C Programming

(Basics, Pointers, Functions, Arrays, Structures)

PROF. NAVRATI SAXENA

Introduction

Dennis Ritchie

Brian Kernighan

- **1972**: **Dennis Ritchie** at Bell Labs wrote C
- **1978**: Publication of "**The C Programming Language"** by Kernighan & Ritchie revolutionized the computing world

Why C?

- C produces code that runs nearly as fast as code written in assembly language
- Major examples of the use of C might be:
  1. Operating Systems of major languages
  2. Language Compilers
  3. Assemblers
  4. Computer Network Drivers
  5. Databases
  6. Language Interpreters
  7. Text Editors
  8. Print Spoolers
  9. Utilities

Tips for Learning of C!

- Resources
  - Book: **"The C Programming Language"** by Kerninghan and Ritchie
  - These lectures

- Learning any Programming Language
  - **Best way**: Start writing programs – make your hands dirty
  - Work your way through examples from lectures, KR, and/or any additional tutorials

C: Writing and Running Programs

1. Write text of program (source code) using an editor such as emacs, save as file e.g. my_program.c

2. Use compiler to convert program from source to an “executable” or “binary”:

3 ~ N. Compiler gives errors and warnings; edit source file, fix it, and re-compile

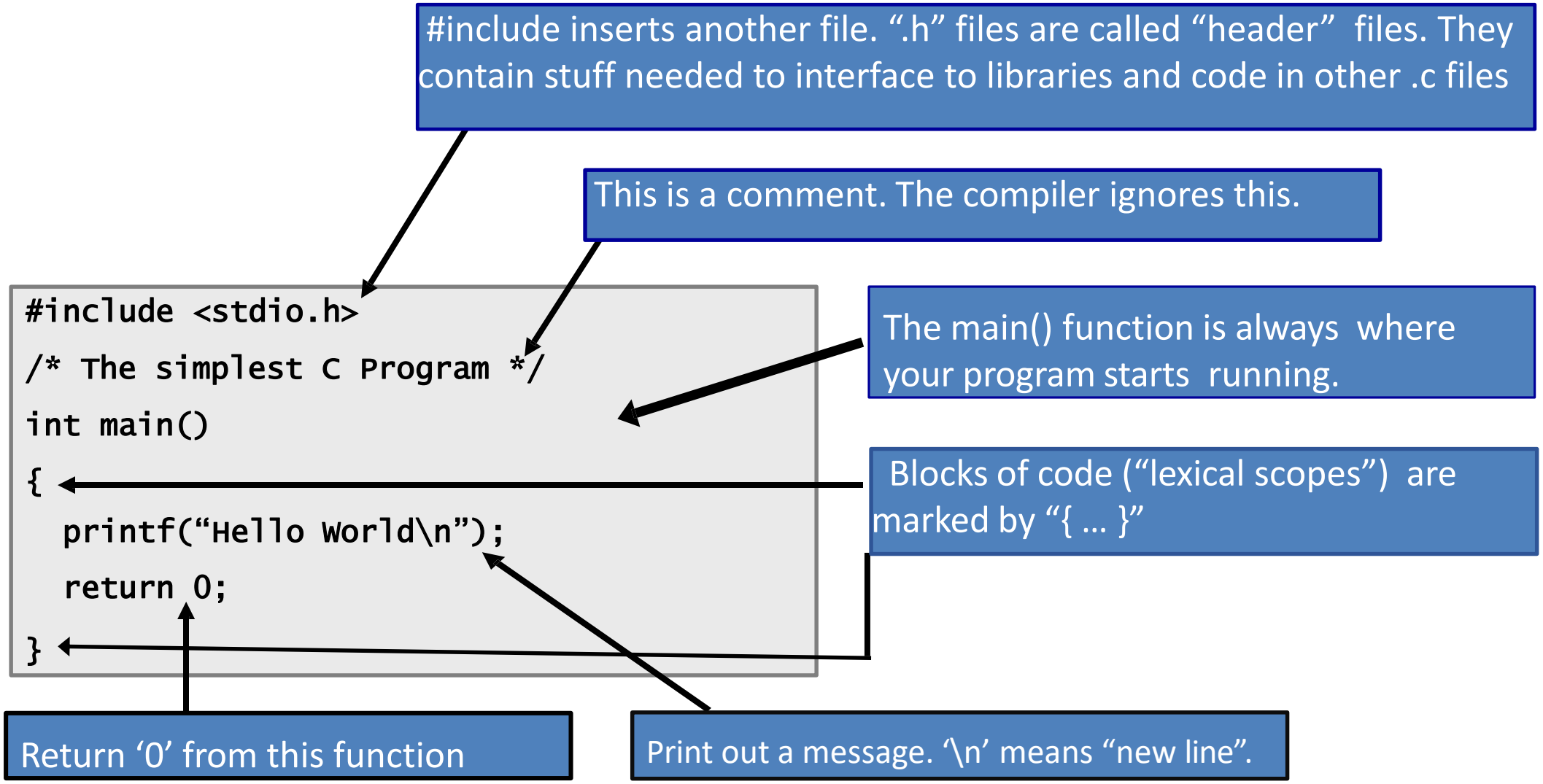
N. Run it and check if it works

Check any logical error

C: Fundamentals

- Mostly hardware independent
- Programs portable to most computers
- Case-sensitive
- Run in MS Windows (Visual C), Unix and Linux (gcc, bcc)
- Four stages
  1. Editing: Writing the source code by using some IDE or editor
  2. Preprocessing or libraries: Already available routines
  3. Compiling: translates or converts source to object code
     - for a specific platform source code -> object code
  4. Linking: resolves external references and produces the
     - executable module

C Syntax with Example



C: Simple Variables (1/4)

- Consider the declaration in C ***int i = 3*** ;

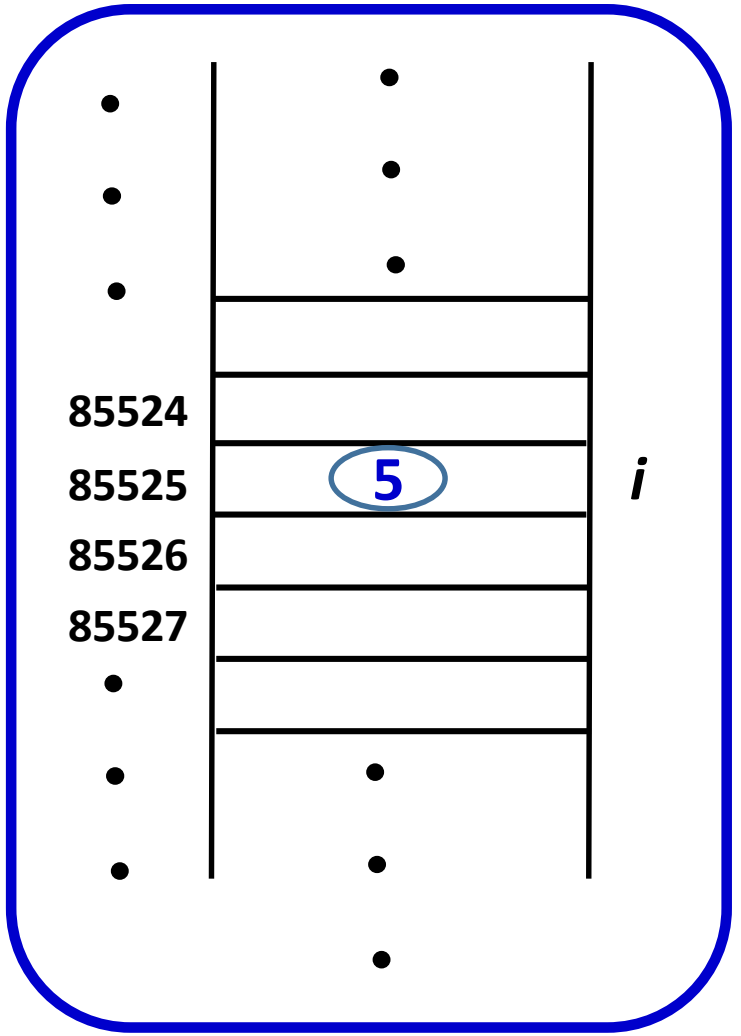
- This declaration tells the C compiler to:
  - Reserve space in memory to hold the integer value.
  - Associate the name ***i*** with this memory location.
  - Store the value ***3*** at this location.

C: Simple Variables (2/4)

Consider the declaration in C

int i = 5;

Name of a simple variable, which stores a single value

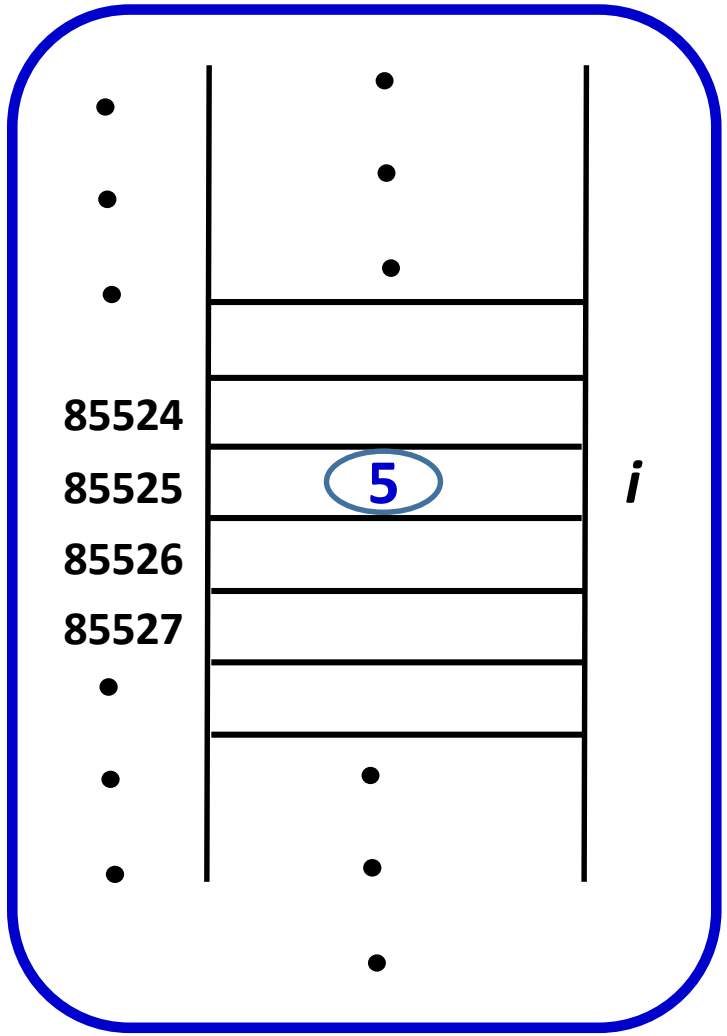


C: Simple Variables (3/4)

Operator '&'

```
main( )
  {
   int i = 5 ;
  printf ( "\nAddress of i = %u", &i ) ;
  printf ( "\nValue of i = %d", i ) ;
  }
```

Output?

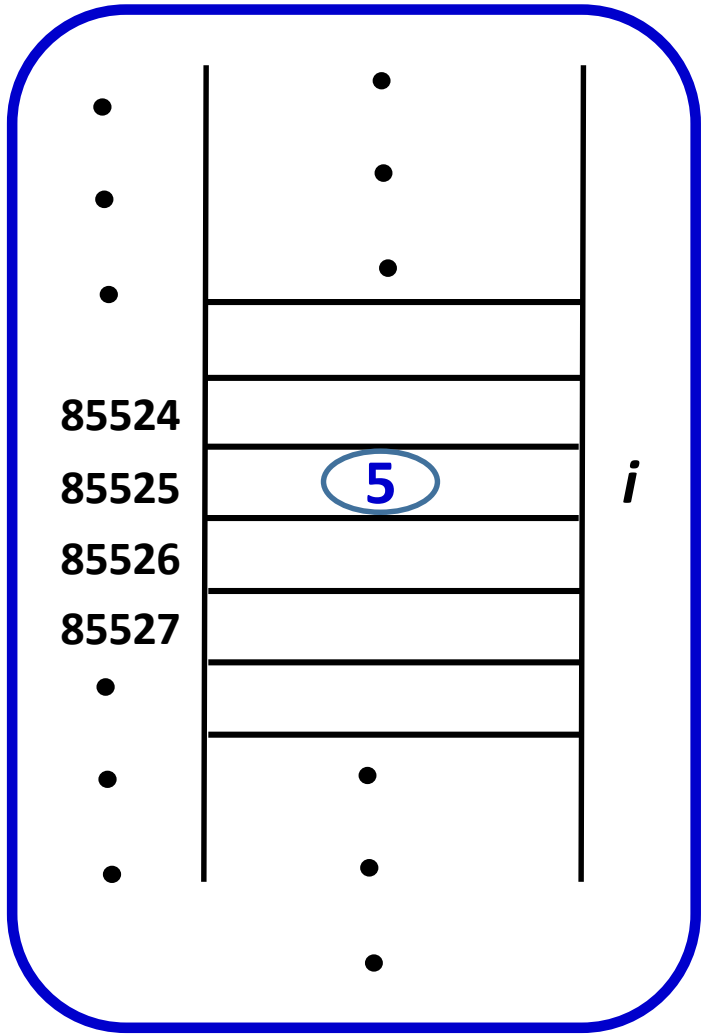


C: Simple Variables (4/4)

Operator '*'

```
main( )
 {
  int i = 5 ;
  printf ( "\nAddress of i = %u", &i) ;
  printf ( "\nValue of i = %d", i ) ;
  printf ( "\nValue of of i = %u", *(&i)) ;
 }
```

Output?



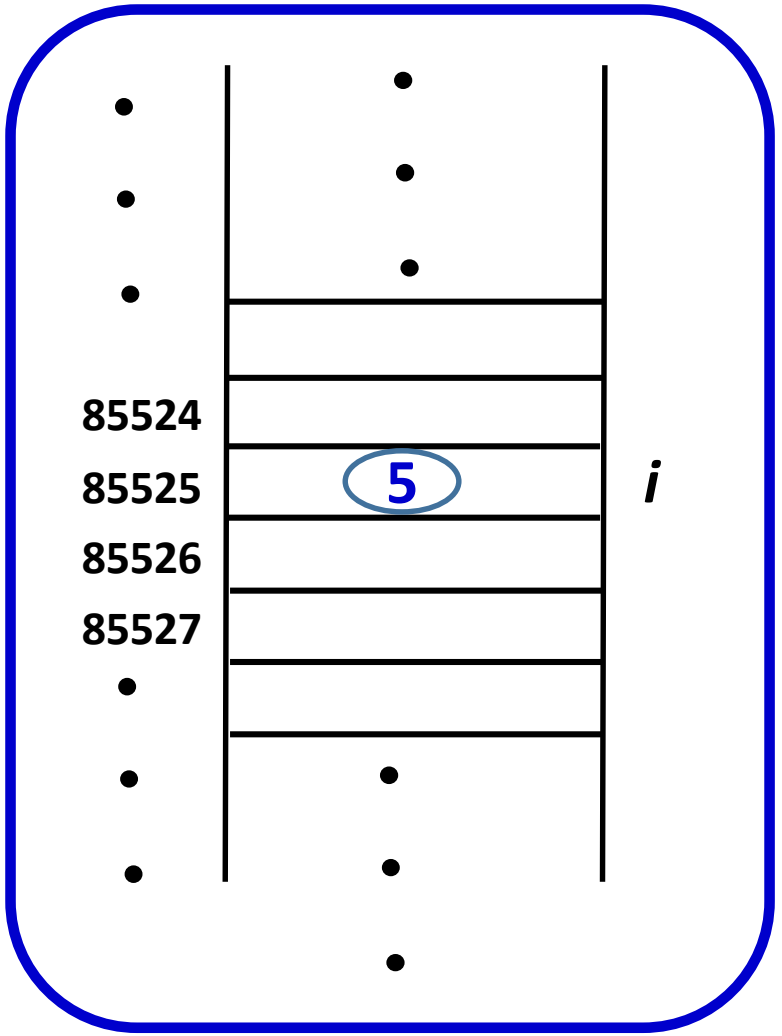
C: Pointer Variables (1/2)

Consider the declaration in C

Int *i = 5;

Name of another variable,
But this time it's a pointer variable

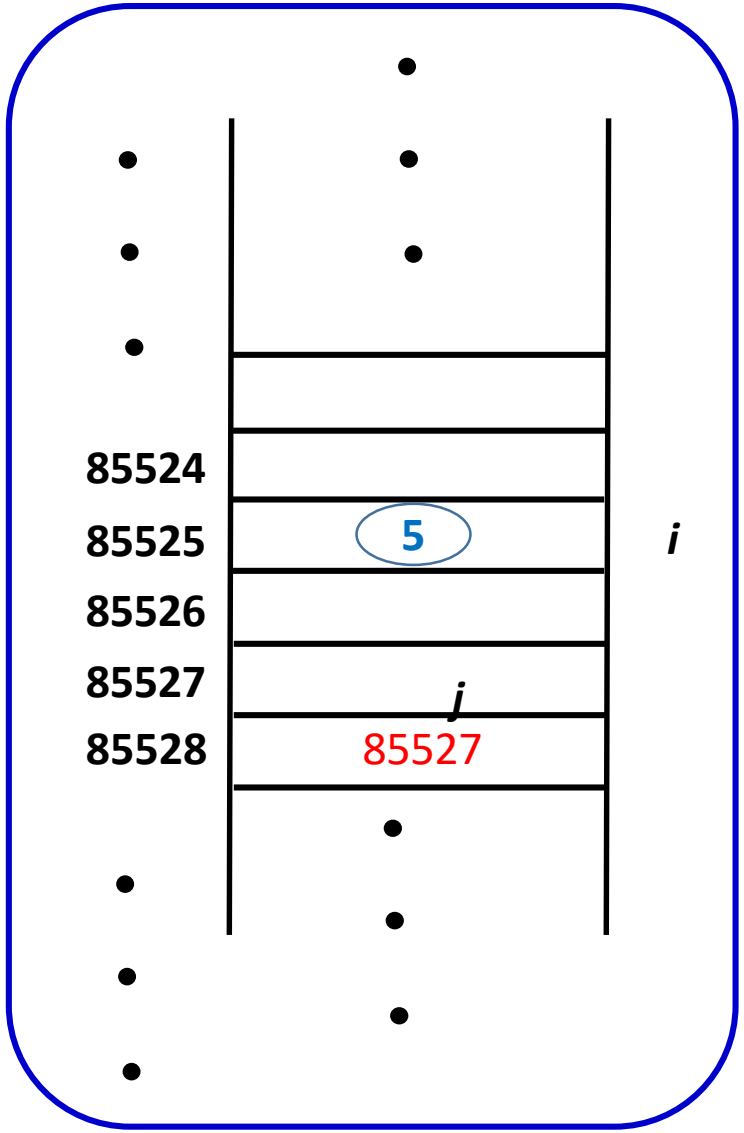
This variables will store address, containing the value, rather than the value itself



C: Pointer Variables (2/2)

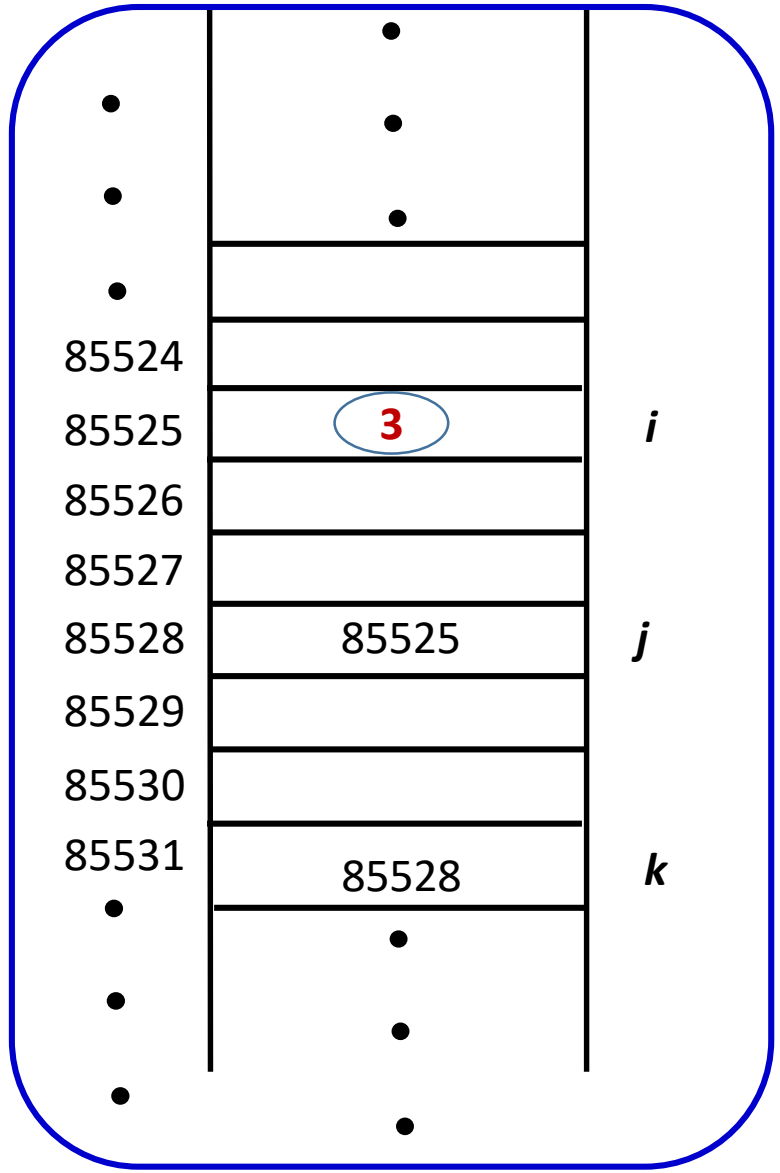
Consider the following C code fragment

```
main( )
 {
   int i = 5 ;
   int *j ;
   j = & i;

        printf ( "\nAddress of i = %u", &i ) ;
        printf ( "\nAddress of i = %u", j ) ;
        printf ( "\nAddress of j = %u", &j ) ;
        printf ( "\nValue of j = %u", j ) ;
        printf ( "\nValue of i = %d", i ) ;
        printf ( "\nValue of i = %d", *( &i ) ) ;
        printf ( "\nValue of i = %d", *j ) ;
  }
```



Pointers to a Pointers!

```
main( )
{
 int i = 3, *j, **k ;
 j = &i ;
 k = &j ;
     printf ( "\nAddress of i = %u", *k ) ;
     printf ( "\nAddress of j = %u", &j ) ;
     printf ( "\nAddress of j = %u", k ) ;
     printf ( "\nAddress of k = %u", &k ) ;
     printf ( "\nValue of j = %u", j ) ;
     printf ( "\nValue of k = %u", k ) ;
     printf ( "\nValue of i = %d", i ) ;
     printf ( "\nValue of i = %d", * ( &i ) ) ;
     printf ( "\nValue of i = %d", *j ) ;
     printf ( "\nValue of i = %d", **k ) ;
}
```



Pointer Arithmetic (1/4)

1. Addition of a number to a pointer. For example

```
int i = 4, *j, *k ;
    j = &i ;
    j = j + 1 ;
    j = j + 9 ;
    k = j + 3 ;
```

2. Subtraction of a number from a pointer. For example

```
int i = 4, *j, *k ;
    j = &i ;
    j = j - 2 ;
    j = j - 5 ;
    k = j - 6 ;
```

Pointer Arithmetic (2/4)

3. Subtraction of one pointer from another

```
main( )
  {
    int arr[ ] = { 10, 20, 30, 45, 67, 56, 74 } ;
    int *i, *j ;

    i =&arr[1];
    j = &arr[5];
    printf ( "%d %d", j - i, *j - *i ) ;
  }
```

Pointer Arithmetic (3/4)

4. Comparison of two pointer variables

```
main( )
  {
   int arr[ ] = { 10, 20, 36, 72, 45, 36 } ;
   int *j, *k ;

   j = &arr [ 4 ] ;
   k = ( arr + 4 ) ;

   if ( j == k )
        printf ( "The two pointers point to the same location" ) ;
   else
        printf ( "The two pointers do not point to the same location" ) ;
  }
```

Pointer Arithmetic (4/4)

Do not attempt following operations on pointers !

- **Addition of two pointers**
- **Multiplication of a pointer with a constant**
- **Division of a pointer with a constant**

The “while” Loop

Problem: Repeat the same instructions a fixed number of times

=> Looping

Solution: “while” loop.

```
loop:
   if (condition)
   {
     statements;
     goto loop;
   }
```

```
while (condition)
{
   statements;
}
```

```
float pow(float x, int exp)
{
   int i=0;
   float result=1.0;
   while (i < exp)
   {
      result = result * x;  i++;
   }
   return result;
}

int main()
{
   float p;
   p = pow(10.0, 5);
   printf(“p = %f\n”, p);  return 0;
}
```

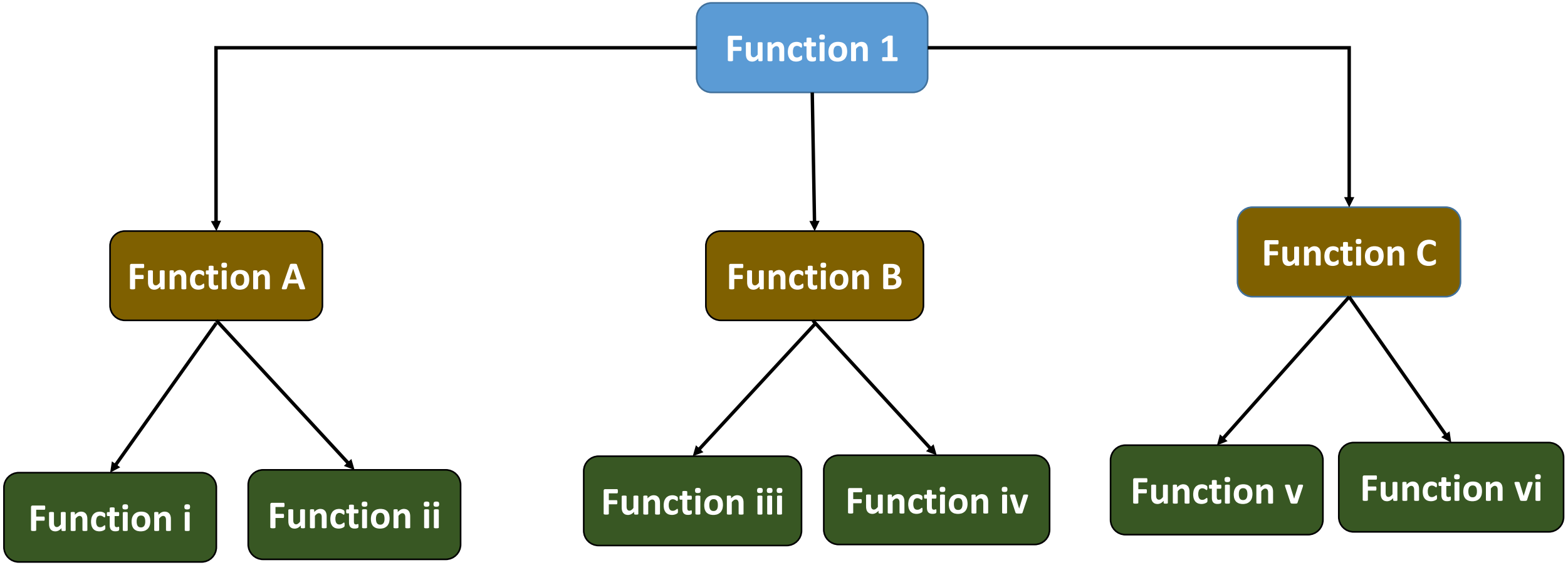
The “for” Loop

```
float pow(float x, int exp)
{
  float result=1.0;  int i;
  i=0;
  while (i < exp)
  {
  result = result * x;
  i++;
  }
  return result;
}

int main()
{
  float p;
  p = pow(10.0, 5);
  printf(“p = %f\n”, p);  return 0;
}
```

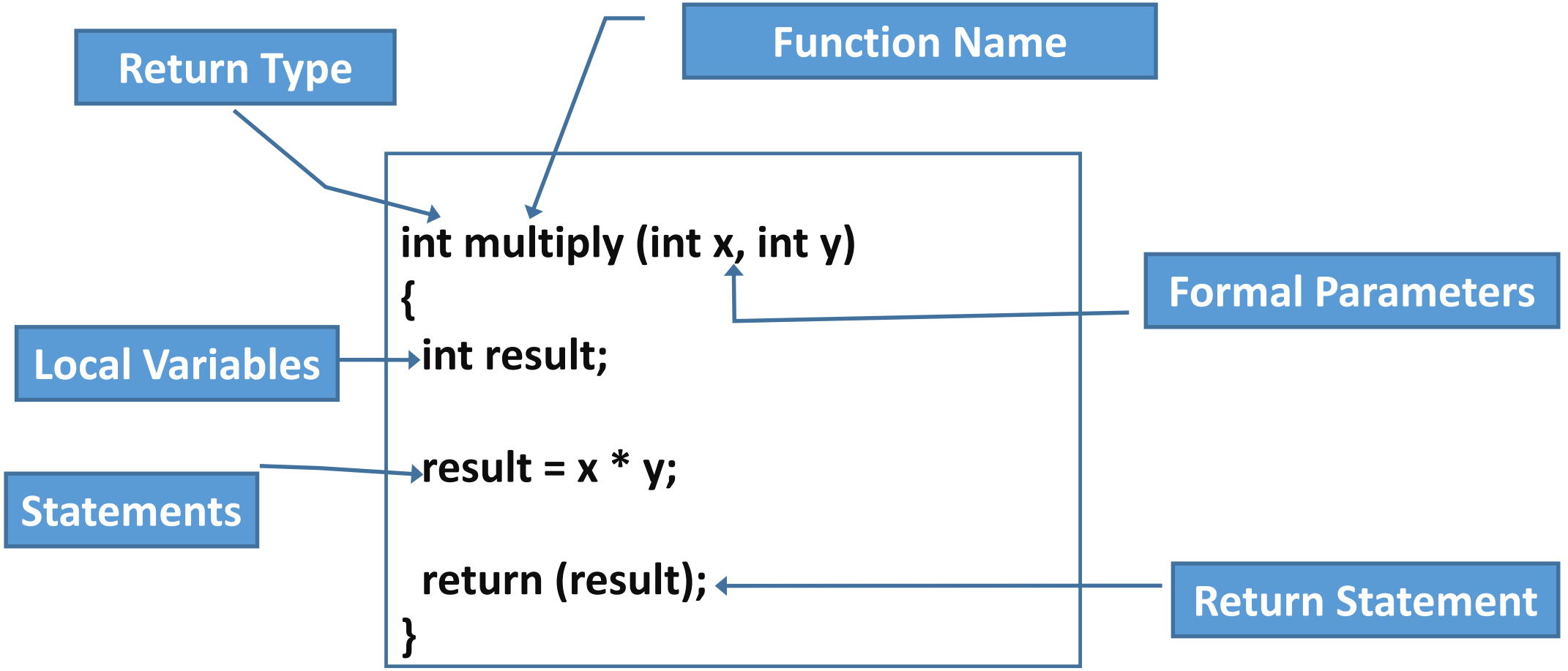
```
float pow(float x, int exp)
{
  float result=1.0;  int i;

  {
   for (i=0; i< exp; i++)
      result = result * x;
    }
  return result;
}


int main()
{
  float p;
  p = pow(10.0, 5);
  printf(“p = %f\n”, p);  return
  0;
}
```

Functions and Arrays

Functions


Function 1
Function A
Function B
Function C
Function i
Function ii
Function iii
Function iv
Function v
Function vi

Functions: Major Components (1/3)



Functions: Major Components (2/3)

Function Name
- Unique name to identify the function

Return Type
- Type of Value returned by the function
- Could be void, i.e. no value

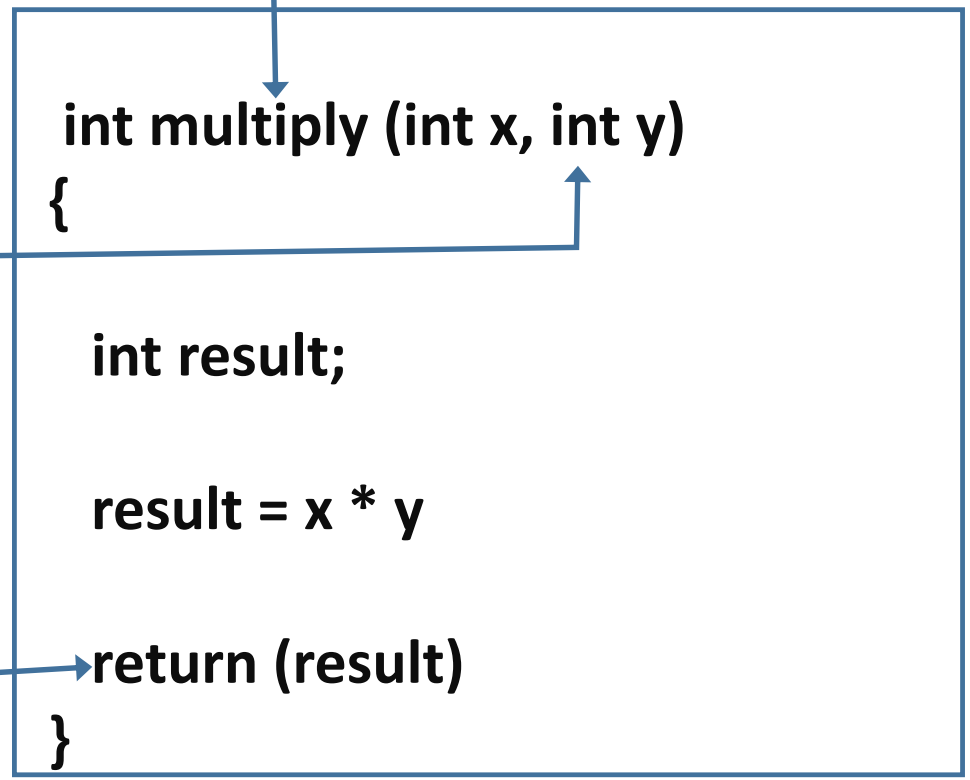
```
int multiply (int x, int y)
{
   int result;

   result = x * y

   return (result)
}
```

Functions: Major Components (3/3)

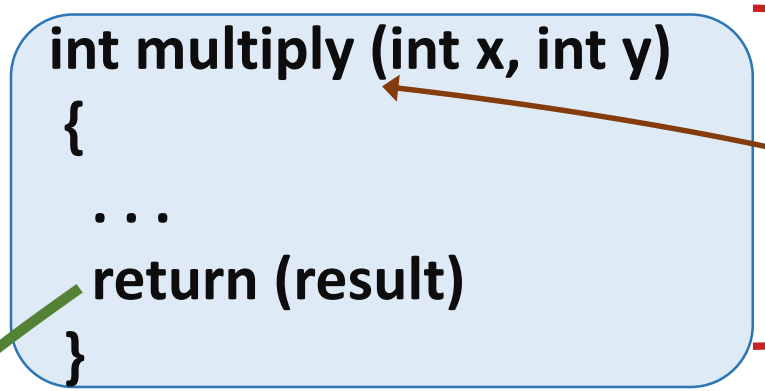
Function Name
- Unique name to identify the function

Function Parameters
- List of types and names, separated by ","
- External values for the function to work on
- Optional field, could be omitted if not needed

Return Value
- Value returned by the function in the end
- No return value, if function's return type is void



Functions: Calling and Returning



2. Defining a function

```
void main ()
{
  int number1, number2, mult;
  int multiply(int,int); /*1. declaring a function*/
  mult = multiply(number1, number2); /*3. Calling the function*/

}
```

Formal and Actual Parameters

- **Formal parameters**: variables declared in the header of the function definition
- **Actual parameters**: expressions in the calling statement
- Formal and actual parameters must match exactly in type, order, and number
- Their names, however, do not need to match

```
int multiply (int x, int y)
{
  . . .
  return (result)
}
```

```
void main ()
{
  int num1, num2;

  mult = multiply(num1, num2)

}
```

Functions: Parameter Passing (1/3)

In function calls, parameters are passed by

- Value (of the arguments)
- Reference (addresses of the arguments)

Functions: Parameter Passing (2/3)

Pass by value

```
main( )
  {
       int p = 40, q =80;
       swap ( p, q ) ;
       printf ( "\np = %d q = %d", p, q ) ;

  }
```

```
swap ( int x, int y )
  {
  int temp;

  temp = x ;
  x = y ;
  y = temp ;
  printf ( "\nx = %d y = %d", x, y ) ;
  }
```

Output?

Functions: Parameter Passing (3/3)

Pass by Reference

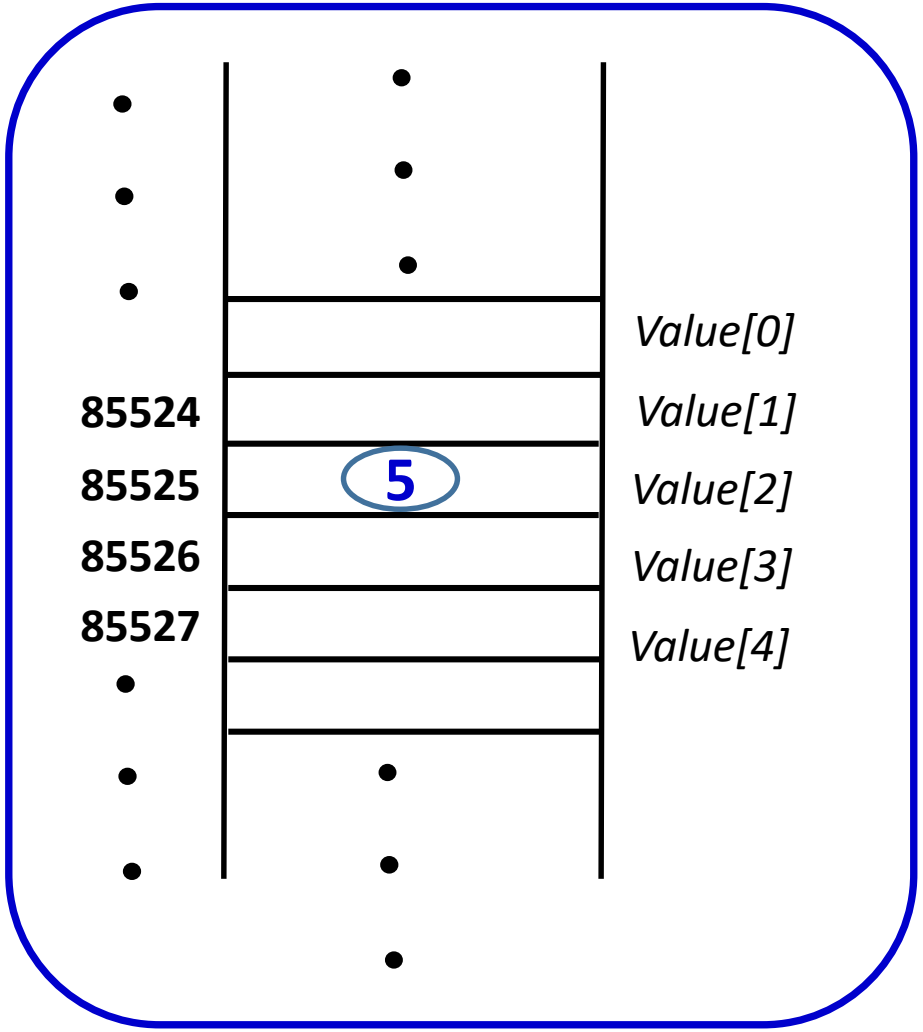
```
main( )
  {
      int p = 40, q =80;
      swap ( &p, &q ) ;
      printf ( "\np = %d q = %d", p, q ) ;

  }
```

```
swap ( int *x, int *y )
  {
  int temp;

  temp = *x;
  *x = *y ;
  *y = temp ;
  printf ( "\nx = %d y = %d", x, y ) ;
  }
```

Output?

Arrays (1/2)

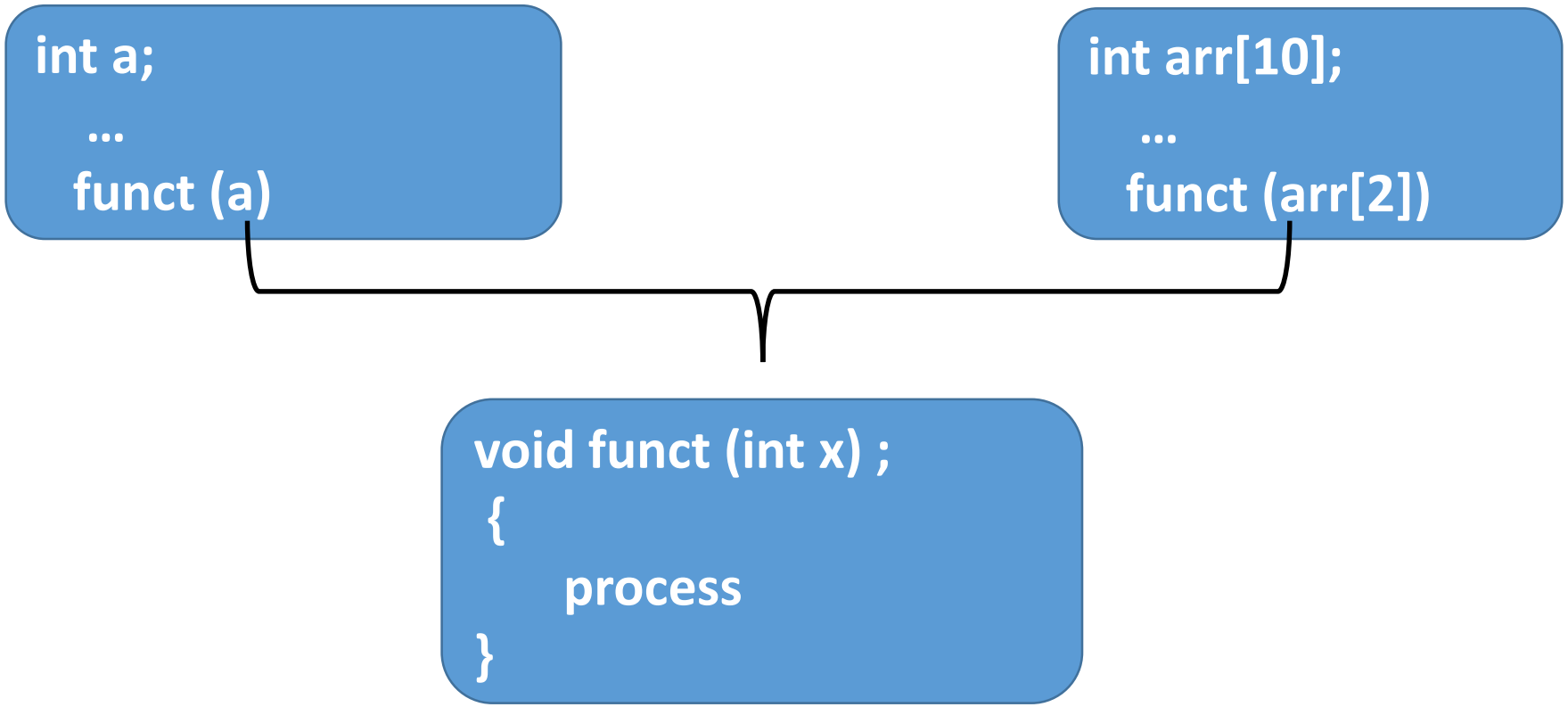
- A collection of similar elements stored in consecutive memory locations
- An array is known as a ‘subscripted variable’
- *int Value[5];*



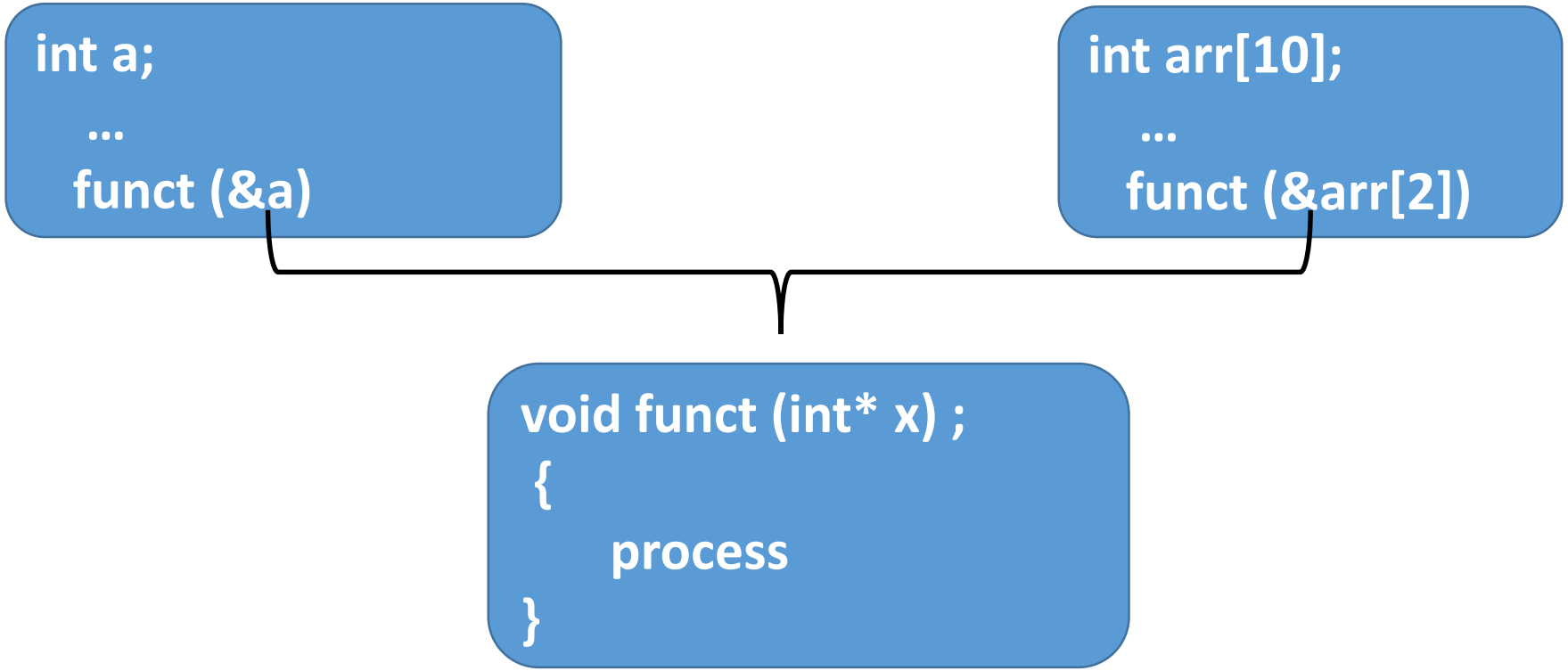
Arrays (2/2)

- **Functions manipulating arrays**
  - An array element can be passed as an argument/parameter to a function (as a primitive data type)
  - An array can also be returned from a function
  - An array of pointers

Passing Array: Passing a Single Element



Passing Array: Passing Address of an Element



Passing the Whole Array

```
int arr[10];
   …
  funct (arr)
```

```
void funct (int ar[ ]) ;
{
     process
}
```

Fixed Size

```
int arr[var_size];
   …
  funct (arr)
```

```
void funct (int ar[ * ]) ;
{
     process
}
```

Variable Size

Structures

A group of meaningful, related data to identify an entity

```
main()
 {
struct student{
      int roll_no[15];
      Char[] name;
      char *authors;
      double marks;
 }
  //declarations
  struct student student1, student2;
  }
```

```
main() {
  printf("Student1 roll-number is %d",
  student1.roll_no);  printf("Student1
  marks is %f", student1.marks);
}
```

Array of Structures: Example

```
#define MAX 100

Struct addr {
        char name[30];
        char street[40];
        char city[20];
        char state[3];
        int zip;
} addr_list[MAX];

//The functions to be used
//with the collection
int menu_select();
int find_free();
```

```
int menu_select() {
char s[80];
int c;
printf("\n\n 1. Enter a name\n");
printf(" 2. Delete a name\n");
printf(" 3. List the file\n");
printf(" 4. Quit\n");
do {
 printf("\nEnter your choice: );  gets(s);
 c = atoi(s);
} while(c<0 || c>4);
return c;
}
```

```
/* Find an unused structure. */
int find_free()
{
      int i;
      for(i=0; addr_list[i].name[0] && i<MAX; ++i) ;
      if(i==MAX) return -1;    /* no slots free */
      return i;
}
```