like this, If you want to implement the original logic
```

5. Pointers On C.pdf 5.8 question 12

There are 2 kinds of way used for right shift in C, arithmetic or logical shift. The rules as following

unsigned	logical shift	fill the left always end with 0

singed	arithmetic shift	fill the left end with repetitions of the most significant bit

More detail to reference Computer Systems A Programmer's Perspective.pdf -> 2.1.10

For current question, we can do determine to use program as following

```c
int a = -1;

printf("%s\n", a == (a >> 3) ? "arithmetic" : "logical");
```

6. Pointers On C.pdf 5.9 question 3

There are 2 points need to note:

- To apply what you've learned, it's easier to use the shifts you've just learned.
- Integer type is not always 32 bits. The program should be portable

7. What is the output of the following program? Why?

```c
a.
int a[3][2] = {(1,2),(3,4),(5,6)}; // Be equivalent to {2, 4, 6};
printf("%d\n", a[0][1]);           // 4
The value of multi expression separated by comma operator is just the value of the last expression

b.
int x = 3, y = 0, z = 0;
if (x = y + z)
    printf("111\n");
else
    printf("222\n");        // output
The value of an assignment expression is the new value of the left operand.

c.
int a = 0, b = 0, c = 0, d = 0;
d = (c = (a = 11, b = 22)) == 11;
printf("%d %d\n", c, d);    // 22 0
a combination of concepts comma expression and assignment expression.
```

```
19
20 d.
21 int x = 3, y = 2, z = 3;
22 z = x < y ? !x : !((x =2) || (x = 3)) ? (x = 1): (y = 2);
23 printf("%d %d %d\n", x, y, z);
```

This answer for c and d get from Mormo

7(c)

The output is **22 0.** The reason is given in the below table:

No	Expression Procedure	Reason	a	b	c	d
1		Initial	0	0	0	0
2	d = (c = (a = 11, b = 22)) == 11	Assignment of a, b	11	22	0	0
3	d = (c = (b = 22)) == 11	Due to comma operator C is assigned 22	11	22	22	0
4	d = (c = 22) == 11	Due to assignment operator d is assigned to 22	11	22	22	22
5	d == 11	Relational Operator Returns 0 or 1	11	22	22	0

7(d)

The output is **2 2 2**

No	Expression Procedure	Reason	x	y	z
1	z = x < y ? !x :	Initial	3	2	3
2	Z = !((x=2) || (x=3)) ? (x = 1) : (y = 2)	Conditional operator goes false	3	2	3
3	Z = !(x) ? (x = 1) : (y = 2)	x=2 assignment is executed	2	2	3
4	Z = !(x) ? (x = 1) : (y = 2)	Conditional Operator goes false	2	2	3
5	Z = (y = 2)	y gets assigned, then z gets assigned	2	2	2

Macro & Typedef

Guidance

1. What is typedef? Is it used for create a new type?
2. Why do we need typedef?

3. Pointers On C.pdf 14.2
4. The name of micro & typedef(reference the Programming specification doc)

3.1 Naming

- Function names and variable names use lowercase with underscores.
- Macro definitions use uppercase with underscores.
- The name of typedef use lowercase end with _t.

Practice

1. What is the different between define and typedef? Illustration

	define	typedef
nature	replace text	type alias
timing	pre-processing stage	compile stage
mode	replace text only	a part of compile, including type check
function	Used to define constants or long string replace	Simplify complex declarations

Both define and typedef can help simplify code and improve its readability. But sometimes they are different

```
#define U_INT32_1 unsigned int*;
#typedef U_int32_2 unsigned int*;

U_INT32_1 a, b;
U_INT32_2 c, d;
```

2. How to define a macro function? What are the points for attention?

- Do not forget to put every expression enclosed in parenthesis

```
#define S(a,b) a*b
int value = S(3 + 1, 3 + 4);
printf("%d %d %d\n", value);
```

- Avoid to use the operator that may cause side effect

```
#include <stdio.h>
#define SQUARE(a) (a) * (a)

int main(void)
{
    int a = 1, b = 1;
    printf("%d %d\n", SQUARE(a++), SQUARE(++b));

    return 0;
}
```

- Do not end with semicolon, it's not a statement

3. Pointers On C.pdf 14.9 question 2

- Improve program readability. We can know the meaning of the constant from macro name easily
- Increase program maintainability. All it takes is one change if we need to change the value of macro

4. What is the type of T1?

```
struct foo
{
 int x;
 char y;
};

typedef struct foo *T1;
```

The type of T1 is struct foo *

5. Define a macro swap(t, x, y) that interchanges two arguments of type t.

```
#define SWAP(data_type, a, b) {data_type temp = a; a = b; b = temp;}
```

6. Write a macro MIN that accepts three parameters and returns the smallest one

```
#define MIN(x,y) ((x)<(y)?(x):(y))
#define MIN2(x,y,z) (MIN(x,y)<(z)?MIN(x,y):(z))
#define MIN3(x,y,z) MIN(x, MIN(y, z))

#define MIN(x,y,z) ((x)<(y)?((x)<(z)?(x):(z)):((y)<(z)?(y):(z)))
```

7. When do we need typedef? Illustration

I think it is better to use typedef here

- used for pointer

```
// normal pointer
typedef char * pchar_t;
int main (void)
{
  pchar_t str = "hello world!";
  printf("str: %s\n", str);
  return 0;
}
// pointer to function
typedef int (*func_t)(int a, int b);
int sum (int a, int b)
{
  return a + b;
}
int main (void)
{
  func_t fp = sum;
  printf ("%d\n", fp(1,2));
  return 0;
}
```

- Increased portability of code.

```
#ifdef PIC_16
    typedef unsigned long U32
#else
    typedef unsigned int U32
#endif
```

- Simplify complex definitions

```
int *(*array[10])(int *p, int len, char name[]);

typedef int *(*func_ptr_t)(int *p, int len, char name[]);
func_ptr_t array[10];
```

I do not think it is better to use typedef here

- Used for struct, It can save a keyword struct, but loses the information that it is a struct

```
struct student
{
  char name[20];
  int  age;
};
typedef struct student student_t;

struct student stu1;
student_t stu2;
```

- Used for array, Do not conducive to code readability, need to look at the definition

```
typedef int array_t[10];
array_t array;

int main (void)
{
  array[9] = 100;
  printf ("array[9] = %d\n", array[9]);
  return 0;
}
```

- Used for enumeration, It can save a keyword enum, but lose the information that it is a enumeration

```
enum color
{
  red,
  white,
  black,
  green,
  color_num,
};
typedef enum color color_t;

enum color color1 = black;
color_t    color2 = black;
```

8. What is the different between macro function and normal function?

	macro function	normal function
nature	replace text	function call stack
timing	pre-processing stage	running stage
advantage	Do not need new stack	Call function are not affected
disadvantage	Expanding makes the call function become larger, increasing the cost of calling the function	Need new stack and register

Topic 3:Control Flow

Selection Statement

Guidance

Pointers On C.pdf Chapter 4.1-4.4 4.8
The C Programming Language 2nd.pdf 3.1-3.4

1. What is the relationship between e**xpressions and statement?**
2. bool type

3. Why do we need to indent if the compile allow us write code like this?

```
if (A) if (B) C; else D;

if (A)
      if (B)
 C;
else
      D;
```

Practice

1. What is the function of control statements? What kind of form it can be?

Control statement is a statement used to control the selection, loop, turn, and return of a program flow.

Control expression can be any form

```
int fun(void){return 1;}

int main(void)
{
    int a, b, c;

    if (2 + 3);
    if (a % b);
    if (c = a / b);
    if (a = 1, b = 2);
    if (a > b? a : b);
    if (fun());
    if (a = fun());
    if (a == fun());
    if (a | fun());
    if (a || fun());

    return 0;
}
```

All uses of if can also be used to while/for/switch/return

2. How many control statements can C code have? How many categories can we divide them into?

There are 9 control statements be devided into 3 categories

category	control statements
Selection	if, switch
Loop	while, do while, for
Turn to	break, continue, return, goto

3. How many kinds of selection statements are there? What are their application scenarios?

- It is simpler to use if when the value of an expression is range, and to use switch when the value of an expression is definite

example:A class will be graded according to the requirements of 0-59: E-level 60-69: D-level 70-79: C-level 80-89: B-level 90-100: A-level

```c
if (score >=0 && score < 60)
    ;
else if (score >=60 && score < 69)
    ;
else if (score >= 70 && score < 79)
    ;
else if (score >= 80 && score < 89)
    ;
else if (score >= 90 && score <= 100)
    ;
else
    ;

// Need to find a way to change the range to a definite value
switch(score / 10)
{
    case 10:
    case 9:
        break;
    case 8:
        break;
    case 7:
        break;
    case 6:
        break;
    default:
        break;
}
```

- It is convenience to use if in multiple condition
- The control expression of if use logical expression and switch use arithmetic expression.Because the result of a logical expression is 0 or non-0

```c
if (condition_1)
    ;
else
    ;

switch(condition_1)
{
    case v1:
        break;
    default:
        break;
}
```

- Some compilers optimize the switch statement to make instruction execution more efficient.(Need more research)

4. How many forms of an if statement?

There are 3 forms of if statements

	if	if-else	if-elseif-else
description	only need logical true	logical true or false	Multiple logic or need arithmetic results

syntax	if (control expression)
	;
		if (control expression)
		;
		else
		;
			if (control expression 1)
			;
			else if (control expression 2)
			;
			else if (control expression ...)
			;
			else
			;

5. What should we pay attention to in the switch statement?

- The value of case must be a inter
- The case lable must be unique
- Do not forget to add after case statement otherwise a fall-though will happen
- Do not forget to add default statement end with all of the case,which can help you catch the case that you thoughtless
- The Statements that do not put in a case statement will not be executed

```c
#include <stdio.h>

int main(void)
{
    int a = 1, b = 0;

    switch(a)
    {
        b = 1;
        case 1:
            printf("111\n");
            break;
        case 2:
            printf("222\n");
            break;
        default:
            printf("333\n");
            break;
        b = 3;
    }

    printf("%d\n", b);        // 1

    return 0;
}
```

6. What is Dangling-else? Illustration

The rule of an else match with if is that it will match with the if closest with it.
Therefore, Something will happen that out of our expect if we wrote if without brace

```
// original intent:
if A is true then:
    if B is ture then:
        statement1
    else
        nothing to do
else
    statement2

// The code out of our expect
if (A)
    if (B)
        statement1;
    else
        statement2;
if A is true then:
    if B is ture:
        statement1
    else
        statement2
else
    nothing to do

// Corrent code
if (A)
{
    if (B)
        statement1;
}
else
    statement2;
```

7. What is switch fall-though? What are its advantages and disadvantages?

Advantages:Avoid writing duplicate code.Make the code look cleaner
Disadvantages:Multiple cases reuse the same code. If the logic of one branch need to change, it's easy to forget about fall-though, so that the logic of the other cases is also modified

8. Pointers On C.pdf 4.13 question 4

There are 2 approaches can be taken in this case

- empty statements
- get logical not of the control expression

```c
// emply statements
if (condition1)
    ;
else
    statement1

// logical not
if (!condition1)
    statement1
```

9. Pointers On C.pdf 4.13 question 16

```c
if (precipitating)
{
    if (temperature < 32)
        printf("snowing\n");
    else
        printf("raining\n");
}
else
{
    if (temperature < 60)
        printf("cold\n");
    else
        printf("warm\n");
}
```

Loop & Goto Statement

Guidance

Pointers On C.pdf Chapter 4.5-4.7 4.9
The C Programming Language 2nd.pdf 3.5-3.8

1. What is the execute order of the 3 parts of the for control expression?
2. How to convert between while and for?

Practice

1. How many kinds of loop statements are there? What are their application scenarios?

2. What is the different between following 2 programs? Give me an example to illustrate the impact of these two methods on the actual results

```
a.
while(expression_1)
{
    statement_1
}
b.
while(1)
{
    if(expression_1)
        statement_1
}
```

3. What are the difference between break, continue and return when we use them in a nested loop? Illustration
4. What are the difference between break and return when we use them in a case statement?
5. Pointers On C.pdf 5.8 question 8
6. What are advantages and disadvantages of goto statement?

Topic 4:Function

Basic Knowledge

Guidance

Pointers On C.pdf 3.5
Pointers On C.pdf 7.1-7.3 7.7-7.9

1. Can variables defined in one function be accessed in another function?
2. What will happen if you define a function with the static keyword?
3. Write a simple program and read the according assembly language
4. Function define/declaration
5. Inline function