南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Advanced Programming

Lab 06

CONTENTS

- Master how to declare, define, and call a user-defined function

2 Knowledge Points

2.1 User-Defined Function

2.2 Recursive function

2.3 Pointers to functions

2.1 User-Defined Function

Syntax of defining a function:

function header

```
return_type function_name (datatype parameter1, datatype parameter2, …)
{
        // function body
}
```

- **return type:** suggests what type the function will return. It can be int,char, string, pointer or even a class object. If a function does not return anything, it is mentioned with void.
- **function name:** is the name of the function, using a legal identifier.
- **parameters:** are variables to hold values of arguments passed while function is called. A function may not contain parameter list, give **void** in the parentheses.

Function prototype:

The simplest way to get a prototype is to copy the **function header** and add a **semicolon**.

Example: Declaring, Defining and Calling a function

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

//function declaration(function prototype)
int sum(int x,int y);

int main()
{
    int a =10; int b=20; int c;
    //function call
    c = sum(a,b);
    cout << a << "+" << b << "=" << c << endl;
    return 0;
}

//function definition
int sum(int x,int y)
{
    int s = x + y;
    return s;
}
```

- Provide a function definition
- Provide a function prototype
- Call the function

Actual parameter and Formal parameter

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

//function declaration(function prototype)
int sum(int x,int y);

int main()
{
    int a =10; int b=2
    //function call
    c = sum(a,b);
    cout << "The Address of a is " << &a <<endl;
    cout << "The Address of b is " << &b <<endl;
    cout << a << "+" << b << "=" << c << endl;
    return 0;
}
//function definition
int sum(int x,int y)
{
    int s = x + y;
    cout << "The Address of x is " << &x <<endl;
    cout << "The Address of y is " << &y <<endl;
    return s;
}
```

Actual parameters(arguments)

When calling a function, the values of arguments are assigned to the parameters

Formal parameters

Process of the calling a function:

- The values of arguments are assigned to the those of parameters by the sequence of their definition from left to right one by one.
- The control flows into the function body and executes the statements inside the body.
- When it encounters the return statement, the control flow returns back to the calling function with a return value.

Result:

```
The Address of x is 0x7ffc4242276c
The Address of y is 0x7ffc42422768
The Address of a is 0x7ffc4242279c
The Address of b is 0x7ffc424227a0
10+20=30
```

Scope and duration of a variable

- An variable's **scope** is where the variable can be referenced in a program. Some identifiers can be referenced throughout a program, others from only portions of a program.
- A variable defined inside a function is referred to as a **local variable**. A **global variable** is defined outside functions.
- The **scope of a local variable** is from where it is defined to the end of the block which it is included or the end of the function.
- The **scope of a global variable** is from where it is defined to the end of the file(or the program).
- An variable's **storage duration** is the period during which that variable exists in memory.

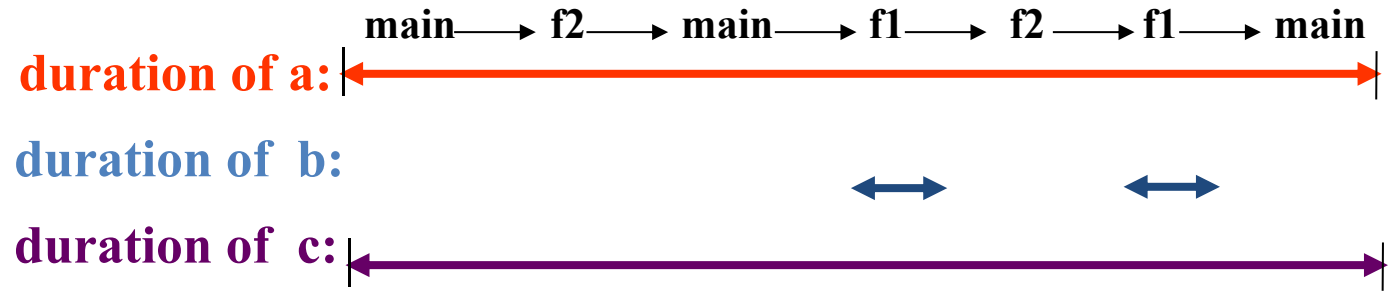
Scope and duration of a variable

```
int a;
void main( )
{    .......
     .......
       f2;
     .......
       f1;
     .......
}
f1( )
{   auto int b;
     ..........
      f2;
     ........
}
f2( )
{   static int c;
     ..........
}
```

scope of a

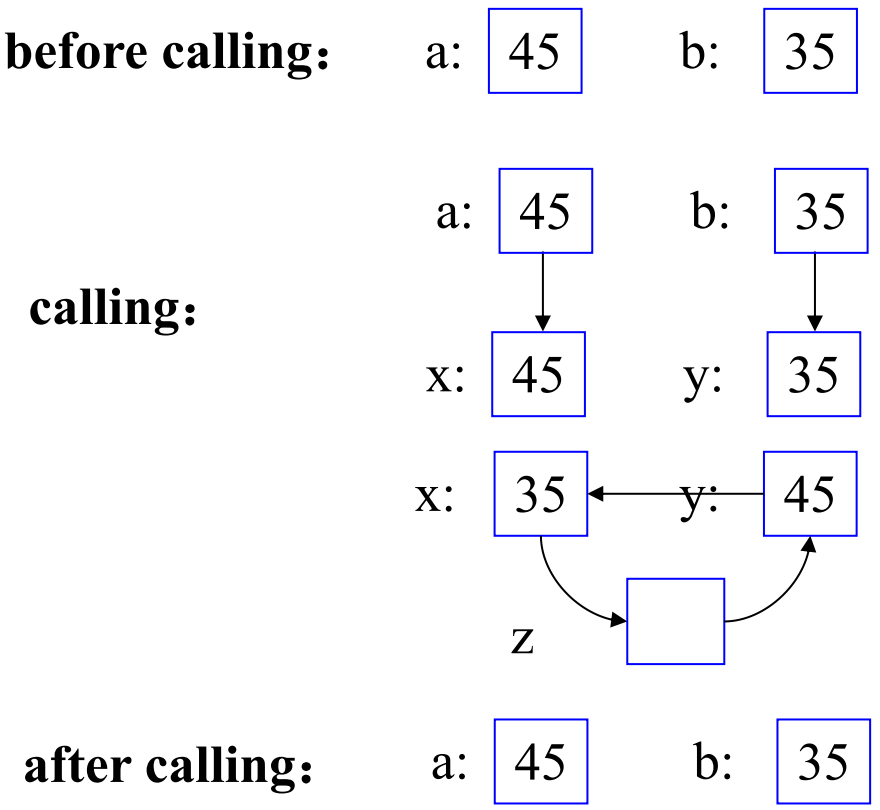
scope of b

scope of c



1. Passing arguments to a function by value

```
#include <iostream>
using namespace std;
void swap(int x,int y)
{
    int z;
    z = x;
    x = y;
    y = z;
}
int main()
{
    int a = 45,b = 35;
    cout << "Before swap:" << endl;
    cout << "a =" << a << ",b =" <<b << endl;

    swap(a,b);

    cout << "After swap:" <<endl;
    cout << "a =" << a << ",b =" << b << endl;
    return 0;
}
```

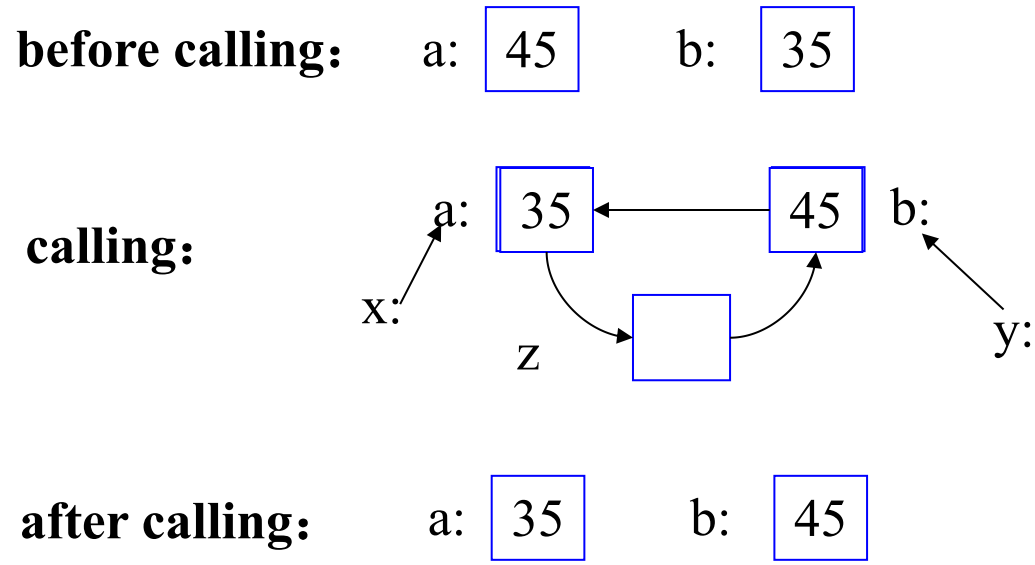


Result:

```
Before swap:
a =45,b =35
After swap:
a =45,b =35
```

2. Passing arguments to a function by pointer

```
#include <iostream>
using namespace std;
void swap(int *x,int *y)
{
    int z;
     z = *x;
    *x = *y;
    *y = z;
}
int main()
{
    int a = 45,b = 35;
    cout << "Before swap:" << endl;
    cout << "a = " << a << ",b = " << b << endl;

    swap(&a,&b);

    cout << "After swap:" <<endl;
    cout << "a = " << a << ",b = " << b << endl;

    return 0;
}
```



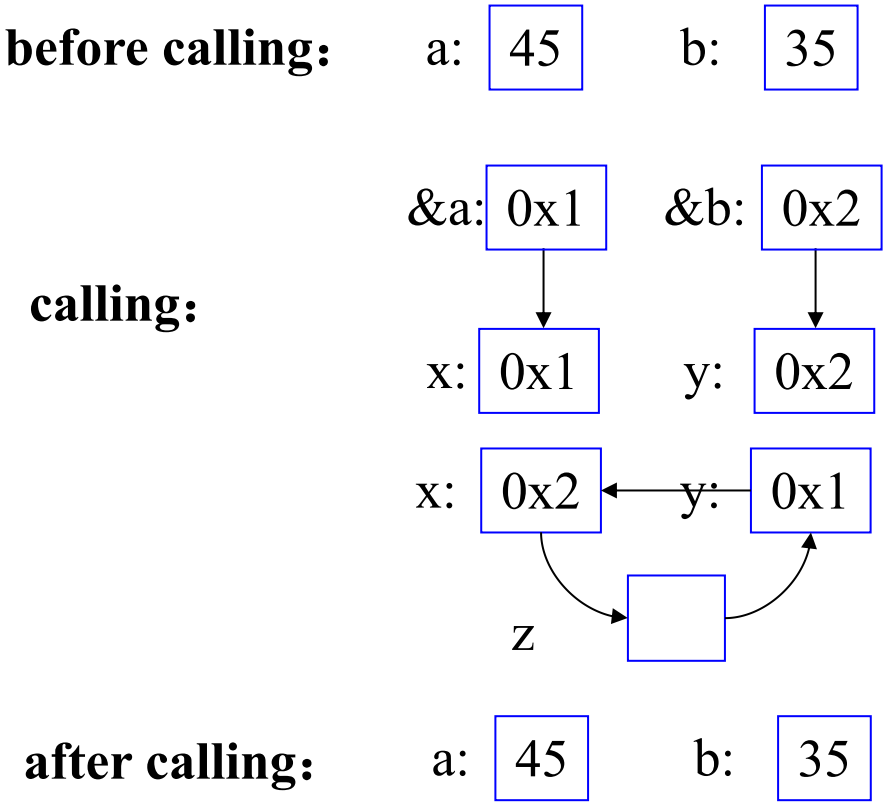
Result:

```
Before swap:
a = 45,b = 35
After swap:
a = 35,b = 45
```

Only Swap the Pointers

```
#include <iostream>
using namespace std;
void swap(int *x,int *y)
{
    int *z;
    z = x;
    x = y;
    y = z;
}
int main()
{
    int a = 45,b = 35;
    cout << "Before swap:" << endl;
    cout << "a = " << a << ",b = " << b << endl;

    swap(&a,&b);

    cout << "After swap:" <<endl;
    cout << "a = " << a << ",b = " << b << endl;
    return 0;
}
```

swap the pointers, does it work?



Result:

```
Before swap:
a = 45,b = 35
After swap:
a = 45,b = 35
```

3. Passing arrays to a function

(array as parameters and arguments)

```
#include <iostream>
#define SIZE 5
using namespace std;
int sumAl1Elements(int a[],int n);

int main()
{
    int arr[SIZE]={10,20,30,40,50};
    int total =sumAl1Elements(arr,SIZE);
    cout << "The sum of all elements is: "<< total << endl;
    return 0;
}

int sumAl1Elements(int a[],int n)
{
    int total =0;
    for(int i=0;i<n;i++)
        total +=a[i];
    return total;
}
```

Using array name as the argument

a = arr

Using array as a parameter

Result:

```
The sum of all elements is:150
```

3. Passing arrays to a function

(pointers as parameters and array name as arguments)

```
#include <iostream>
#define SIZE 5
using namespace std;
int sumAl1Elements(int a[],int n);

int main()
{
    int arr[SIZE]={10,20,30,40,50};
    int total =sumAl1Elements(arr,SIZE);
    cout << "The sum of all elements is: " << total << endl;
    return 0;
}

int sumAl1Elements(int *pa,int n)
{
    int total =0;
    for(int i=0;i<n;i++)
        //total +=*(pa+i);
        total+=pa[i];
    return total;
}
```

Using array name as the argument

p = arr; or

p = &arr[0];

Using pointer as a parameter

Result:

```
The sum of all elements is:150
```

3. Passing arrays to a function

(The values in an array can be modified inside the function body)

```
#include <iostream>
#define SIZE 5
using namespace std;
void sum(int* ,int* ,int);
int main()
{
    int a[SIZE]={10,20,30,40,50};
    int b[SIZE]={1,2,3,4,5};
    cout << "Before calling the function,the contents of a are: "  << endl;
    for (int i = 0; i < SIZE; i++)
    {
        cout << a[i] << " ";
    }
    sum(a,b,SIZE);
    cout << "\nAfter calling the function,the contents of a are: "  << endl;
    for (int i = 0; i < SIZE; i++)
    {
        cout << a[i] << " ";
    }
    cout << endl;
    return 0;
}
void sum(int *pa,int *pb,int n)
{
    for (int i = 0; i < n; i++)
    {
        *pa+=*pb;
        pa++;
        pb++;
    }
}
```

Modify the value which the pointer is pointed to

Result:

```
Before calling the function,the contents of a are:
10 20 30 40 50
After calling the function,the contents of a are:
11 22 33 44 55
```

The values of elements in array **a** are changed.

3. Passing arrays to a function

(protect the value of the argument from modifying, please use const)

```cpp
#include <iostream>
#define SIZE 5
using namespace std;
void sum(const int* ,const int* ,int);
int main()
{
    int a[SIZE]={10,20,30,40,50};
    int b[SIZE]={1,2,3,4,5};
    cout << "Before calling the function,the contents of a are: "  << endl;
    for (int i = 0; i < SIZE; i++)
    {
        cout << a[i] << " ";
    }
    sum(a,b,SIZE);
    cout << "\nAfter calling the function,the contents of a are: "  << endl;
    for (int i = 0; i < SIZE; i++)
    {
        cout << a[i] << " ";
    }
    cout << endl;
    return 0;
}
void sum(int *pa,int *pb,int n)
{
    for (int i = 0; i < n; i++)
    {
        *pa+=*pb;
        pa++;
        pb++;
    }
}
```

Use the **pointer-to-const** form to protect data!!

In definition, if the **const** is omitted, it will cause compiling error.

Error02:

```cpp
25
26  void sum(const int *pa,const int *pb,int n)
27  {
28      for (int i = 0; i < n; i++)
29      {
30          *pa+=*pb;
31          pa++;
32          pb++;
33      }
34
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS Filter (

_14_passingArrayPointer2.cpp 1

expression must be a modifiable lvalue C/C++(137) [Ln 30, Col 9]

Error:

```
cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week06$ g++ _14_passingArrayPointer2.cpp
/usr/bin/ld: /tmp/cclQFhff.o: in function `main':
_14_passingArrayPointer2.cpp:(.text+0xd6): undefined reference to `sum(int const*, int const*, int)'
collect2: error: ld returned 1 exit status
```

4. Passing multidimensional array to a function

Passing two-dimensional array as a parameter, the length of column can not be omitted.

```
#include <iostream>
#define SIZE 5
using namespace std;
void square(int arr[][3],int);
void square01(int (*arr)[3],int);
void square02(const int(*arr)[3],int n);
int main()
{
    int a[2][3]={
        {1,2,3},
        {4,5,6}
    };
    square02(a,2);
    return 0;
}
```

```
void square01(int (*arr)[3],int n)
{
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            //*(*(arr+i)+j) *= *(*(arr+i)+j);
            arr[i][j] *= arr[i][j];
            cout << *(*(arr+i)+j) <<" ";
        }
        cout << endl;
    }
}
```

```
void square(int arr[][3],int n)
{
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            arr[i][j] *= arr[i][j];
            cout << arr[i][j] <<" ";
        }
        cout << endl;
    }
}
```

If the values in the array can not be modified, use **const** in the prototype and definition. void square(**const** int arr[][3],int n);

```
void square02(const int(*arr)[3],int n)
{
    int temp;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            temp = *(*(arr+i)+j);
            cout << temp*temp << " ";
        }
        cout << endl;
    }
}
```

If the function definition is like this, can we invoke the function by two-dimensional array name?

```
64  void square03(const int **arr,int n)
65  {
66      cout << "square03 :" <<endl;
67      int temp;
68      for (int i = 0; i < n; i++)
69      {
70          for (int j = 0; j < 3; j++)
71          {
72              temp = *(*(arr+i)+j);
73              cout << temp*temp << " ";
74          }
75          cout << endl;
76      }
77  }
78
79
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS Filter (e.g. text, **

_16_passingTwoDArray.cpp 1

argument of type "int (*)[3]" is incompatible with parameter of type "const int **" C/C+

5. Passing C-style string to a function

```
#include <iostream>
#include <cstring>
using namespace std;
void mCopy(char *s,int m);
int main()
{
    char str[80];
    int m;
    cout <<"Enter a string:\n";
    cin.getline(str,80);
    cout <<"Enter the starting number you want to copy:\n";
    cin >> m;
    mCopy(str,m);
    cout <<"The copied string is:"<<str <<endl;
    return 0;
}

void mCopy(char *s,int m)
{
    strcpy(s,s+m-1);
}
```

You can use **character array** or **pointer-to-char** as a parameter.

Result:

```
Enter a string:
hello CS219
Enter the starting number you want to copy:
8
The copied string is:S219
```

6. Passing structure to a function

Passing structure to function by **value**

```
#include <iostream>
#include <string.h>
using namespace std;
struct student
{
int id;
char name[20];
float score;
};
void printStudent(student record);
int main()
{
    student record;
    record.id =1;
    stpcpy(record.name,"Raju");
    record.score =86.5;
    printStudent(record);
    return 0;
}
void printStudent(student st)
{
    cout << "Id is:" << st.id << endl;
    cout << "Name is:" << st.name << endl;
    cout << "Score is:" << st.score << endl;

}
```

Passing structure to function by **pointer**

```
#include <iostream>
#include <string.h>
using namespace std;
struct student
{
int id;
char name[20];
float score;
};
void printStudent(student *record);
int main()
{
    student record;
    record.id =1;
    stpcpy(record.name,"Raju");
    record.score =86.5;
    printStudent(&record);
    return 0;
}
void printStudent(student *st)
{
    cout << "Id is:" << st->id << endl;
    cout << "Name is:" << st->name << endl;
    cout << "Score is:" << st->score << endl;

}
```

Multiple files

```
//student1.h
#pragma once

struct student
{
    int id;
    char name[20];
    float score;
};

void printstudent(student *record);
```

```
// student_multifile.cpp
#include <cstring>
#include "student1.h"
int main()
{
    student record;

    record.id = 1;
    stpcpy(record.name, "Raju");
    record.score = 86.5;

    printstudent(&record);
    return 0;
}
```

Header file:

- const variable or macro definition
- structure declaration
- function prototype

When the preprocessor spots an **#include** directive, it looks for the following filename and includes the contents of that file within the current file.

```
//student.cpp
#include <iostream>
#include "student1.h"

void printstudent(student *st)
{
    std::cout << "Id is: " << st->id << std::endl;
    std::cout << "Name is: " << st->name << std::endl;
    std::cout << "Score is: " << st->score << std::endl;
}
```

look for file in standard system directories

look for file in your current directory first, and then in the standard system directories.

Result:

```
cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week06/_20$ g++ student.cpp student_multifile.cpp
cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week06/_20$ ./a.out
Id is: 1
Name is: Raju
Score is: 86.5
```

Multiple files

```
//student2.h
#ifndef STUDENT_H_
#define STUDENT_H_
struct student
{
    int id;
    char name[20];
    float score;
};
void printstudent(student *record);
#endif
```

Using conditional compilation directives to avoid duplicate including.

```
// student_multifile.cpp
#include <cstring>
#include "student2.h"
int main()
{
    student record;

    record.id = 1;
    stpcpy(record.name, "Raju");
    record.score = 86.5;

    printstudent(&record);
    return 0;
}
```

```
//student.cpp
#include <iostream>
#include "student2.h"

void printstudent(student *st)
{
    std::cout << "Id is: " << st->id << std::endl;
    std::cout << "Name is: " << st->name << std::endl;
    std::cout << "Score is: " << st->score << std::endl;
}
```

Result:

```
cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week06/_20$ g++ student.cpp student_multifile.cpp
-o student_program
cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week06/_20$ ./student_program
Id is: 1
Name is: Raju
Score is: 86.5
```

compile all the source files, with a given executable name

7. Return an array (or a pointer) from a function

```
#include <iostream>
#define SIZE 5
using namespace std;
int *fun();

int main()
{
    int *ptr = fun();
    for(int i = 0; i < SIZE; i++)
        cout << ptr[i] << " ";
    return 0;
}
int *fun()
{
    int arr[SIZE];
    // Some operation on arr
    for(int i = 0; i < SIZE; i++)
        arr[i] = (i+1) * 10;

    return arr;
}
```

arr is a local variable

Return the address of a local variable is wrong.

Result:

```
cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week06$ g++ _21_pointArray.cpp
_21_pointArray.cpp: In function ‘int* fun()’:
_21_pointArray.cpp:23:12: warning: address of local variable ‘arr’ returned [-Wre
turn-local-addr]
   23 |     return arr;
      |            ^~~
_21_pointArray.cpp:17:9: note: declared here
   17 |     int arr[SIZE];
      |         ^~~
cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week06$ ./a.out
Segmentation fault
cs@DESKTOP-L61ETB1:/mnt/h/CS219_2024F/code/week06$
```

A warning is caused when compiling.

The program can not be executed.

Three correct ways of returning an array (or a pointer):

- Return a static array
- Return a dynamically allocated array (or a pointer)
- Return a parameter pointer

```
#include <iostream>
#define SIZE 5
using namespace std;

int *fun()
{
    static int arr[SIZE];
    // Some operation on arr
    for(int i = 0; i < SIZE; i++)
        arr[i] = (i+1) * 10;

    return arr;
}

int main()
{
    int *ptr = fun();
    for(int i = 0; i < SIZE; i++)
        cout << ptr[i] << " ";
    return 0;
}
```

arr is a dynamically allocated array

return the dynamically allocated array **arr**

release the memory in caller

```
#include <iostream>
#define SIZE 5
using namespace std;

int *fun()
{
    int *arr=new int[SIZE];

    for(int i = 0; i < SIZE; i++)
        arr[i] = (i+1) * 10;

    return arr;
}

int main()
{
    int *ptr = fun();
    for(int i = 0; i < SIZE; i++)
        cout << ptr[i] << " ";

    delete[] ptr;

    return 0;
}
```

Return a parameter pointer

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

const char *match(const char *s, char ch)
{
    while(*s != '\0')
    {
        if(*s == ch)
            return s;
        else
            s++;
    }
    return NULL;
}
```

You can return the parameter pointer

```
int main()
{
    char ch, str[81];
    const char *p = NULL;

    cout << "Please input a string:\n";
    cin.getline(str, 81);
    cout << "Please input a character:\n";
    ch = getchar();

    if((p = match(str, ch)) != NULL)
    {
        cout << ch << " is in the string." << endl;
        cout << "The rest of string is: " << p << endl;
    }
    else
        cout << ch << " is not in the string." << endl;

    return 0;
}
```

Result:

```
Please input a string:
hello world
Please input a character:
e
e is in the string.
The rest of string is: ello world
```

Result:

```
Please input a string:
hello world
Please input a character:
a
a is not in the string.
```

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

void mcopy(char *s, int m);

int main() {
    char str[81];
    int m;
    cout << "Enter a string:\n";
    cin.getline(str, 81);
    cout << "Enter the starting number you want
to copy:\n";
    cin >> m;
    mcopy(str, m);
    cout << "The copied string is:
" << str << endl;
    return 0;
}

void mcopy(char *s, int m) {
    strcpy(s, s + m - 1);
}
```

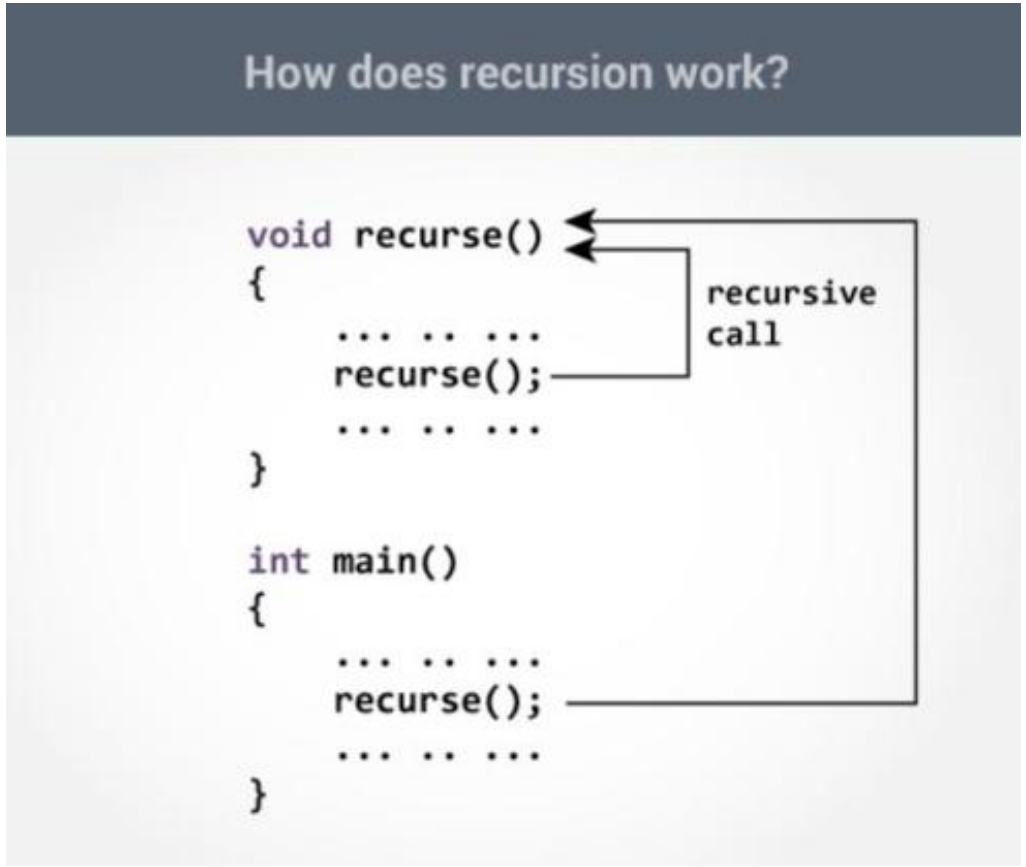
Modify the contents of the array,
need not return value.

```cpp
#include <iostream>
#include <cstring>
using namespace std;
const char *mpos(const char *s, int m);

int main() {
    char str[81];
    const char *p = NULL;
    int m;
    cout << "Enter a string:\n";
    cin.getline(str, 81);

    cout << "Enter the starting number you want to copy:\n";
    cin >> m;
    if ((p = mpos(str, m)) != NULL) {
        cout << "The original string is: " << str << endl;
        cout << "The copied string is: " << p << endl;
    } else {
        cout << m << " is illegal." << endl;
    }
    return 0;
}
const char *mpos(const char *s, int m) {
    if (m <= 0 || m > strlen(s)) {
        return NULL;
    }
    return (s + m - 1);
}
```

Donot modify the contents of the array,
return the proper position(pointer).

2.2 Recursive function

A function that **calls itself** is known as **recursive function**. And, this technique is known as **recursion**.



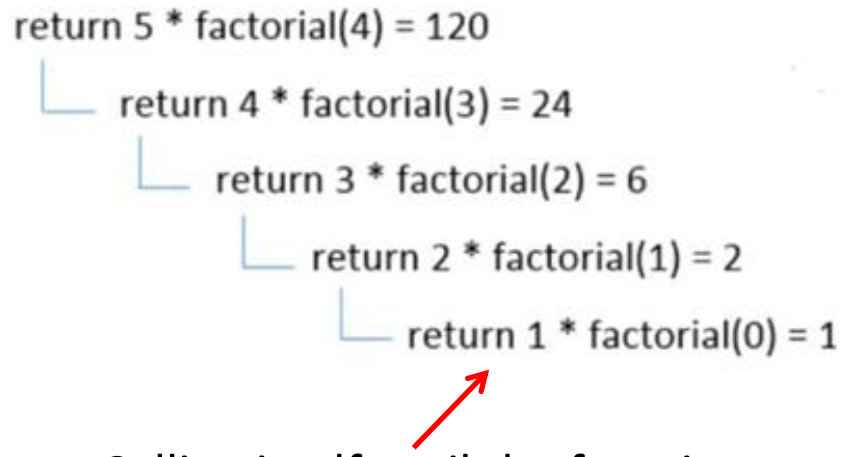
Recursion is used to solve various mathematical problems by dividing it into smaller problems.

Example: compute factorial with recursive function

Compute factorial of a number Factorial of n = 1*2*3…*n

```
#include <iostream>
using namespace std;
long factorial(int n);

int main() {
    long fact;
    int num;

    while (true) {
        cout << "Enter a positive integer: ";
        cin >> num;
        if (num < 0)
            cout << "Please input a positive integer!\n";
        else
            break;
    }
    fact = factorial(num);
    cout << "Factorial of "<<num <<" is: " << fact << endl;
    return 0;
}

long factorial(int n) {
    if (n == 0 || n == 1)
        return 1;
    return n * factorial(n - 1);
}
```

- Factorial function: f(n) = n*f(n-1),
- base condition: if n<=1 then f(n) = 1



Calling itself until the function reaches to the **base condition**!

Result:

```
Enter a positive integer: -9
Please input a positive integer!
Enter a positive integer: 9
Factorial of 9 is: 362880
```

Disadvantages of recursion:

- **Recursive programs are generally slower than nonrecursive programs**. Because it needs to make a function call so the program must save all its current state and retrieve them again later. This consumes more time making recursive programs slower.

- **Recursive programs requires more memory to hold intermediate states in a stack**. Non recursive programs don’t have any intermediate states, hence they don’t require any extra memory.

2.4 Pointers to Functions(Function Pointer)

Declare a pointer to a function:

return_type (*pointername)(parameter lists);

Return type of a function

The address of a function will be stored in the pointer, which indicates that the pointer is pointed to a function. Note () can not be omitted.

Parameters of a function

Example:

```
int findmax(int, int);
int   (*funptr)(int,int);
funptr = findmax;
int max = funptr(3,5);
```

Declaring a function

Declaring a pointer to a function

Assigning the address of a function to the pointer

Calling the function by the pointer

Example:

Compute the definite integral, suppose calculate the following definite integrals

$$\int_a^b f(x)dx=(b-a)/2*(f(a)+f(b))$$

$$\int_0^1 x^2\,dx \qquad \int_1^2 \sin x/x\,dx$$

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

double calc(double (*funp)(double), double a, double b);
double f1(double x);
double f2(double x);

int main() {
    double result;
    double (*funp)(double);

    result = calc(f1, 0.0, 1.0);
    cout << "1: result = " << result << endl;

    funp = f2;
    result = calc(funp, 1.0, 2.0);
    cout << "2: result = " << result << endl;

    return 0;
}
```

function pointer as a parameter

Declaring a function pointer

Calling the function by function name

Assigning the address of function f2 to the pointer

Calling the function by function pointer

$$\int_a^b f(x)dx=(b-a)/2*(f(a)+f(b))$$

```
double calc(double (*funp)(double), double a, double b)
{
    double z;
    z = (b - a) / 2 * ((*funp)(a) + (*funp)(b));
    return z;
}

double f1(double x)
{
    return x * x;
}

double f2(double x)
{
    return sin(x) / x;
}
```

$$\int_0^1 x^2 dx$$

$$\int_1^2 \sin x/x \, dx$$

Result:

```
1: result= 0.5
2: result= 0.64806
```

qsort() in general utilities library stdlib.h

The quick sort method is one of the most effective sorting algorithms. **qsort()** function sorts an array of data object.

```
void qsort(void *base, size_t nmemb, size_t size, int(*compar)(const void *, const void *));
```

void *base: pointer to the beginning of the array to be sorted, it permits any data pointer type to be typecast to a pointer-to-void.

size_t nmemb: number of items to be sorted.

size_t size: the size of the data object, for example, if you want to sort an array of double, you would sizeof(double).

int (*compar)(const void *, const void *): a pointer to a function that returns an **int** and take two arguments, each of which is a pointer to type const void. These two pointers point to the items being compared.

```cpp
#include <iostream>
#include <stdlib.h>

#define NUM 10

void fillArray(double ar[], int n);
void showArray(const double ar[], int n);
int myComp(const void *p1, const void *p2);

int main()
{
    double vals[NUM];
    fillArray(vals, NUM);
    std::cout << "Random list:\n";
    showArray(vals, NUM);

    qsort(vals, NUM, sizeof(double), myComp);

    std::cout << "\nSorted list:" << std::endl;
    showArray(vals, NUM);
    return 0;
}
```

```cpp
void fillArray(double ar[], int n)
{
    for (int i = 0; i < n; i++)
        ar[i] = (double)rand() / ((double)RAND_MAX + 1);
}
void showArray(const double ar[], int n)
{
    for (int i = 0; i < n; i++)
        std::cout << ar[i] << " ";
    std::cout << std::endl;
}
int myComp(const void *p1, const void *p2)
{
    const double *pd1 = (const double *)p1;
    const double *pd2 = (const double *)p2;
    if (*pd1 < *pd2)
        return -1;
    else if (*pd1 > *pd2)
        return 1;
    else
        return 0;
}
```

Sorting rule

Result:

```
Random list:
0.840188 0.394383 0.783099 0.79844 0.911647 0.197551 0.335223 0.76823 0.277775 0.55397

Sorted list:
0.197551 0.277775 0.335223 0.394383 0.55397 0.76823 0.783099 0.79844 0.840188 0.911647
```

Structure sort

```cpp
#include <iostream>
#include <cstring>
#include <cstdlib>  // For qsort
using namespace std;
#define SIZE 5
void display(const student *s, int n);
int myComp(const void *p1, const void *p2);
struct student {
    char name[20];
    int age;
};
int main() {
    student stu[SIZE] = {{"Alice", 19}, {"Bob", 20},
     {"Alice", 16}, {"Leo", 20}, {"Billy", 19}};
    cout << "Original students:\n";
    display(stu, SIZE);
    qsort(stu, SIZE, sizeof(student), myComp);
    cout << "\nSorted students:" << endl;
    display(stu, SIZE);
    return 0;
}
void display(const student *s, int n) {
    for (int i = 0; i < n; i++) {
        cout << "Name: " << s[i].name << ", age: " << s[i].age << endl;
    }
}
```

Result:

```
Original students:
Name: Alice, age: 19
Name: Bob, age: 20
Name: Alice, age: 16
Name: Leo, age: 20
Name: Billy, age: 19

Sorted students:
Name: Alice, age: 16
Name: Alice, age: 19
Name: Billy, age: 19
Name: Bob, age: 20
Name: Leo, age: 20
```

```
int myComp(const void *p1, const void *p2)
{
    const student *ps1 = (const student *)p1;
    const student *ps2 = (const student *)p2;

    int res = strcmp(ps1->name, ps2->name);
    if (res != 0) {
        return res;
    } else {
        if (ps1->age < ps2->age) {
            return -1;
        } else if (ps1->age > ps2->age) {
            return 1;
        } else {
            return 0;
        }
    }
}
```

If the name is the same, sort by age

3 Exercises

1. Write a program that will display the calculator menu. The program will prompt the user to choose the operation choice(from 1 to 5). Then it will ask user to input two integer values for the calculation. See the sample below.

```
Calculator Menu:
1. Add
2. Subtract
3. Multiply
4. Divide
5. Modulus
Choose operation (1-5): 1
Enter first number: 3
Enter second number: 4
Result: 7
Press Y/y to continue, N/n to exit: y
Choose operation (1-5): 2
Enter first number: 5
Enter second number: 6
Result: -1
Press Y/y to continue, N/n to exit: y
Choose operation (1-5): 4
Enter first number: 30
Enter second number: 5
Result: 6
Press Y/y to continue, N/n to exit: n
Done!
```

The program also asks the user to decide whether he/she wants to continue the operation. If he/she inputs ‘y’(‘Y’), the program will prompt the user to choose the operation gain. Otherwise, the program will show “Done” and terminate.

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

void display()
{
    // complete code here
}

int Add(int a, int b)
{
    // complete code here
}

int Subtract(int a, int b)
{
    // complete code here
}

int Multiply(int a, int b)
{
    // complete code here;
}

int Divide(int a, int b)
{
    // complete code here}

int Modulus(int a, int b)
{
    // complete code here
}
```

```cpp
int main()
{
    int choice;
    int num1, num2;
    char continueChoice;
    display();

    do
    {
        cout << "Choose operation (1-5): ";
        cin >> choice;

        cout << "Enter first number: ";
        cin >> num1;
        cout << "Enter second number: ";
        cin >> num2;

        switch (choice)
        {
          // complete code here
        }
        cout << "Press Y/y to continue, N/n to exit: ";
        cin >> continueChoice;

    } while (continueChoice == 'Y' || continueChoice == 'y');

    cout << "Done!" << endl;

    return 0;
}
```

2. Write a program that uses the following functions:

- **int fill_array(double arr[], int size)** prompts the user to enter double values to the array. It ceases taking input when the array is full or when the user enters non-numeric input, and it returns the actual number of entries.
- **void show_array(double *arr, int size)** displays the contents of the array.
- **void reverse_array(double *arr, int size)** is a **recursive function**, it reverses the values stored in the array.

The program should use these functions to fill an array, show the array, reverse the array. Hint: use the dynamic array to store the data.

```
Enter the size of an array: 6
Enter value #1:1
Enter value #2:2
Enter value #3:3
Enter value #4:4
Enter value #5:5
Enter value #6:6
The original array is: 1 2 3 4 5 6
The reversed array is: 6 5 4 3 2 1
```

```
Enter the size of an array: 6
Enter value #1:12
Enter value #2:13
Enter value #3:14
Enter value #4:a
The original array is: 12 13 14
The reversed array is: 14 13 12
```