STUDY OF C PROGRAMMING LEARNING ASSISTANT SYSTEM (CPLAS)

Basic C Programming Language (C_VTP1) file

Funabiki Laboratory

Department of Electrical and Communication Engineering

Okayama University

C Programming Learning Assistant System (CPLAS)

- Data Type Usage 5
- Hierarchy Of Operations 5
- Simple Interest Rate Calculation 6
- If Statement Usage 7
- If_Else Usage 7
- Nested If_Else Usage 8
- Logical Operator Usage 9
- Relational Operator Usage On Integers 10
- While Loop With Postincrementation 11
- While Loop Usage 11
- For Loop Usage 12
- Nested For Loop Usage 13
- Do While Loop Usage 14
- Switch-Case Usage Without Break 14
- Arithmetic Expression Usage In Switch 15
- Many Cases In Switch-Case Usage 16
- Simple Function Usage 16
- Function Usage With Call By Value 17
- Function Usage With Call by Reference 18

Correspondence Between Each Topic and Related VTPs

- Simple Factorical Calculation 19
- Factorical Calculation With Recursion 21
- Auto Variable With Different Blocks 22
- Various Macro Template Usage 22
- Macro Arguments Usage 23
- Simple Array Usage 23
- Array With Various Function Call 24
- Pointer Usage For Different Data Types 26
- Various Accessing Methods In Array 27
- Pointer In 2-D Array 27
- String Length Calculation 28
- String Copy Calculation 28
- String Concatenation 29
- Element Exchange In 2-D Array 29
- Accessing Structure Element 30
- Copying Structure Element 31
- Nested Structure Element 31
- Passing Structure Element To Method 32
- Pointer Usage With Structure Element 32

Correspondence Between Each Topic and Related VTPs

Correspondence Between Each Topic and Related VTPs

※ Click the link to go reference pages directly.

C Program Concepts	Related VTPs Version	Related Problem Numbers
Data Type Usage	C_VTP1	1
Hierarchy Of Operations	C_VTP1	2
Simple Interest Rate Calculation	C_VTP1	3
If Statement Usage	C_VTP1	4
If_Else Usage	C_VTP1	5
Nested If_Else Usage	C_VTP1	6
Logical Operator Usage	C_VTP1	7
Relational Operator Usage On Integers	C_VTP1	8
While Loop With Postincrementation	C_VTP1	9
While Loop Usage	C_VTP1	10
For Loop Usage	C_VTP1	11
Nested For Loop Usage	C_VTP1	12
Do While Loop Usage	C_VTP1	13
Switch-Case Usage Without Break	C_VTP1	14
Arithmetic Expression Using In Switch	C_VTP1	15
Many Cases In Switch-Case Usage	C_VTP1	16
Simple Function Usage	C_VTP1	17
Function Usage With Call By Value	C_VTP1	18
Function Usage With Call By Reference	C_VTP1	19
Simple Factorial Calculation	C_VTP1	20
Factorial Calculation With Recursion	C_VTP1	21
Auto Variable With Different Blocks	C_VTP1	22
Various Macro Templates Usage	C_VTP1	23
Macro Arguments Usage	C_VTP1	24
Simple Array Usage	C_VTP1	25
Array With Various Function Call	C_VTP1	26
Pointer Usage For Different Data Types	C_VTP1	27
Various Accessing Methods In Array	C_VTP1	28
Pointer In 2-D Array	C_VTP1	29
String Length Calculation	C_VTP1	30
String Copy Calculation	C_VTP1	31
String Concatenation	C_VTP1	32
Element Exchange In 2-D Array	C_VTP1	33
Accessing Structure Element	C_VTP1	34
Copying Structure Elements	C_VTP1	35
Nested Structure Element	C_VTP1	36
Passing Structure Element To Method	C_VTP1	37
Pointer Usage With Structure Element	C_VTP1	38

Correspondence Between Each Topic and Related VTPs

1. Data Types Usage

A data type defines a collection of data **values** and a set of predefined **operations** on those values.

Data declare (variable) $\longrightarrow \boxed{\text{Type name}}$

In C programming, data types are declarations for variables. This determines the type and size of data associated with variables. For example,

```
int a
a=1
```

Here, *a* is a variable of int (integer) type. The value of a is 1.

char is used for declaring **character** type variables. For example,

```
char test = ' h';
```

Here, h is a variable of char type.

char array, char c[8]
"JOHn"

0	1	2	3	4	5	6	7	
J	O	H	n	//	//	//	//	C

c[0] = 'J'; c [1] = 'O'; c[2] = 'H'; c[3] ='n'

%d is used for outputting a decimal integer.

%f is used for outputting a real number in decimal form when the input data is float.

%c is used for outputting a character.

printf means to produce output.

2. Hierarchy Of Operations

The hierarchy of operations instructs the compilers and interpreters on the order in which the expression has to be executed. Operator precedence in C is also similar to that in most other languages. Division and multiplication occur first, then addition and subtraction. The result of the calculation 5+3*4 is 17, not 32, because the * operator has higher precedence than + in C. You can use parentheses to change the normal precedence ordering:

(5+3)*4 is 32. The 5+3 is evaluated first because it is in parentheses.

Operators	(binary operator)	
Arithmetic		for example, x=100, y= 50
Addition +	z= x+y	z=150
Subtraction -	z= x-y	z= 50
Multiplying *	z= x*y	z= 500
Division /	z= x/y	z= 2
Reminder %	z= x%y	z=0

%. 0f – to get the integer value.

\ **n** - new line

Example:

```
main ( )
{
    int m, n, s, t, x, y, z, ans1, ans2;
    m =2, n =3, s =4, t =5;
    x =3, y =2, z =6;
    ans1 = 2* x* y+2;
    ans2 = (m + n) / (s + t);

    printf  ("\n %d" , ans1);
    printf  ("\n%.0f" , ans2) ;
}
```

Output: ans1 = 24
ans2= 0

3. **Calculation Of Simple Interest**

Write a C program to input principle, time and rate (P, T, R) from user and find Simple Interest. Simple interest formula is given by.

$$SI = \frac{P \times T \times R}{100}$$

Example:

```
Input
Enter principle: 1200
Enter time: 2
Enter rate:  5.4
Output
Simple Interest = 129.6
```

4. If Statement

➢ 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.**

Example:

```
#include < stdio.h >
int main ( )
{
      int x = 25;
      int y = 30;
  if (x < y)
      {
     printf (" x is less than y");

      }
    return 0;
}
```

Output: x is less than y

5. If Else Statement

➢ 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.

Example:

```
#include < stdio.h >
int main ( )
{
     int age;
    printf("Enter your age:");
    scanf("%d",&age);
    if(age  >=18)
        {
            printf ("You are eligible for voting");
         }
   else
       {
            printf ("You are not eligible for voting");
        }
  return 0;
}
```

Output:
```
Enter your age: 14
You are not eligible for voting
```

6. Nested If Statement

- Nested If in C Programming is placing If Statement inside another IF Statement. Nested If in C is helpful if you want to check the condition inside a condition.
- If the condition fails, we will check one more condition (Nested If), and if it succeeds, we print something. When the nested If the condition fails, we print some other thing.

Example:

```
int main ( )
{
            int a, b;
            printf ("Input the value of a");
            scanf ("%d", &a);
            printf("Input the value of b");
            scanf("%d",&b);
            if  (a != b)
             {
                        printf("a is not equal to b \n");  // Nested if else
                        If (a > b)
                           {
```

Correspondence Between Each Topic and Related VTPs

```
                                        printf ("a is greater than b \n");
                               }
                          else
                              {
                                   printf (“b is greater than a \n”);
                              }
                  }
                 else
                  {
                            printf ("a is equal to b \n");
                  }
         return 0;
     }
```

Output:

```
Input the value of a: 12
Input the value of b: 21
a is not equal to b
b is greater than a
```

- **return 0** used for returning a value when exiting the function
- & = call by reference

7. Working Of Logical Operators

AND &&
OR ||
NOT ! (is not equal)

They are used to combine two or more conditions/constraints or to complement the evaluation of the original condition under consideration.

- **Logical AND operator:** The **‘&&’** operator returns true when both the conditions under consideration are satisfied. Otherwise it returns false. For example, **a && b** returns true when both **a** and **b** are true (i.e. non-zero).
- **Logical OR operator:** The **‘||’** operator returns true even if one (or both) of the conditions under consideration is satisfied. Otherwise it returns false. For example, **a || b** returns true if one of a or b or both are true (i.e. non-zero). Of course, it returns true when both a and b are true.
- **Logical NOT operator:** The **‘!’** operator returns true the condition in consideration is not satisfied. Otherwise it returns false. For example, **!a** returns true if a is false, i.e. when a=0.

Example:

```
// Logical AND example
int main ( )
```

```
  {
     int a,b;
a= 20, b= 15,  c=  10, d= 5;
if (a>b && c == d)
            printf("I am a programmer.\n");
else
            printf("I am not a programmer\n");

// Logical OR example
if (a>b  | |   c == d)
            printf("I am a programmer\n");
else
            printf("I am not a programmer\n");

// Logical NOT example
if ( !b)
            printf (“ b is zero. \n");
else
            printf (“ b is not zero. \n");
}
```

Output:

```
I am not a programmer.
I am a programmer.
b is not zero.
```

8. Using Relational Operations On Integers

= =	equal	(for e.g), x == 10
!=	not equal	x! = 10
>	greater than	x>10
<	less than	x<10
>=	greater than equal	x>10 (or) x == 10
<=	less than equal	x<10 (or) x == 10

Example:

```
int main ( )
{
            int i = 2;
            int j = 9;
 // The output will be printed like o or 1
printf(“ i >  j:   %d     \n", i > j );
printf("i  >=  j  %d     \n", i >= j);
printf(“i  <=  j: %d     \n", i <= j);
```

```
    printf(" i <    j:  %d    \n", i < j );
    printf(" i == j  %d    \n", i == j);
    printf (" i != j  %d    \n", i != j );
    }
```

Output:
```
i  > j:   0
i >=j:  0
i <=j:  1
i  < j:  1
i ==j:  0
i  !=j:  1
```

9. While Loop With Post Incrementation

Post-increment operator: A post-increment operator is used to increment the value of variable after executing expression completely in which post increment is used. In the Post-Increment, value is first used in a expression and then incremented.

Example:

```
main ( )
{
int k =  1;
        while ( k ++< 12)
                printf ( "%d \t", k);
}
```

Output: The values of k are: 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12

Example:

```
main ( )
{
                int j = 0;
while ( j ++< 6)
                printf ("%d \t", j);
}
```

Output: The values of j are 1, 2, 3, 4, 5, 6

\t - tab

10. **While Loop**

In programming, a loop is used to repeat a block of code until the specified condition is met. C programming has three types of loops:

Correspondence Between Each Topic and Related VTPs

1. for loop
2. while loop
3. do...while loop

While Loop Works

- The while loop evaluates the test expression inside the parenthesis ().
- If the test expression is true, statements inside the body of while loop are executed. Then, the test expression is evaluated again.
- The process goes on until the test expression is evaluated to false.
- If the test expression is false, the loop terminates (ends).

++ increment x++, ++x   x+=1

- - decrement x- -, --x, x - = 1

- int x= 10; output x// 10
  x++
  Output x // 11

Example:

```
int main ( )
{
          int i=0;
          while (i += 3, i <10)
         {
                        printf ( "%d ",i);
         }
         printf("%d" , i);
}
```

Output: 3, 6, 9, 12

11. For Loop Works

- The initialization statement is executed only once.
- Then, the test expression is evaluated. If the test expression is evaluated to false, the for loop is terminated.

- However, if the test expression is evaluated to true, statements inside the body of for loop are executed, and the update expression is updated.
- Again the test expression is evaluated.
- This process goes on until the test expression is false. When the test expression is false, the loop terminates.

Example:

```
main ( )
{
        For ( int i = 0;  i<= 4 );
        {
                    printf ("%d\n", i);
                    i = i + 1;
        }
}
```

Output: The values of i are: 0, 1, 2, 3, 4

12. Nested For Loop

- Using a for loop within another for loop is said to be **nested for loop**. In nested for loop one or more statements can be included in the body of the loop. In nested for loop, the number of iterations will be equal to the number of iterations in the outer loop multiplies by the number of iterations in the inner loop.

Example:

```
main ( )
{
        int a, b, sum;
        for ( a = 0 ;   a <= 1 ; r++ ) /* outer loop */
        {
            for   (  b=1 ; b <= 2; c++ ) /* inner loop */
             {
                    sum =  r + c;
                    printf ( "r = %d c = %d sum = %d\n", r, c, sum ) ;
             }
         }
}
```

Correspondence Between Each Topic and Related VTPs

Output: a =0, b=1, sum= 1

a =1, b=2, sum= 3

13. Do...while Loop Works

- The body of do...while loop is executed once. Only then, the test expression is evaluated.
- If the test expression is true, the body of the loop is executed again and the test expression is evaluated.
- This process goes on until the test expression becomes false.
- If the test expression is false, the loop ends.

Example:

```
    int main ( )
{
        int x = 3, y = 1;
        do
        {
                        printf("\t %d %d \n", x,y);
                        x--  ;
                        y++ ;
        } while (x >= 0);
}
```

Output: The values of x and y are: 3, 1, 2, 2, 1, 3, 0, 4

14. Using Switch-Case Control Structure Without Break

- **Switch statement in C** tests the value of a variable and compares it with multiple cases. Once the case match is found, a block of statements associated with that particular case is executed.
- Each case in a block of a switch has a different name/number which is referred to as an identifier.
- The value provided by the user is compared with all the cases inside the switch block until the match is found.
- If a case match is NOT found, then the default statement is executed, and the control goes out of the switch block.

If there is no break statement then the cases matched other than default will get executed.

Correspondence Between Each Topic and Related VTPs

Example:

```
main ( )
{
        char a = 3 ;

        switch ( a )
        {
            case 1 :
            printf ("\n Apple" );

            case 2 :
            printf ( “\n Banana " );

            case 3 :
            printf ( “\n Orange " );

          default :
          printf ( "\n Fruit") ;
        }

        printf ( "\n I like fruits." ) ;
}
```

Output:

```
Orange
Fruit
I like fruits.
```

15. Switch-Case Using Arithmetic Expression

```
main ( )
{
        int t,  k=1 ;
        switch  (  t =  k+1 ) ;
        {
            case 0 :
            printf ( "\n%d\t Demonstrator ",t) ;
            break;

            case 1 :
            printf ( "\n%d\t Tutor", t) ;
            break;

          default :
          printf (“\n%d \t All are University Teachers!",t ) ;
         }
}
```

Correspondence Between Each Topic and Related VTPs

Output: 2 All are University Teachers!

16. Using Switch-Case With Multiple Cases

```
main ( )
{
            int x = ‘ 1 ’+ ‘ 2 ’  ;
           char str [  ] = "Morning!";

            switch ( x )

            {
                case 'a' :

                case 'b' :
                printf ( "%s\t You entered b.",str ) ;
                case ‘A’ :
                printf ( "%s\tYou entered a.",str ) ;

           case '2' + '1' :
           printf ( "%s\tYou entered a and b.",str ) ;
           }
}
```

Output: Morning! You entered a and b. **(%s** is used for outputting a string.)

17. Why We Need Functions In C

Functions are used because of following reasons –

a) To improve the readability of code.

b) Improves the reusability of the code, same function can be used in any program rather than writing the same code from scratch.

c) Debugging of the code would be easier if you use functions, as errors are easy to be traced.

d) Reduces the size of the code, duplicate set of statements are replaced by function calls.

Actual parameters: The parameters that appear in function calls.

Formal parameters: The parameters that appear in function declarations.

1) **Function – Call by value method** – In the call by value method the actual arguments are copied to the formal arguments, hence any operation performed by function on arguments doesn’t affect actual parameters.

Correspondence Between Each Topic and Related VTPs

2) **Function – Call by reference method** – Unlike call by value, in this method, address of actual arguments (or parameters) is passed to the formal parameters, which means any operation performed on formal parameters affects the value of actual parameters.

Simple Usage Of Functions

Example:

```
main ( )
{
            printf ( "\n   I am a teacher " ) ;
            Germany  (  ) ;
            italy ( );
            England (  );
}
            gremany(  );
      {
            printf  (“ \n   I am in Germany " );
       }

            italy ( );
       {
            printf (“ \n   I am in italy " );
        }
            England ( );
       {
            printf ((“ \n   I am in England " );

       }
```

Output: I am a teacher

```
            I am in Germany
            I am in Italy
            I am in England
```

18. Call By Value In C

- A copy of the value is passed into the function
- Changes made inside the function is limited to the function only. The values of the actual parameters do not change by changing the formal parameters.
- Actual and formal arguments are created at the different memory location

Usage of Function with Call by Value

```
main ( )
{
            int a = 5, b = 10 ;

            printf ( "\n Before swap, a = %d b = %d", a, b ) ;
            swap ( a, b ) ;
            printf ("\nAfter swap, a = %d b = %d", a, b ) ;
 }

swap   (int p, int q)
{
    int t ;

    t = p;
    printf ("\nThe value of t = %d " , t);
    p= q;
    printf ( "\nThe value of x = %d ", p) ;
   q= t;
   printf ( ""\nThe value of y = %d ", q) ;
}
```

Output:
```
Before swap, a =5, b= 10
The value of t= 5
The value of p = 10
The value of q = 5
After swap, a = 5, b = 10
```

19. Call By Reference In C

- An address of value is passed into the function
- Changes made inside the function validate outside of the function also. The values of the actual parameters do change by changing the formal parameters.
- Actual and formal arguments are created at the same memory location

& = call by reference

Function With Call By Reference Usage

Example:

```
main ( )
```

```
{
            int a = 15, b = 20 ;

            printf ( "\n Before swap, a = %d b = %d", a, b ) ;
            swapr ( &a, &b ) ;
            printf ("\nAfter swap, a = %d b = %d", a, b ) ;
}
swapr  (int *p, int *q)
    {
            int t ;

           t =  *p;
            printf ("\nThe value of t = %d " , t);

            *p= *q;
            printf ( "\nThe value of x = %d ", p) ;

            *q= t;
            printf ( ""\nThe value of y = %d ", q) ;
    }
```

Output: Before swap, a = 15, b = 20

```
            The value of t    =15
            The value of *p =20
            The value of *q = 15
           After swap, a = 20, b = 15.
```

20. Finding Factorial Value

The factorial of a positive number *n* is given by:

Factorial of n (n!) = 1 * 2 * 3 * 4 *... * n

Factorial of a positive integer n, denoted by n!, is the product of all positive *descending* integers less than or equal to n. For example:

1. 5! = 5*4*3*2*1 = 120

2. 3! = 3*2*1 = 6

Example: 1

```
main
{
            int a, fact ;
            a = 2
```

```
        fact = factorial ( a ) ;
        printf ( "\n Factorial value = %d", fact ) ;
}
factorical ( int x )
{
            int f = 1, i,c;
            c =1 ;

            for ( i = x ;  i >= 1,  i-- )
            {
                        f = f * i ;
                        printf ( "\nThe value of f%d = %d", c, f ) ;
                        c++;
            }
            return ( f );
}
```

```
Output:   The value of f1 = 2
            The value of f2 = 2
            The value of f3 = 0
            Factorial value  = 2
```

Example: 2

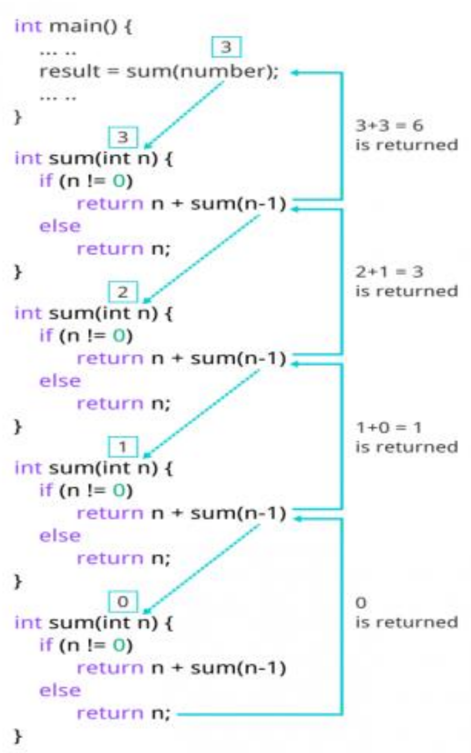
```
main ( )
{
            int x, fact ;
            x = 4 ;

            fact = factorial ( x ) ;
            printf ( "\n Factorial value = %d", fact ) ;
}
factorical ( int a )
{
            int f = 1, i, c;
            c =1 ;
        for ( i = a;  i >= 0,  i-- )
            {
                        f = f * i ;
                        printf ( "\nThe value of f%d = %d", c, f ) ;
                        c++;
             }
              return (f);
}
```

Output:

```
The value of f1 =4
The value of f2 = 12
The value of f3 = 24
The value of f4 =24
Factorical value = 24
```

21. Recursion In C

Recursion is a process in which a function calls itself directly or indirectly.



```
int main() {
    ... ..
    result = sum(number);
    ... ..
}
int sum(int n) {
    if (n != 0)
        return n + sum(n-1)
    else
        return n;
}
int sum(int n) {
    if (n != 0)
        return n + sum(n-1)
    else
        return n;
}
int sum(int n) {
    if (n != 0)
        return n + sum(n-1)
    else
        return n;
}
int sum(int n) {
    if (n != 0)
        return n + sum(n-1)
    else
        return n;
}
```

Finding Factorial Value With Recursion

Example:

```
main ( )
{
            Int  x , fact;
            x = 3 ;
        fact = rec ( x ) ;
        printf ( "\n Factorial value = %d", fact ) ;
}
```

```
rec  ( int  a)
{
               int f,   c = 1

                if ( a == 1 )
               return ( 1 ) ;
else
               f = a * rec ( a - 1 ) ;
          return ( f ) ;
}
```

Output: Factorial value = 6

22. Initializing The Automatic Variables With Different Blocks

In computer programming, an **automatic variable** is a local variable which is allocated and deal located automatically when program flow enters and leaves the variable's scope.

Example:

```
main ( )
{
               auto int j =3;
                {
                    auto int j = 2 ;
                     {
                       auto int j = 1 ;
                       printf ("\n%d ", j ) ;
                       }
            printf ("\n%d ", j) ;
                }
    printf ("\n%d ", j ) ;
}
```

Output: 1, 2, 3

23. Different Macro Templates Usage

- Usually macros make the program run faster but increase the program size, whereas functions make the program smaller and compact.
- This gets avoided with macros since they have already been expanded and placed in the source code before compilation.

Example:

```
#define PI 3.142
#define AND &&
#define OR ||
main( )
{           float r = 4.2 ;
            float area ;
            int f = 1, x = 4, y = 80 ;

            area = PI * r * r ;
           printf ( "\n Area of circle = %.2f", area ) ;

            if ( ( f < 3 ) AND ( x <= 10 OR y <= 65 ) )
              printf ( "\n You are University students " ) ;
            else
              printf ( "\n They are clever " ) ;
}
```

Output: Area of circle - 55.42
another output - You are University students

24. Macro Usage With Arguments

Example:

```
#define   Area (a)   ( 3.142 * a * a)

main( )
{
            float r1 = 4.25 ; r2 = 3.0,   A;

             A  = area (r1) ;
             printf ( "\n Area of circle1 = %f", A) ;

             A  = area (r2)  ;
             printf  ( "\n Area of circle2 = %f", A );
}
```

Output: Area of circle1 (only two decimal place) = 56.75
Area of circle2 (only two decimal place) = 28.27

25. Array

➢ An array is a variable that can store multiple values.

For example,

if you want to store 100 integers, you can create an array for it.

```
int data[100];
```

Correspondence Between Each Topic and Related VTPs

For example,

```
float mark[5];
```

Here, we declared an array, *mark*, of floating-point type. And its size is 5. Meaning, it can hold 5 floating-point values. It is possible to initialize an array during declaration.

For example,

int mark[5] = {19, 10, 8, 17, 9};
mark[0] is equal to 19
mark[1] is equal to 10
mark[2] is equal to 8
mark[3] is equal to 17
mark[4] is equal to 9

Simple Program Using Array

Example:

```
main ( )
{
            int avg,  sum = 0 ;

            int marks[ ] = { 19, 10, 8, 17, 9 };  /* array declaration*/

            for (int i = 0; i < 5 ; i++ )
              sum = sum + marks[ i ] ;  / * read data from array*/


            avg = sum / 5 ;
            printf (  avg = sum "\n Average marks = %d", avg ) ;
}
```

Output: Average marks – 55.8

26. Different Function Calls With By Value and By Reference

```
main ( )
{
            int marks [  ] = { 3,5,7,9 };

      for  (int i = 0   i < =  2 ; i++ )
        {
            display1 (  marks [ i ] )  ; // call by value
```

Correspondence Between Each Topic and Related VTPs

```
                display2 ( &marks[ i ] ) ; // call by reference
            }
    }
    display 1 ( int m)
    {
                printf ( "%d ", m ) ;
    }
    display2 ( int *n);
    {
                printf ("%d ", *n ) ;
    }
```

Output: The first function call = 3, 5, 7, 9
The second function call = 3, 5, 7, 9

Pointer In C

The **Pointer** in C is a variable that stores address of another variable. A pointer can also be used to refer to another pointer function. A pointer can be incremented/decremented, i.e., to point to the next/ previous memory location. The purpose of pointer is to save memory space and achieve faster execution time.

Pointer Syntax

pointer = &variable;

A simple program for pointer illustration is given below:

Example:

```
# include <stdio.h>
int main ( )
{
  int a = 10;
  int *p;
  p = &a;
  printf (“ Address stored in a variable p is: %xn\n”, p);  //   accessing the address
  printf (“Value stored in a variable p is: %d \n , *p );   // accessing the value
  return0;
}
```

Output:
```
Address stored in a variable p is:60ff08
Value stored in a variable p is:10
```

Correspondence Between Each Topic and Related VTPs

27. Pointer Usage In Memory Allocation For Different Data Types

If origin address is increasing new address, int and float types will be increased **four** bytes and char type will be increased **one** bytes.

Operator

* - Declaration of a pointer
   - Returns the value of the referenced variable

& - Returns the address of a variable

%x is used for outputting integer in hexadecimal

Example:

```
int main ( )
{
        int i = 2, *x ;
        float j = 3.5, *y;
        char k = 'c', *z ;

        printf ( "\nValue of i = %d", i );
        printf ( "\nValue of j = %f", j );
        printf  ( "\nValue of k = %c",k );

        x = &i ; // Suppose the address of i is 65524
        y = &j ; // Suppose the address of j is 65520
        z = &k ; // Suppose the address of k is 65519

        printf ( "\nOriginal address in x = %u", x ) ;
        printf ( "\nOriginal address in x = %u", y ) ;
        printf ( "\nOriginal address in x = %u", z ) ;

        x++ ;
        y++ ;
        z++ ;
        printf ( "\nNew address in x = %u", x ) ;
        printf ( "\nNew address in y = %u", y ) ;
        printf ( "\nNew address in z = %u", z ) ;
}
```

Output:
```
Value of i = 2
Value of j (two decimal place) = 3.50
Value of k = c
Original address in x = 65524
Original address in y = 65520
Original address in z = 65519
```

Correspondence Between Each Topic and Related VTPs

```
New address in x = 6228
New address in y = 65524
New address in z = 65520
```

28. Array Elements In Different Ways

```
main ( )
{
        int num[ ] = { 14, 44} ;

      for (int i= 0 ; i <= 1; i++ )
       {
        printf ( "\naddress = %u   ", &num[i] ) ;
        printf ( "element = %d %d ", num[i], *( num + i ) ) ;//suppose the address starts 65535
        printf ( "%d %d", *( i + num ), i[num] ) ;
       }
}
```

Output:
```
Address  - 65535        element - 14
Address  - 65539        element - 44
```

29. Pointer Notation To Access 2-D Array Elements

Example:

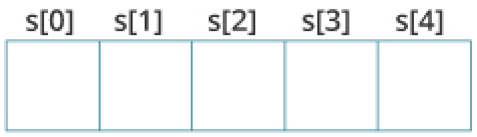
```
main ( )
{
        int s[3][2] = {
                        { 1234 , 65 },
                        { 4567 , 34 },
                        { 9875 , 90 },
                    } ;
        for (int i = 0 ; i <= 3 ; i++ )
    {
                printf ( "\n" ) ;
                for (int j = 0 ; j <= 1 ; j++ )
                  printf ( "%d ", *( *( s + i ) + j ) ) ;
     }
  }
```

Output:
```
1234,   65
4567,   34
9875,   90
```

30. To Declare A String

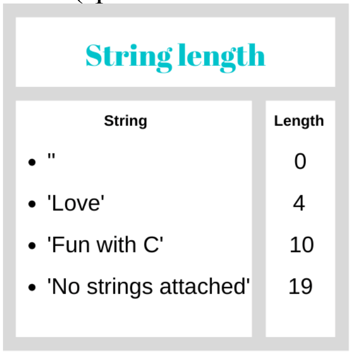
char s[5];

Correspondence Between Each Topic and Related VTPs



A string is a sequence of characters terminated with a null character \0.

C program to find length of a string, for example, the length of the string "C programming" is 13 (space character is counted). The null character isn't counted when calculating it.



String length

String	Length
• ''	0
• 'Love'	4
• 'Fun with C'	10
• 'No strings attached'	19

Finding String Length

```
main ( )
{
            char arr[   ] = "Myanmar" ;
            int len1, len2 ;

            len1 = strlen ( arr ) ;
            len2 = strlen (" Warmly Welcome " ) ;

            printf ( "\nstring = %s Length = %d", arr, len1 ) ;
            printf ( "\nstring = %s Length %d", "Humpty Dumpty", len2 );
}
```

Output:
```
Myanmar        -  7
Warmly Welcome - 14
```

31. Copying String From Source To Destination

```
main ( )
{
            char source[ ]  = "Magalarba" ;
            char target[10] ;

            strcpy ( target, source ) ;
```

Correspondence Between Each Topic and Related VTPs

```
                printf ( "\ n  source string = %s", source ) ;
                printf ( "\ n  target string = %s",  target);
    }
```

Output:
```
source string = Magalarba
target string  = Magalarba
```

32. String Concatenation In C

The concatenation of strings is a process of combining two strings to form a single string. If there are two strings, then the second string is added at the end of the first string.

For example, Hello + javaTpoint = HellojavaTpoint

Concatenation String from Source to Destination

```
    main ( )
    {
                char source[  ]  = "Konichiwa" ;
                char target[10]  =  " Hello!" ;

                strcat   ( target, source ) ;

                printf ( "\ n  source string = %s", source ) ;
                printf ( "\ n  target string = %s",  target);
    }
```

Output:
```
source string  = Konichiwa
target string  = Hello!Konichiwa
```

33. Exchange Names Using 2-D Array Of Characters

Example:

```
    main( )
    {
                char names [ ][10] = {
                                        "Myanmar",
                                        "English"
                                        "Math",
                                        "Physics",
                                        "Chemist"

                                    } ;
                char t ;
               printf ( "\nOriginal: %s %s", &names[4][0], &names[3][0] ) ;

                for (int i = 0 ; i <= 9 ; i++ )
                {
                          t = names[ 4 ][ i ]  ;
                          names[ 4 ][ i ] = names [ 3 ] [ i ] ;
                          names [ 3 ] [ i ] = t ;
                }
                printf ( "\nNew: %s %s", &names[2][0], &names[3][0] ) ;
    }
```

```
Output: Original: Chemist, Physics
          New:  Physics, Chemist
```

34. Structure

- User defined data types
- Using function we can define a data type which holds more than one element of different data types.

Accessing Structure Elements

```
 main ( )
{
         struct b
         {
            char name ;
            float number ;
            int notes;
        } ;
         struct book1 = { 'A', 123.00, 500 } ;

         printf ( "\nAddress of name = %c", book1.name ) ;
         printf ( "\nAddress  of number = %f", book1. number ) ;
         printf  "\nAddress of notes = %d",  book1. notes );
}
```

Output:
```
Address of name       = A
Address of number     = 123.00
Address of notes      = 50
```

35. Copying Structure Elements

```
main ( )
{
            struct book
            {
               char name  [8]
               int yr ;
               float value;
            };
struct book b1 = { "English", 20, 1500.50 } ;
struct book b2, b3 ;

strcpy ( b2.name, b1.name ) ; /* piece-meal copying */
b2.yr = b1.yr ;
b2.value = b1. value ;
b3 = b2 ; /* copying all elements at one go */

printf ( "\nbook1 = %s %d %f", b1.name, b1.yr, b1.value ) ;
printf ( "\nbook2 = %s %d %f", b2.name, b2.yr, b2.value ) ;
printf ( "\nbook3 = %s %d %f", b3.name, b3.yr, b3.value ) ;
}
```

Output:
```
book1 - English, 20, 1500.50
book2 - English, 20, 1500.50
book3 - English, 20, 1500.50
```

36. Nested Structures Usage

```
main ( )
{
            struct address
            {
                char phone[10] ;
                char city[20] ;
                int Number ;
            } ;
struct temp
            {
               char name[25] ;
               struct address b ;
            }  ;

struct temp x = { "Khangyi ", "202130", "romagyi ", 10 };
printf ( "\ nname = %s phone = %s", x.name, x.b.phone ) ;
printf  ( "\ncity = %s Number = %d", x.a.city, x.b.Number ) ;
```

Correspondence Between Each Topic and Related VTPs

```
    }
```

Output : name - Khangyi, phone -202130
city - romagyi , Number -10

37. Passing The Entire Structure Variable To The Method

```
struct main
{
        char type [15] ;
        char author[15] ;
        int number ;
 } ;
main ( )
  {
        struct book b1 = = { "C program ", "YZA", 306 } ;
        display ( b1 ) ;
  }
display ( struct book b )
  {
        printf ( "\nThe output is : %s %s %d", b.type, b.author, b.number ) ;
  }
```

Output: C program , YZA, 306

38. Usage Of A Structure Pointer

A pointer is a variable which points to the address of another variable of any data type like int, char, float etc. Similarly, we can have a pointer to structures, where a pointer variable can point to the address of a structure variable.

Example:

```
main ( )
{
        struct note
        {
            char name[25] ;
            char author[25] ;
            int callno ;
        };

        struct note N1 = { "Let us C", "AYD", 103 } ;
        struct note *ptr ;

        ptr = & N1;
```

```
                printf ( "\nThe output without using pointer is:%s %s %d", N1.name,      N1.author,
N1.callno ) ;
                printf ( "\nThe output with pointer is%s %s %d", ptr->name, ptr->author, ptr->callno ) ;
   }
```

Output:
```
The output without using pointer is  Let us C, AYD, 103
The output with pointer is   Let us C, AYD, 103
```