7.3 Passing Arrays as Parameters

7.3.1Passing The Elements of Arrays as Parameters

[ex.7.10] count letters in a string. (design a function to judge a letter

```
int  isalp(char c)
     {
        if  (c>='a'&&c<='z'||c>='A'&&c<='Z')
             return(1);
        else  return(0);
     }
```

```
main()
  {        int i,num=0;
           char str[255];
           printf("Input  a  string: ");
           gets(str);
           for(i=0;str[i]!='\0';i++)
                if (isalp(str[i]))    num++;

           printf("num=%d\n",num);
           getch();
     }
```

7.3.2 Passing Arrays as Parameters

Pass the whole array as parameter

[ex.7.11] calculate average

```
float aver(float a[5])
  {   int i;
      float av,s=0;
      for(i=0;  i<5;  i++)  s +=  a[i];
      av=s/5;
      return av;
   }
```

```
float aver(float a[ ])
{   int i;
      float av,s=0;
      for(i=0;  i<5;  i++)  s +=  a[i];
      av=s/5;
      return av;
   }
```

```
void main()
   { float score[5], av;
      int i;
      printf("\ninput 5 scores:\n");
      for(i=0;  i<5;  i++)  scanf("%f",&score[i]);
      av=aver(score);
      printf("average score is %5.2f\n",av);
      getch();
   }
```

Transfer address

Explain：

（1）With the function parameters of array masterpieces, we should define the array separately in the keynote and the modulated function, and the data type must be identical, otherwise the result will be wrong.

For example, in this case, the parameter group is a[], the real argument group is sco[], all are float types.

（2）C compilation system does not check the size of the formal parameter array, so the formal parameter array is not required to specify the size.

For example, the parameter group in this case can be defined as float a[].

（3）Array is delivered as an address.

[ex.7.12] bubble sort

```
#define N 10
main()
{  void bubble(float b[ ]);
    float a[N+1], t;
    int i;
    for (i=1; i<=N; i++)       scanf("%f",&a[i]);
    bubble(a);
    printf("the sorted numbers:");
    for (i=1; i<=N; i++)      printf("%5.1f",a[i]);
    printf("\n");   getch();
}
```

```
void bubble(float b[ ])
{   int i,j;
     float t;
     for (i=1;  i<N; i++)
        for (j=1; j<=N-i; j++)
           if (b[j]<b[j+1])
                 {   t=b[j];
                     b[j]=b[j+1];
                     b[j+1]=t;
                 }
}
```

```
#define N 10
main()
{  void bubble(float b[ ]);
    float a[N+1],t;
    int i;

    for (i=1; i<=N; i++)
         scanf("%f",&a[i]);
    bubble(a);
    printf("the sorted numbers:");
    for (i=1; i<=N; i++)
          printf("%5.1f",a[i]);
    printf("\n");
    getch();
}
```

```
void bubble(float b[ ])
{   int i,j;
     float t;
     for (i=1;  i<N; i++)
        for (j=1; j<=N-i; j++)
             if (b[j]<b[j+1])
                {  t=b[j];
                   b[j]=b[j+1];
                   b[j+1]=t;}
                 }
}
```

7.4 The Scope of Variables

- The scope of variables: variables can be used in a scope which is decided by the position of variables' definition.

- Internal variables
- External variables

7.4.1 Local Variables

Local variables:

- **1. the definition of location: In a function (or compound statement) inside;**
- **2. Scope: Valid only within the scope of the function (or compound statement).**

Called "function scope" or "block scope"

- **Note: You can use a defined variable only within a function that contains a variable definition, and you cannot use these variables outside of this function. So the internal variable is also called "local variable".**

EXAMPLES

```
int f1(int a)
   {  int b,c;
        ……
   }             /*the scope of a,b,c,only in f1() */

int f2(int x)
   {  int y,z;
        ……
   }                /* the scope of x,y,z, only in f2() */
main()
   { int m,n;
        ……
   }             /* the scope of m,n, only in main() */
```

Supplemental description of the scope of local variables:

1．Local variables defined in main () are also used only in the main function and cannot be used by other functions. Also, internal variables defined in other functions cannot be used in the primary function. Because the main function is parallel to other functions. This is different from other languages and should be noted.

2. Formal parameters are also local variables, which belong to the modulated function;

An argument is generally an internal variable for a keynote function.

3．Allows the use of the same variable names in different functions, representing different objects, assigning different units, interfering with each other, and not confusing.

4．You can also define a variable in a compound statement that is scoped only within a compound statement.

An example for block scope：

```
main()
{  int i=0,s=0;
   {   int s=0, i;
       for( i=1;i<=10;i++)
           s+=i;
   }
   printf("%d,%d\n",i,s);
}
```

/* Declaration i,s has function scope * /

/* Declaration i,s with block Scope * /

/* Function scope i,s not visible */

/* Block scope termination */

/* Output function scope i,s*/

Output Result: 0,0

7.4.2 Global Variables

The adjective "external" is used in contrast to "internal," which describes the arguments and variables defined inside functions.

Global variables are defined outside of any function, and are thus potentially available to many functions.

- Scope: start at defined location, end of this file.

  Called "File Scope"

- An global variable does not belong to any function and can be referenced directly by all functions in the scope, and therefore called global variables.

Description of global variables:

(1) External variables strengthen the data connection between modules, but they are prone to side effects.

Makes the function less independent.

From the point of view of modular programming, it should be avoided.

(2) The same source program file allows local and global variables to have the same name. The whole variable is masked (invisible) within the local variable scope.

(3) Global variables are not visible before the definition point, and extern declarations can be used to change their visibility.

General form: extern data type external variable table;

(4) The difference between the definition and declaration of an external variable: Definition of external variable: must be outside all functions and can only be defined once. Declaration of an external variable: multiple occurrences within a function that uses the external variable.

An example for external variables

```
#include <stdio.h>
char c='A';    int i;
void p1(int m,int n)
{  int j;
   for(i=1;i<=n;i++)
      {   for(j=m-i;j>0;j--)  putchar(' ');
          for(j=1;j<=2*i-1;j++) putchar(c++);
          putchar('\n');
      }
}
main()
{  int i;
   for(i=1;i<=3;i++)      p1(5,i);
   getch();
}
```

Global variables

c， i

Local variables

m， n， j

Local variable

I

In main function

7.5 The Storage Type of Variables

- Commonly form of variable definition

  [< storage type >] <data type> <variables' list>;

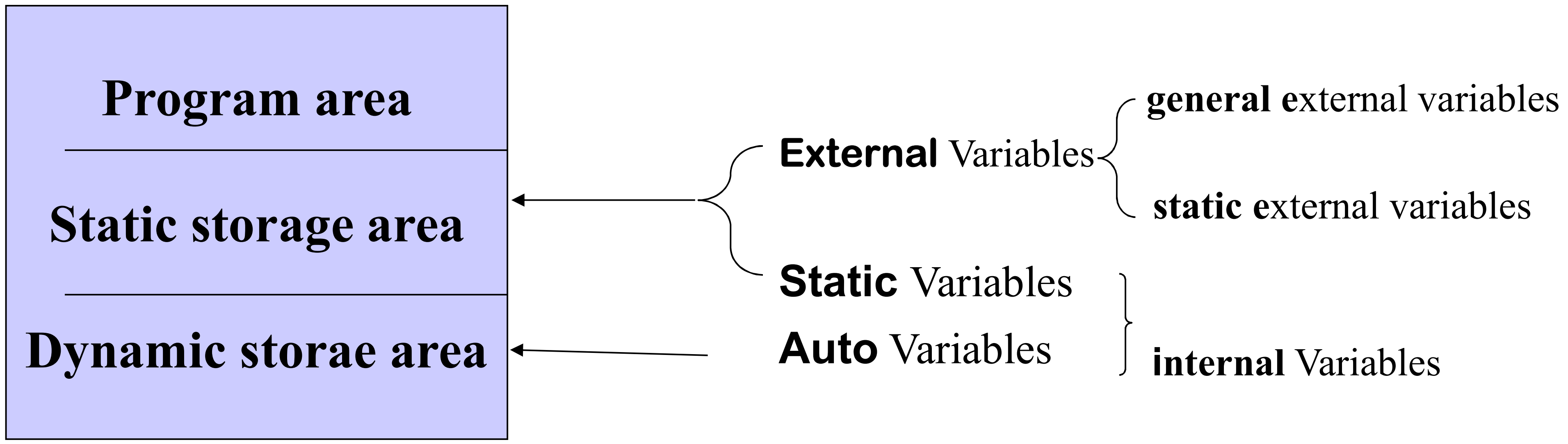
Lifetime of a variable The action type of the variable

- storage type

auto	automatic
register	register type
static	jingtaixing
extern	exernal reference type

- The memory space allocated to the user when the C program is running:

- User area



7.5.1 Storage of Internal Variables(内部变量的存储方式)

auto variables

•1, storage Features: dynamic storage, storage space coverage

An automatic variable defined in a function (compound statement) that allocates storage space when a function is called, and then releases it at the end of the call.

•2, the definition is not initialized, the value is indeterminate.

If initialized, the initial value operation is performed at the time of the call, and the initial value is assigned each time the call is made.

•3 , Because the scope and lifetime of an automatic variable are limited to the individual (function or compound statement) that defines it, a variable with the same name is allowed in different individuals without confusion.

Static Variables

1, Storage features: static storage.

The storage space is allocated before the program is executed, and the storage space is occupied for the whole process. Does not release even if the function call ends. It is not visible in other functions.

2, If defined but not initialized, it is automatically assigned to "0" (integer and solid) or ‘ \0 ' (character), and each time they are called with the function, no longer reset the initial value, just keep the last call at the end of values!

3, Scope of application

(1) You need to preserve the value at the end of the last call to the function; or

(2) a variable is referenced without changing its value.

```
int sum(int k)
{  int i;
   static int y =0 ;
   for(i=1;i<=k;i++)
      y+=i;
   return y;
}
```

/* Define static variable y, initial value is 0, No initial value is assigned when you call sum again 0*/

```
main()
{  int m,n;
   printf("m,n=");
   scanf("%d%d",&m,&n);
   printf("%d\n",sum(m)+sum(n));
   getch();
}
```

Result

```
m,n=  4   5↙
35
```

[Example 7.] The storage characteristics of automatic variables and static local variables.

```
void  auto_static(void)
 {  int var_auto=0;           /* Auto Variable: Reinitialize every call.
    static int var_static=0; /* Static local variable: Initialize only 1 times.
    printf(“var_auto=%d, var_static=%d\n”,
                var_auto, var_static);
      ++var_auto;
      ++var_static;
    }

main()
    {  int i;
        for(i=0; i<5; i++) auto_static( );
    }
```

Optput results：

```
var_auto=0, var_static=0
var_auto=0, var_static=1
var_auto=0, var_static=2
var_auto=0, var_static=3
var_auto=0, var_static=4
```

Register Variables

- **1, storage features: stored in the CPU registers in order to improve the running speed.**

  Reflects the low level of C language, and hardware closely related.
- **2, the use of a limited number.**

7.5.2 Storage of Global Variables

•Global variable storage features: static storage

1，General global variables----allow functions to be referenced in other source files

To refer to a function in another source file, make an extern declaration in the source file where the calling function is located to extend the scope of the global variable:

extern data type global variable list;

2，Static global variables----only allowed by function references in source files. Definition format:

static data type global variable list;

3, Pay attention：

An extern declaration within a function that refers to a global variable in the source file!

An extern declaration outside a function that refers to a global variable in another file.

Scope, lifetime?

The difference between a static local variable and a static external variable:

Both belong to the static storage mode

(1)The location of the definition is different.

(2)The scope of the function is different.

A static local variable is an internal variable whose scope is limited to the function in which it is defined; Although the lifetime is the entire source program, other functions cannot use it.

A static external variable is defined outside of a function and scoped to the source file in which it is defined; The lifetime is the entire source program, but a function in another source file cannot use it.

(3) initialization is different.

A static local variable, which is initialized only on the 1th call to its function, is no longer initialized when the function that defines it is called again, but instead retains the value at the end of the 1 call. Static external variables are defined outside the function, there is no "repeat" initialization problem for static internal variables, and the current value is determined by the operation assigned to it in the last 1 times.

7.6 Local Functions and External Functions

When a source program is composed of multiple source files, C functions can be divided into local functions and external functions depending on whether the function can be called by functions in other source files.

- 7.6.1 Local functions (Static functions)
- 7.6.2 External functions

Local Functions(Static Functions)

1, internal function: refers to the scope of the function is limited to this file.

2,define the format：

static function type function name (function argument list)

{ ……

}

The meaning of "static" here does not mean storage, but the scope of the function is limited to this file.

3, the advantage of using internal functions is: different people write different functions, do not worry about their own definition of functions and other files in the same name.

7.6.2 External functions

1, the definition of external functions:

When you define a function, you do not add static, or add extern.

```
[extern]  function type  function name (function argument list)
                 {   ……
                  }
```

2, external function calls:

In the file that calls the external function, use the extern declaration first, then call again.

```
extern  function prototype；
```

[Example] external functions use

（1） File mainf.c

```
main()
{  extern void input(…)；
   extern void process(……)；
   extern void output(…);
    … …
    input(…);
    process(…);
    output(…);
}
```

（2）File subf1.c

```
  ……
  extern void input(……)        /* define external functions */
   {……}
```

（3） File subf2.c

```
   ……
  extern void process(……)     /* define external functions */
  {……}
```

（4） File subf3.c

```
    ……
   extern void output(……)      /*define external functions*/
   {……}
```