

Tutorial 01: Review of C Programming

CSCI2520 - DATA STRUCTURES AND APPLICATIONS

TUTOR: ZHANG KAI

A solid blue horizontal bar at the bottom of the slide.

Outlines

1. Tutorial Arrangements
2. Contact of Tutors
2. Review of C programming

Tutorial Arrangements

- Time: Thu 5:30 pm - 6:15 pm
- Venue: William M W Mong Eng Bldg 404
- Material Download:
Blackboard: <https://blackboard.cuhk.edu.hk/>

Contact of Tutors

ZHANG Kai 张凯

- Room 101, Ho Sin Hang Engineering Building
- kzhang@cse.cuhk.edu.hk

ZHENG Chenguang 郑晨光

- Room 122, Ho Sin Hang Engineering Building
- cgzheng@cse.cuhk.edu.hk

Review of C Programming

Program Structure:

This is an example of c program which will output "Hello World!" first and then the user need to input an integer to be the value of b, and it will output the value of both a and b.

```
#include <stdio.h>
int main() {
    printf("Hello World!\n"); // output
    int a = 10;
    int b;
    scanf("%d", &b); // get value of b
    printf("a = %d, b = %d.\n", a, b);
    //output
    return 0;
}
```

```
Hello World!
1
a = 10, b = 1.
```

Blue for output and red for input.

Review of C Programming

Basic Syntax:

1. include

```
#include <stdio.h> // standard input & output header file  
#include "myhead.h" // user defined header file
```

- Please note: for standard header file using `< >` and user-defined header file using `" "`.

2. main()

```
int main(){  
    // ...  
    return 0;  
}
```

```
void main(){  
    // ...  
}
```

- For `int` `main()`, you need `return 0`; `void` `main()` not.

Review of C Programming

Basic Syntax:

3. variable declaration

```
int a = 10, b = 20;  
float c, d = 30;
```

- Result: a = 10, b = 20, d = 30.0, the value of c is not specified.

4. scanf() and printf() functions

```
scanf("%f", &c);  
printf("c = %f, d = %f.\n", c, d);
```

- `\n` for a new line.
- `scanf()` need the address of variable, so we use `&c` here.

```
15.4  
c = 15.400000, d = 30.000000.
```

Blue for output and red for input.

Review of C Programming

Keywords in C Programming

<u>auto</u>	<u>break</u>	<u>case</u>	<u>char</u>
<u>const</u>	<u>continue</u>	<u>default</u>	<u>do</u>
<u>double</u>	<u>else</u>	<u>enum</u>	<u>extern</u>
<u>float</u>	<u>for</u>	<u>goto</u>	<u>if</u>
<u>int</u>	<u>long</u>	<u>register</u>	<u>return</u>
<u>short</u>	<u>signed</u>	<u>sizeof</u>	<u>static</u>
<u>struct</u>	<u>switch</u>	<u>typedef</u>	<u>union</u>
<u>unsigned</u>	<u>void</u>	<u>volatile</u>	<u>while</u>

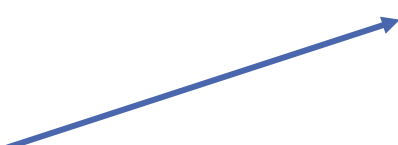
Reference: <https://www.programiz.com/c-programming/list-all-keywords-c-language>

Review of C Programming

Conditional Statement:


1. if else

- The statements inside the body of “if” only execute if the given condition returns true. If the condition returns false then the statements inside “if” are skipped.



```
if (condition){  
    // ...  
}
```

- If condition returns true then the statements inside the body of “if” are executed and the statements inside body of “else” are skipped.
If condition returns false then the statements inside the body of “if” are skipped and the statements in “else” are executed.




```
if (condition){  
    // ...  
} else{  
    // ...  
}
```

Review of C Programming

Conditional Statement:

1. if else

- The else..if statement is useful when you need to check multiple conditions within the program, nesting of if-else blocks can be avoided using else..if statement.
- The last else statement is optional. 

```
if (condition 1){  
    // ...  
} else if (condition 2){  
    // ...  
} else if (condition 3){  
    // ...  
}  
...  
else {  
    // ...  
}
```

Review of C Programming

Conditional Statement:

2. switch case

```
switch (variable or an integer expression){  
    case constant-1:  
        //C Statements  
    case constant-2:  
        //C Statements  
    default:  
        //C Statements  
}
```

- The **switch case statement** is used when we have multiple options and we need to perform a different task for each option.
- Usually we need **break;** for every case.

Review of C Programming

Conditional Statement:

2. switch case

```
int i=2;
switch (i){
    case 1:
        printf("Case1 ");
    case 2:
        printf("Case2 ");
    case 3:
        printf("Case3 ");
    default:
        printf("Default ");
}
```

Output: Case2 Case3 Default

```
int i=2;
switch (i){
    case 1:
        printf("Case1 ");
        break;
    case 2:
        printf("Case2 ");
        break;
    case 3:
        printf("Case3 ");
        break;
    default:
        printf("Default ");
}
```

Output: Case2

Review of C Programming

Loop Statement:

1. for loop

```
for (initialization; condition test; increment or decrement){  
    //Statements to be executed repeatedly  
}
```

- **Step 1:** First initialization happens and the counter variable gets initialized.
- **Step 2:** In the second step the **condition** is checked, where the counter variable is tested for the given condition, if the condition returns true then the C statements inside the body of for loop gets executed, if the condition returns false then the for loop gets terminated and the control comes out of the loop.
- **Step 3:** After successful execution of statements inside the body of loop, the counter variable is **incremented or decremented**, depending on the operation (++ or --).

Review of C Programming

Loop Statement:

2. while loop

```
while (condition test){  
    //Statements to be executed repeatedly  
    // Increment (++) or Decrement (--) Operation  
}
```

3. do while loop

```
do{  
    //Statements to be executed repeatedly  
}while(condition test);
```

- do while loop will run the statements at least once.

Review of C Programming

continue & break Statement:

1. continue statement

- The **continue statement** is used inside **loops**.
- When a continue statement is encountered inside a loop, control jumps to the beginning of the loop for next iteration, skipping the execution of statements inside the body of loop for the current iteration.

```
for (int j=0; j<=8; j++){  
    if (j==4){  
        continue;  
    }  
    printf("%d ", j);  
}
```

Output: 0 1 2 3 5 6 7 8

Review of C Programming

continue & break Statement:

2. break statement

- It is used to come out of the loop instantly. When a break statement is encountered inside a loop, the control directly comes out of loop and the loop gets terminated.
- This can also be used in switch case control structure. Whenever it is encountered in switch-case block, the control comes out of the switch-case.

```
for (int j=0; j<=8; j++){  
    if (j==4){  
        break;  
    }  
    printf("%d ", j);  
}
```

Output: 0 1 2 3

Review of C Programming

User defined functions:

```
return_type function_name (argument list){  
    // Set of statements - Block of code  
}
```

- **return_type**: Return type can be of any data type such as int, double, char, void, short etc. Don't worry you will understand these terms better once you go through the examples below.
- **function_name**: It can be anything, however it is advised to have a meaningful name for the functions so that it would be easy to understand the purpose of function just by seeing it's name.
- **argument list**: Argument list contains variables names along with their data types. These arguments are kind of inputs for the function. For example – A function which is used to add two integer variables, will be having two integer argument.
- **Block of code**: Set of C statements, which will be executed whenever a call will be made to the function.

Review of C Programming

User defined functions:

Example:

```
int addition(int num1, int num2){  
    int sum;  
    sum = num1+num2;  
    return sum;  
}
```

```
int main() {  
    int var1 = 10, var2 = 20;  
    int res = addition(var1, var2);  
    printf ("Output: %d", res);  
    return 0;  
}
```

Output: 30

Review of C Programming

Structure:

```
struct struct_name {  
    DataType member1_name;  
    DataType member2_name;  
    DataType member3_name;  
    ...  
};  
struct struct_name var_name;
```

How to access data members of a structure using a struct variable?

`var_name.member1_name; var_name.member2_name; ...`

How to assign values to structure members?

1) Using Dot(.) operator `var_name.memeber1_name = value1;`

2) All members assigned in one statement

`struct struct_name var_name = {value for memeber1, value for memeber2 ...so on for all the members}`

Review of C Programming

Typedef statement:

typedef is used to create an alias name for another data type.

```
struct struct_name {  
    DataType member1_name;  
    DataType member2_name;  
    DataType member3_name;  
    ...  
};  
  
typedef struct struct_name my_type;  
my_type var_name;
```

Review of C Programming

Pointer:

A **pointer** is a variable that stores the **address** of another variable. Unlike other variables that hold values of a certain type, pointer holds the address of a variable. For example, an integer variable holds (or you can say stores) an integer value, however an integer pointer holds the address of an integer variable.

Important point to note is: The **data type of pointer and the variable must match**, an int pointer can hold the address of int variable, similarly a pointer declared with float data type can hold the address of a float variable.

Review of C Programming

Pointer:

```
int num = 10;  
int *p;  
p = &num;  
printf("Address of variable num is: %p\n", p);  
printf("Address of variable num is: %d\n", *p);
```

```
Address of variable num is: 0x7ffee99a53b8  
Address of variable num is: 10
```

“Address of”(&) Operator

“Value at Address”(*) Operator

Review of C Programming

Array:

```
int num[35]; // An integer array of 35 elements
char ch[10]; // An array of characters for 10 elements
```

1. How to access element of an array in C ?

- You can use **array subscript** (or index) to access any element stored in array. Subscript **starts with 0**, which means **arr[0]** represents the first element in the array **arr**.
- In general **arr[n-1]** can be used to access **nth element of an array**, where n is any integer number.

2. Pointer to array

- Array elements can be accessed and manipulated using **pointers in C**. Using pointers you can easily handle array. You can have access of all the elements of an array just by assigning the array's base address to pointer variable.

Review of C Programming

Array:

```
int val[7] = { 11, 22, 33, 44, 55, 66, 77 } ;  
for ( int i = 0 ; i < 7 ; i++ ) {  
    printf("val[%d]: value is %d and address is %p\n", i, val[i], &val[i]);  
}
```

```
val[0]: value is 11 and address is 0x7ffeea32e390  
val[1]: value is 22 and address is 0x7ffeea32e394  
val[2]: value is 33 and address is 0x7ffeea32e398  
val[3]: value is 44 and address is 0x7ffeea32e39c  
val[4]: value is 55 and address is 0x7ffeea32e3a0  
val[5]: value is 66 and address is 0x7ffeea32e3a4  
val[6]: value is 77 and address is 0x7ffeea32e3a8
```


Review of C Programming

Array:

```
int *p;  
int val[7] = { 11, 22, 33, 44, 55, 66, 77 } ;  
p = &val[0];  
for ( int i = 0 ; i<7 ; i++ ){  
    printf("val[%d]: value is %d and address is %p\n", i, *p, p);  
    p++;  
}
```

val[0]:	value is 11	and address is 0x7ffeea32e390
val[1]:	value is 22	and address is 0x7ffeea32e394
val[2]:	value is 33	and address is 0x7ffeea32e398
val[3]:	value is 44	and address is 0x7ffeea32e39c
val[4]:	value is 55	and address is 0x7ffeea32e3a0
val[5]:	value is 66	and address is 0x7ffeea32e3a4
val[6]:	value is 77	and address is 0x7ffeea32e3a8

Review of C Programming

Malloc() function:

The C library function **void *malloc(size_t size)** allocates the requested memory and returns a pointer to it.

```
char *str;  
str = (char *) malloc(12);  
strcpy(str, "ds_tutorial");  
  
printf("String = %s\n", str);  
for (int i = 0; i < 12; i++) {  
    printf("%c", str[i]);  
}
```

```
String = ds_tutorial  
ds_tutorial
```