Computer Programming C++ (66111)

Instructors: Dr.Laui Malhis
Miss.Haya Sammaneh
Eng. Muhannad Al-Jabi
Eng.Anas Toameh

Arrays

- Consecutive group of memory locations.
- Same name and type.

To refer to an element in an array we must specify the array name and the position number of that element.

Example: ***name***[*position number*]

The positions of N-locations array are numbered from 0 to N-1

Array

- An array is a collection of data storage locations, each of which holds the same type of data. Each storage location is called an *element* of the array.
- Each element is referred to as an *indexed* variable.

Arrays declaration

When declaring arrays we must specify:-

- Array type.
- Array name.
- Array size.

Type ***name***[*array size*];

We can declare multible arrays of the same type by:

type name1[size1], name2[size2],…;

Array Declaration

- Syntax
  dataType arrayName [ConstIntExpression]
- The number of elements in an array is stated within square brackets, [ ].
- Examples

```
double angle [4];

constant POLY = 8;
double angle [POLY];

int testScore [12];
char password [8];
```

Arrays declaration

Also we can specify the size of the array as follows:-

```
Type name[]={element1,element2,….};
```

By this way the size of the array is specified by the number of elements in the list. We can see that we fill the array at declaration.

Using Array Elements

write the contents of an array element:

```
cout << angle[2];
```

assign values to an array element:

```
cin >> angle[3];
angle[6] = pow(3,4);
```

use it as a parameter:

```
y = sqrt(angle[0]);
```

use it in an expression:

```
x = 2.5 * angle[1] + 64;
```

Example

From the following array declaration, what will be the content of the array after the program has been compiled and executed?

```
int x[5]={1,2,3};
```

The content of x is 1,2,3,0,0.

Initialize an Array

```
double angle[4] = {16.21, 15.89, 7.5, -45.7};

double ATtoCG[5] = {.64, .89, .76, .83, .65};

int scores[12] = {210, 198, 203, 188,
                  200, 224, 220, 217,
                  211, 194, 197, 189};

double iona[8] = {0.0};
```

```
int scores[ ] = {210, 198, 203, 188,
                 200, 224, 220, 217,
                 211, 194, 197, 189};

char name[4] = {‘Z’, ‘o’, ‘l’, ‘a’};

char name[4] = “Zola”;     // array bounds
  overflow

char phrase [ ] = “Hello World”;
```

Initialize an Array

```
double angle[4];        // declaration
```

Example

angle [0] = 6.21;   angle *sub zero* = 6.21
angle [1] = 15.89;   angle *sub one* = 15.89
angle [2] = 7.5;   angle *sub two* = 7.5
angle [3] = -45.7;   angle *sub three* = -45.7

Sequencing Through an Array

- Use the *for* statement to sequence through an array.
- Total the contents of an array:

```
sum = 0;
for(index=0; index < 7; index++)
    sum = sum + grades[index];
```

Loading an Array

```
double grade[10];
int index;

for(index=0; index < 10; index++)
{
    cout<<“Enter a grade “;
    cin >> grade[index];
}
```

Arrays manipulation

Arrays can be filled at declaration or at execution. Such that:-

```
int x[5];
cin>>x[0]>>x[1]>>x[2]>>x[3]>>x[4];
```

Or by using loop:-

```
for(int i=0;i<5;i++)cin>>x[i];
```

Arrays manipulation

Also arrays are printed element by element either by printing each element explicitly, or by using loops.

Example:

```
cout<<x[0]<<x[1]<<x[2]…..<<x[N-1];
for(int j=0;j<N;j++)cout<<x[j];
```

Elements are implicitly mentioned through changing the index j

All elements to be printed are explicitly mentioned

Arrays manipulation

- Arrays can be printed in order: element i is printed before element i+1.

Example: `for(int i=0;i<N;i++)cout<<x[i];`

- Arrays can be printed in reverse order: element N is printed first then N-1 and so on.

```
for(int i=N-1;i>=0;i--)cout<<x[i];
```

Note that: element N is located in location N-1

Arrays manipulation

- Also arrays can be printed in reverse order by:

  for(int i=0;i<N;i++)cout<<x[(N-1)-i];
- How can we print the elements in even locations?

  for(int i=0;i<N;i+=2)cout<<x[i];
- How can we print the elements in odd locations?

  for(int i=1;i<N;i+=2)cout<<x[i];

Arrays manipulation

Example: suppose that you have the following array

1	2	3	4	5	6	7	8	9	10

Write a c++ program in order to revert this array to be

10	9	8	7	6	5	4	3	2	1

Arrays manipulation

```
#include<iostream.h>
void main(){
int x[]={1,2,3,4,5,6,7,8,9,10};
int temp;// for temporary storage
for(int i=0;i<5;i++){
temp=x[i];
x[i]=x[9-i];
x[9-i]=temp;
}//for
}
```

Arrays manipulation

Example: suppose that you have the following array

1	2	3	4	5	6	7	8	9	10

Write a c++ program in order to rotate this array by one element

2	3	4	5	6	7	8	9	10	1

Arrays manipulation

```
#include<iostream.h>
void main(){
int x[10]={1,2,3,4,5,6,7,8,9,10};
temp=x[0];
for(int i=0;i<9;i++)x[i]=x[i+1];
X[9]=temp;
}
```

Finding the Max/Min Value

How would you do this ??

- Set the max or min to element zero.
- Use a *for* loop to cycle through the array.
- Compare each element to the max/min.
- If necessary, assign a new value to max/min.

*

Finding the Maximum Value

```
int temp[30],max;

    max = temp[0];
    for(index=0; index < 30; index++)
        if (temp[index] > max)
                max = temp[index];
    cout<<max<<endl;
```

*

Finding the Minimum Value

```
double find_min(int temp[30])
{
    min = temp[0];
    for(index = 1; index < 30; index++)
        if (temp[index] < min)
                min = temp[index];
    return ( min );
}
```

*

Aggregate Assignment - NOT!

There are no aggregate assignments with arrays. That is, you may not assign one array to another.

```
int x[5] = {11, 22, 33, 44, 55};
int y[5];
```

~~y = x;~~ ~~y[ ] = x[ ];~~

Search

Write a c++ program in order to search within an array for an input integer number.

- If the array is not sorted we use linear search.
- If the array is sorted we can use linear or binary search.

Linear search

```
#include<iostream.h>
void main(){
int x[10]={10,11,9,8,12,13,7,6,14,20};
int n;
cin>>n;
for(int i=0;i<10;i++){if(x[i]==n)cout<<i;break;}
}
```

Binary Search

```
#include<iostream.h>
void main(){
int x[]={1,2,3,4,5,6,7,8,9,10};
int n;cin>>n;
int start=0,end=9,mid;
mid=(start+end)/2;
while(x[mid]!=n){
if(n>x[mid])start=mid;
else if(n<x[mid])end=mid;
mid=(start+end)/2;
}//while
cout<<n<<“is located in location number”<<mid<<endl;
}
```

Array sorting

Write a c++ function in order to sort an array of any given size.

```
void sort(int a[],int size){
int temp;
for(int i=0;i<size-1;i++){
    for(int j=i+1;j<size;j++){
    if(a[j]<a[i]){
temp = a[j];
a[j]=a[i];
a[i]=temp;
}//if
}//inner
} //outer
}
```

Swap elements if a[j] < a[i] to sort the array in ascending order because j is larger than I

Swap elements if a[j]>a[i] to sort the array in descending order

Example

Correct the following code:-

```
int x=5;
int a[x];
```

The corrected code will be

```
int const x=5;
int a[x];
```

2-d arrays

- 2-d array declaration:

  type name[# of rows][# of columns]

Example:

int x[2][2];//to declare 2X2 integer array

- 2-d array filling at declaration time

int x[3][3]={{1,2,3},{4,5,6},{7,8}};//will generate

1	2	3
4	5	6
7	8	0

If we want to manipulate a 2-d array, we need 2 nested loops, one for the rows and the other for columns.

Example

- Write a c++ program to find the transpose of a given 2-d square 4x4 integer array

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

→

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

The contents of locations below the main diagonal are exchanged with the contents of location above the main diagonal

Example

```
#include<iostream.h>
void main(){
int x[4][4]= {{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}};
int temp;
for(int i=1;i<4;i++){//outer
for(int j=0;j<i;j++){//inner
temp=x[i][j];
x[i][j]=x[j][i];
x[i][j]=temp;
}
}
}
```

2-D Array Initialization

- int nums [3] [2] = {7, 4, 1, 8, 5, 2}
- int roomsPSF[3] [7] =
  { {17, 18,19, 110, 111, 112, 113},
  {27, 28, 29, 210, 211,212, 213},
  {37, 38, 39, 310, 311, 312, 313} };

2-D Array Initialization

```
double ATtoCG[2] [3] = {.74, .79, .76,
                        .83, .65, .89};
double ATtoCG[2] [3] = { {.74, .79, .76},
                         {.83, .65, .89}
  };
int scores[4] [3] = {210, 198, 203,
                     188, 200, 224,
                     220, 217, 211,
                     194, 197, 189};
```

2-D Array Initialization

- You can assign values to individual elements:
- ```
  ATtoGC [0] [0] = 0.74;
  ATtoGC [0] [1] = 0.79;
  ATtoGC [0] [2] = 0.76;
  ATtoGC [1] [0] = 0.83;
  ATtoGC [1] [1] = 0.65;
  ATtoGC [1] [2] = 0.89;
  ```

*
  ```

Accessing a 2-D Array

- very similar to creating a multiplication table.
- use nested loops.
  outer *for* loop was for the rows
  inner *for* loop was for the columns

Loading a 2-D Array

The Multiplication Table

```
for (row =1; row <=10; row++)
{
    cout <<row<<" ";
    for (column=1; column <= 10; column++)
        cout << column*row;
    cout << endl;
}
```

Loading a 2-D Array

```
int scores [4][3];

for(row=0; row<4; row++)
    for(col=0; col<3; col++)
    {
        cout<<"Enter the value :";
        cin>>scores[row][col];
    }
```

Displaying a 2-D Array

```
int scores [4] [3];

for(row=0; row<4; row++)
{
    for(col=0; col<3; col++)
        cout  << scores[row][col];
    cout << endl;    // new line for each row
}
```

Displaying a 2-D Array

- **Output**

210	198	203
188	200	224
220	217	211
194	197	189

Strings

A string is a character array terminated by a null.

Write a program to read your name from the keyboard and print it ??

```
#include<iostream.h>
void main ( )
{
   char myname [ 15];
   cout << “Enter Name”;
   cin  >> myname;
   cout << myname;
}
```

Character arrays

Character arrays declaration

- char x[5]={'a','h','m','a','d'};
  cout<<x;//will print ahmad╠╠╠ê
  Because there is no null character (‘\0’) at the end, but
- char x[6]={'a','h','m','a','d‘,’\0’};
  cout<<x;//will print ahmad

Character arrays

- char x[5]="ahmad";

  will generate compilation error, because this way needs a location to put a null character automatically;
- char x[6]="ahmad";

```
cout<<x;// will print ahmad
```

  because there is a null character automatically filled.

Character arrays

Character arrays can be filled at declaration as

- Lists of characters

  If you don't put the null character there will be no null character at the end.
- Strings

  The size of the array must be larger than the number of the letters of the string to automatically contain the null character.

Character arrays

Character arrays can be filled at execution by:-

- cin>>array_name directly.

null character will be assigned

- cin>>array_name[location_number];

  When you fill the character array element by element, there is no null character automatically assigned.

Character arrays can be passed directly to the cin or cout statements

2-d character arrays

Each row in the 2-d character arrays can be treated as if it is 1-d character array in execution.

For example:-

char x[5][20];

Can contain 5 strings of maximum length 20 characters for each.

Example

What is the output of the following code?

char x[5][20];

cin>>x[0];//if you entered “computer”

cout<<x[0];//will print computer

Pointers

```
int *intPtr;
```
Create a pointer

```
intPtr = new int;
```
Allocate memory

```
*intPtr = 6837;
```
Set value at given address

*intPtr ⟶ 6837

intPtr ⟶ 0x0050

```
delete intPtr;
```
Deallocate memory

```
int otherVal = 5;
intPtr = &otherVal;
```
Change `intPtr` to point to a new location

*intPtr ⟶ 5 ⟵ otherVal

intPtr ⟶ 0x0054 ⟵ &otherVal

Pointers

Pointer variables:-

- Contain memory address.
- Pointers contain address of variable.

Pointer declaration:-

- int *mypointer;

We can read this declaration statement from right to left as:-

mypointer is a pointer to integer location.

Pointers

Pointer initialization:-

- int x=5, *xptr=&x;

Makes the pointer xptr points to location x, or store the address of x in location xptr.

- int *xptr = NULL;

Makes the pointer xptr points to nothing.

The pointer type must match the location type that points to.

Pointers

What is the output of the following code?

```
int x=5, *xptr=&x;
*xptr = 7;
cout<<*xptr<<endl;//will print 7
```

* Before the variable name at declaration means that the variable is a pointer

* Before the variable name at execution means that the statement access the content of the location that the pointer points to.

Pointers

What is the output of the following code?

```
int x=5, *xptr =&x;
cout<<*(&x);//will print 5
```

Means that print the content of location that has the address of x

In any statement the existence of * and & will cancel each other

Pointers and arrays

What is the output of the following code?

```
int x[5]={1,2,3,4,5}, *xptr=x;
xptr+=2;//move pointer forward by 2 locations
cout<<xptr[0]<<endl;//print 3
```

Because each array is a static pointer points to the first location in the array.

xptr[0] means the content of the location that the pointer points to.

Pointers and arrays

- What is the output of the following code?

```
int x[5]={1,2,3,4,5},*xptr=x;
xptr+=2;
cout<<xptr[-1]<<endl;//print 2
```

xptr[-1] means that the content of the location that behind the location the pointer xptr points to

Pointers to character arrays

- What is the output of the following code?

```
char x[10]="ahmad";
char *xptr=x;
cout<<x+2;// prints "mad"
cout<<*(x+2);//prints 'm'
```

Printing the character pointers prints the contents of all locations until a null character is encountered.

But printing the content of a character pointer prints the content of the location that the pointer points to.

Dynamic memory allocation

- Dynamic memory allocation is allocating locations in the ***Heap*** area in the memory by using pointers and the operator ***new***
- Dynamic locations can be allocated anywhere in the program by using the ***new*** operator, also it can be deleted anywhere in the program by using the ***delete*** operator.

Dynamic memory allocation

- Example:-

```
int *p1,*p2;
p1=new int(5);     // Makes p1 to allocate new location and initialize it to 5
p2 = new int[5];   // Makes p2 to allocate 5 locations
delete p1;         // Delete the location allocated by p1
delete []p2;       // Delete all locations allocated by p2 but if we write delete p2, only the first location will be deleted
p1=NULL;
p2=NULL;           // These two statements make p1 and p2 to point to NULL (nothing)
```

Dynamic Memory Allocation

- Dynamic allocation allows the creation of arrays whose size is determined at runtime.

Arrays

Stack allocation

```
int intArray[10];
intArray[0] = 6837;
```

Heap allocation

```
int *intArray;
intArray = new int[10];
intArray[0] = 6837;

...

delete[] intArray;
```

Allocating/De-allocating Arrays of Pointers

- As before allocation is performed by the `new` command e.g.

```
int* a=new int[10];  // Allocate an array of 10 integers
```

- `a` is a pointer which contains the memory location of the first element of the array. To access the values stored at the memory locations use:

```
a[0] = 1 ; a[3]=2; etc.
```

- When `a` is no longer needed the memory can be released the `delete []` command e.g.

  delete [] a;

- Do not use the **`delete`** operator to delete arrays, it causes lots of problems use **`delete []`** !!!

Functions

Lecture 9

Functins

- **A function: groups a number of statements into a unit and gives it a name.**

- **The reasons to use the functions is to reduce the program size.**

A function has 3 parts?

1- Function prototype or Function declaration

2- Function call

3- Function implementation or defintion

1. Full Example

```
#include<iostream.h>
void add(int x, int y);           // prototype or declaration

void main(){
Int a=3;
Int b=5;
add(a,b);                         // call function from main
}
void add(int x, int y){           // implementation or definition
    int z;
    z=x+y;
}

// If you define or implement the function after the
   main, YOU MUST put the prototype.
```

2. Full Example

```
#include<iostream.h>

void add(int x, int y){          // implementation or definition
    int z;
    z=x+y;
}

void main(){
Int a=3;
Int b=5;
add(a,b);                        // call function from main
}
// If you define or implement the function before
   the main, YOU CAN remove the prototype. "As You
   Like".
```

3. Another Correct Program

```
#include<iostream.h>
void add(int , int );            // prototype or declaration
void add(int x, int y){          // implementation or definition
    int z;
    z=x+y;
  return;  // Or YOU can Remove the return "As You Like"
}
void main(){
Int a=3;
Int b=5;
add(a,b);                        // call function from main

cout<<add(a,b);          // X     Wrong ???
}
```

4. Another Correct Program

```
#include<iostream.h>
int add(int x, int y);          // prototype or declaration
int add(int x, int y){          // implementation or definition
    int z;
    z=x+y;
return (z);   // or   return z;
}
void main(){
Int a=3;
Int b=5;
Int result;
Result = add(a,b);                      // call function from main
cout<<result;   //Or   cout<< add(a,b);
}
```

Function prototype or declaration

– Describes how a function is called

Example:

void add (int x, int y);   // prototype

OR:

void add(int , int);   // صحيحه

Function prototype

- Function prototype
  - Function name
  - Parameters – what the function takes in
  - Return type – data type function returns
  - Prototype only needed if function definition comes after use in program
  - The function with the prototype:

```
int maximum( int x, int y, int z );
```

- Takes 3 ints
- Returns an int

Examples on Prototype

```
double  AB(double, double);
void  CD(void);
double  times-e(int, int, int, int);
double  myfunc(double, int);
void  print_e(char, double);
```

Functions Definition or implementation

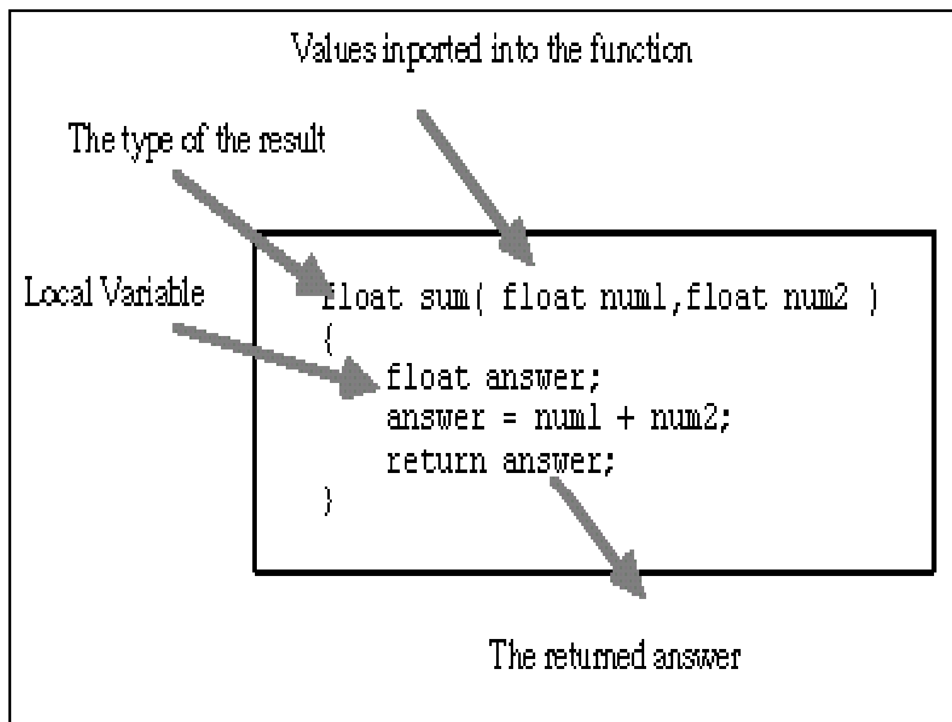
Return-value-type function-name (parameter-list)

{

declarations and statements

return XXX;

}

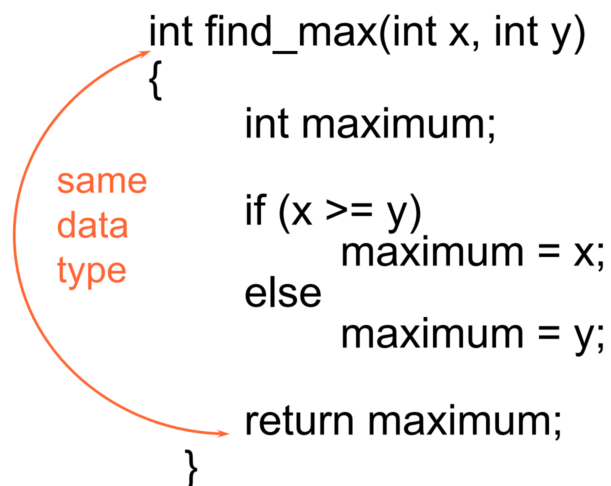


Example

```
void add(int x, int y){
   ….
   ….
  return;    // you can remove it because the returned data type is void
}

Int multiply(int a, float b){
   int z;
   z= a*b;
  return z;  // must return a value of type int, you can't remove it
}
```

Return Statement




```
int find_max(int x, int y)
{
        int maximum;

        if (x >= y)
                maximum = x;
        else
                maximum = y;

        return maximum;
  }
```

*

Function Call

- **Function Call (Call from main function or any other functions or call by itself)**

  Call the function from the main as follow:

```
void main(){
        int a=5;
        int b=3;
        add(a,b);
// function(callee المنادى عليه) is called from main (caller المنادي)
}
```

main function

- **As you know main is a** `special function` **that any C/C++ program must have to be executed.**

Return Statement

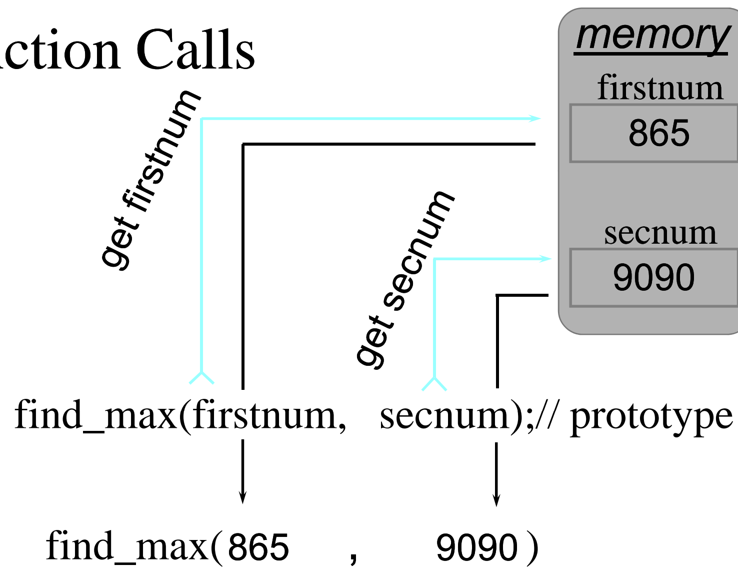
```
int find_max(int x, int y)
{
        int maximum;

        if (x >= y)
                maximum = x;
        else
                maximum = y;

        return maximum;
}
```

same data type

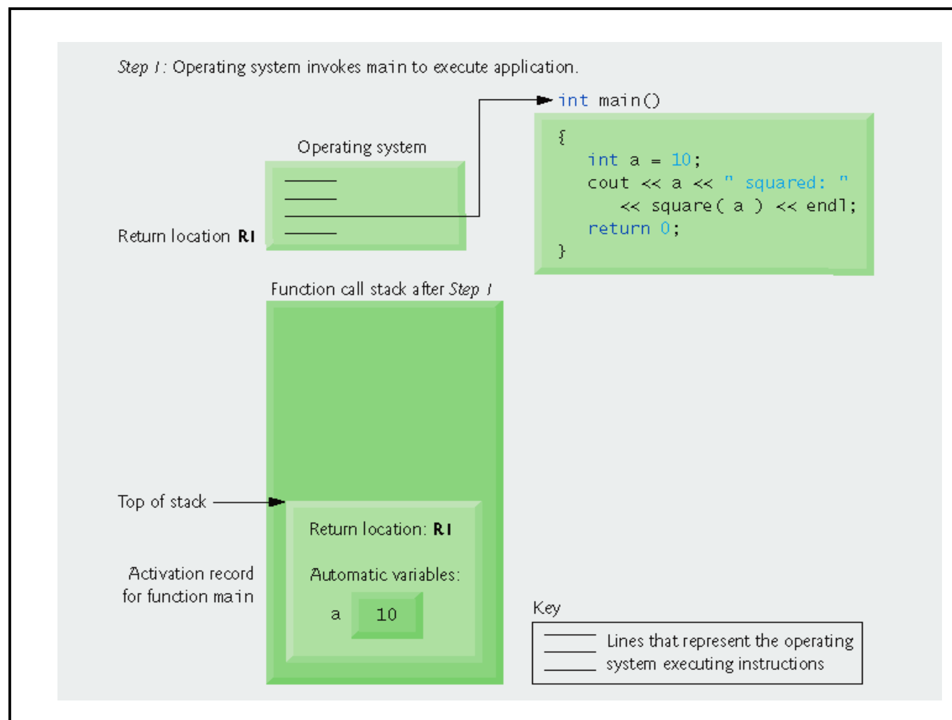
*

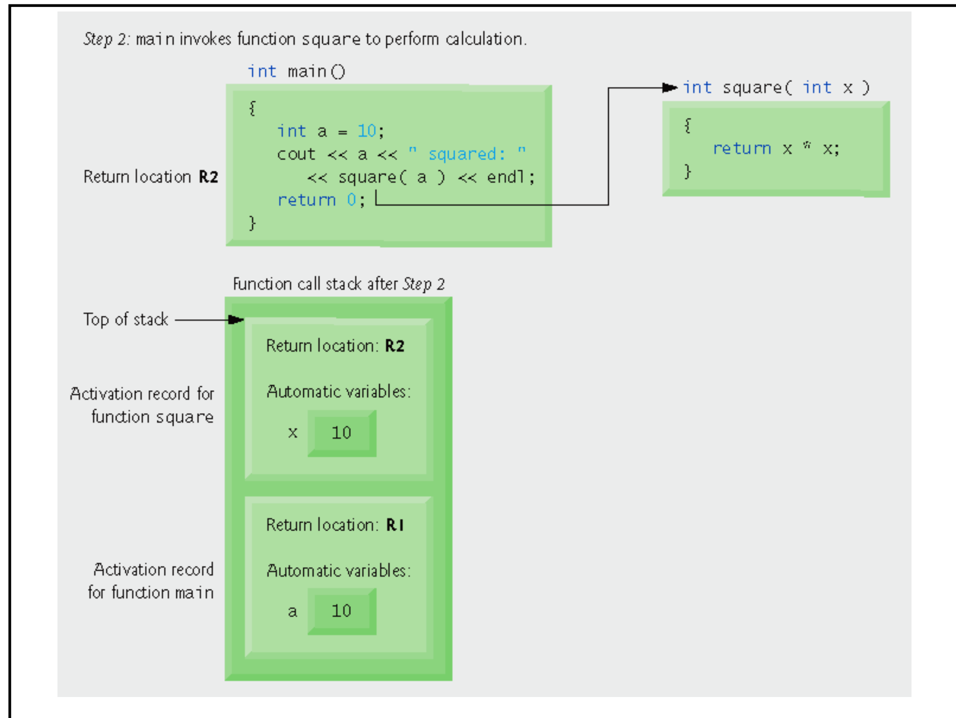


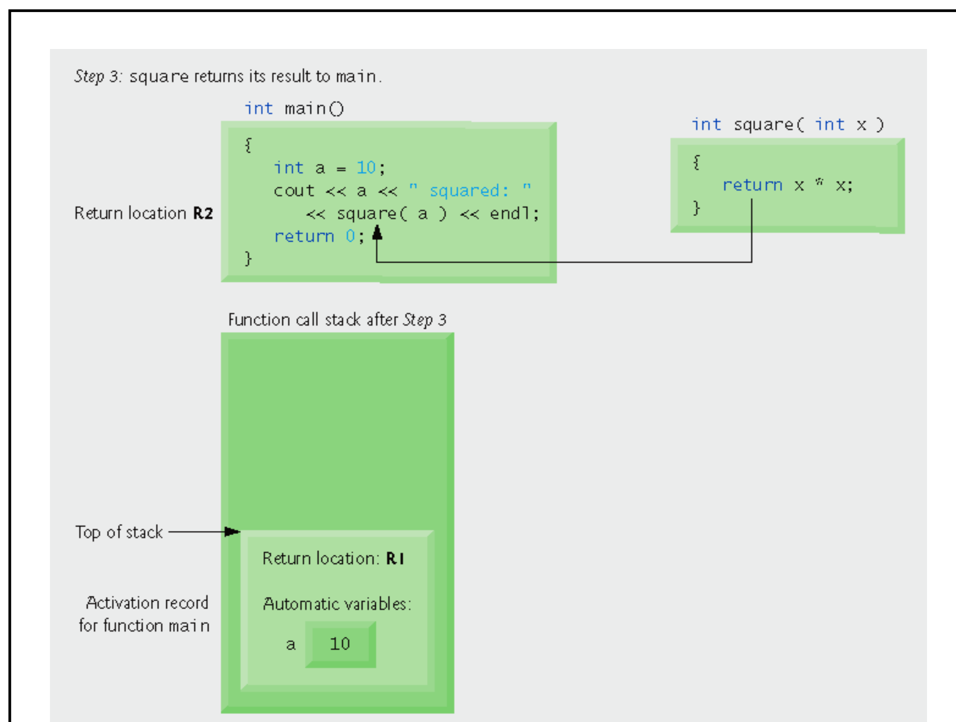
Example: How the function work?

```
#include<iostream.h>
int square (int m);
int main(){
int a=10;
cout<<a<<“squared:”<<square(a)<<endl;
return 0;
}

int square(int x){
int z;
z= x*x;
return z;
}
```




Step 2: main invokes function square to perform calculation.
int main()
{
int a = 10;
cout << a << " squared: "
<< square(a) << endl;
return 0;
}
Return location R2
int square(int x)
{
return x * x;
}
Function call stack after Step 2
Top of stack
Activation record for function square
Return location: R2
Automatic variables:
x 10
Activation record for function main
Return location: R1
Automatic variables:
a 10


Step 3: square returns its result to main.
int main()
{
int a = 10;
cout << a << " squared: "
<< square(a) << endl;
return 0;
}
Return location R2
int square(int x)
{
return x * x;
}
Function call stack after Step 3
Top of stack
Activation record for function main
Return location: R1
Automatic variables:
a 10

Example

- **Example:**
  Write a program to print line of "**" using a function?**

Answer:

```
#include<iostream.h>

void fun();              // prototype  or declaration

void main()
{
    fun();               // call
}

void fun()           // implementation
{
    cout << "*****";
    return;
}
```

Question?

Is it possible for a function to have more than one return statement ?

Yes.

But, one will be executed only

Example

```
void main()
{
    f1();                              // call
}
void f1( )                             // implementation
{
    char a;
    cin >> a;
    switch(a)
    {
    case ‘P': cout << “Palestine";return;
    case ‘N': cout << “Nablus";return;
    }
}
```

Example

Write a program to find the maximum of 3 numbers using a function (call by value?

Answer:

```
#include <iostream.h>

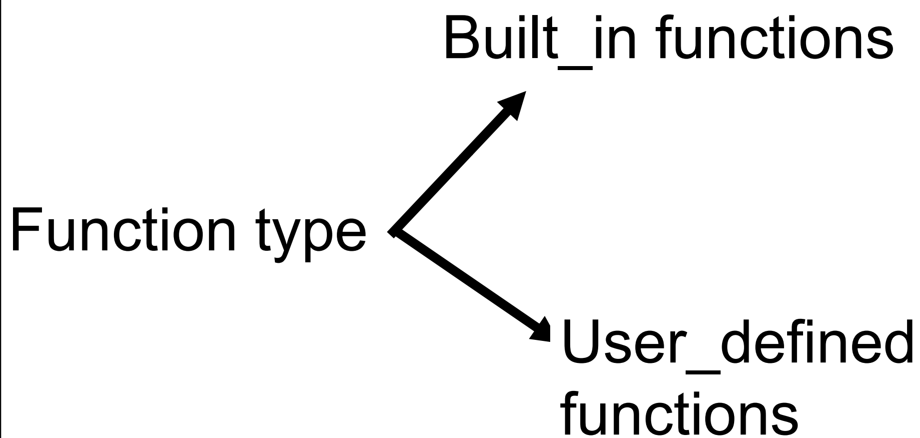
int maximum( int, int, int);          // prototype or declaration

void main()
{
   int a, b, c;
  cout <<"Enter three integers:";
   cin >> a ; cin >> b; cin >> c;
   cout <<"Max value is "<< maximum(a, b, c);          // call the function
}
int maximum( int y, int x, int z)                      // implementation
{
   if( x > y &&y > z)
       return x;
   if ( y > z && y>x )
          return y;
    else
          return z;
}
```

Example

```
#include <iostream.h>

void starline();       // function declaration  (prototype)
void main()
{
        starline();                    // call to function
        starline();                    // call to function
        cout << “Programming in C++”;
}
// function definition or implementation
void starline() {
      for(int j=0; j<45; j++)            // function body
            cout << '*';    cout << endl;
}
```



Built-in Functions

Examples of built-in functions

- abs
- ceil
- cos
- floor
- sin
- pow

Function	Description	Example
ceil(x)	rounds *x* to the smallest integer not less than *x*	ceil(9.2) is 10.0 ceil(-9.8) is -9.0
cos(x)	trigonometric cosine of *x* (*x* in radians)	cos(0.0) is 1.0
exp(x)	exponential function e^x	exp(1.0) is 2.71828 exp(2.0) is 7.38906
fabs(x)	absolute value of *x*	fabs(5.1) is 5.1 fabs(0.0) is 0.0 fabs(-8.76) is 8.76
floor(x)	rounds *x* to the largest integer not greater than *x*	floor(9.2) is 9.0 floor(-9.8) is -10.0
fmod(x, y)	remainder of *x/y* as a floating-point number	fmod(2.6, 1.2) is 0.2
log(x)	natural logarithm of *x* (base *e*)	log(2.718282) is 1.0 log(7.389056) is 2.0
log10(x)	logarithm of *x* (base 10)	log10(10.0) is 1.0 log10(100.0) is 2.0
pow(x, y)	*x* raised to power *y* (x^y)	pow(2, 7) is 128 pow(9, .5) is 3
sin(x)	trigonometric sine of *x* (*x* in radians)	sin(0.0) is 0
sqrt(x)	square root of *x* (where *x* is a nonnegative value)	sqrt(9.0) is 3.0
tan(x)	trigonometric tangent of *x* (*x* in radians)	tan(0.0) is 0

Built_in functions need #include<math.h>

User- defined functions

- Like all the previous example

Functions with Empty Parameter Lists

```
void print(); or void print (void);
```

- The function `print` does not take arguments or parameter and does not return a value

Function Definitions with Multiple Parameters

- Multiple parameters (arguments)
  - Functions often require more than one of information to perform their tasks
  - Specified in both the function prototype as a comma-separated list of parameters

Example:

```
void multiply (int x, int y);
void multiply (int, int);
```

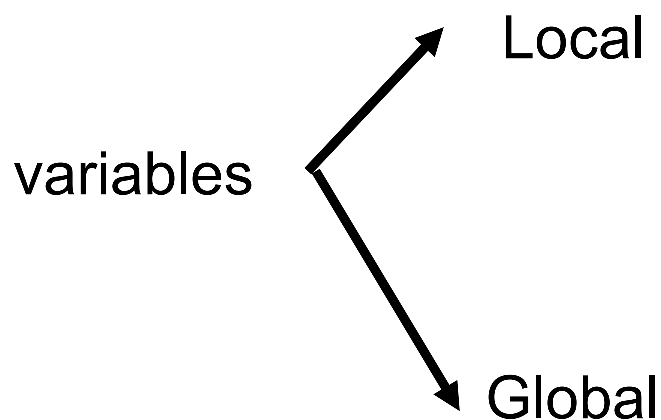
Function Definitions with Multiple Parameters

- **Compiler uses a function prototype to:**
  - Check that calls to the function contain the correct number and types of ***arguments*** in the correct order
  - Ensure that the value returned by the function is used correctly in the expression that called the function

Function return

- **Ways to return control to the calling statement:**
  – If the function does not return a result:
  – If the function does return a result:
    - Program executes the statement `return` *expression*;
      – *expression* is evaluated and its value is returned to the caller

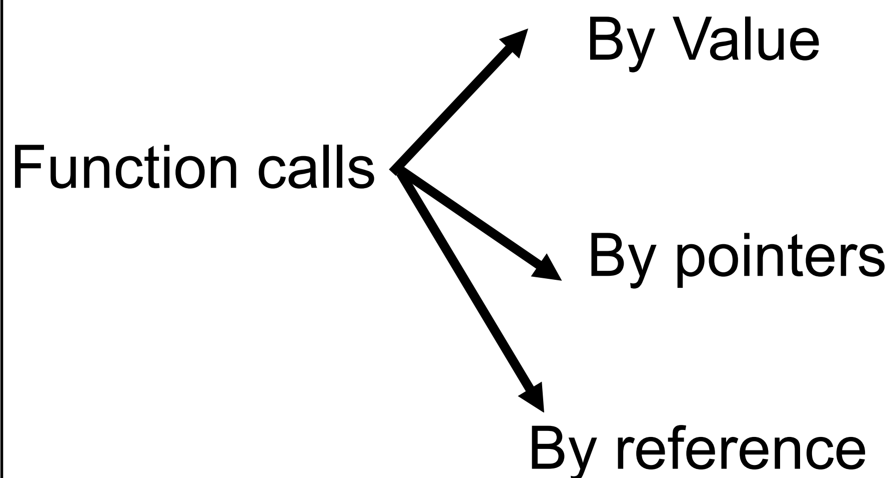
Kinds of variables



Kinds of variables

Local variables: only can used within the scope of definition
Global variable: can be used any where

```
#include<iostream.h>
void add(int x, int y);          // prototype or declaration
void add(int x, int y){          // implementation or definition
    int z;          → Local variable: only can use in main function
    z=x+y;
}
void main(){
Int a=3;
Int b=5;
add(a,b);                        // call function from main
}
```



Passing Arguments or Parameters to a function:

- 1- By value
- 2- By pointers
- 3- By reference parameter

Parameter Passing

1. Passing by Value

- o A copy of the variable contents is passed to the functions
- o Changing the content of an argument in the function has no effect on the associated variable in the calling routine

2. Passing by pointer

- o A pointer that contains the address of variable is passed to the called function

3. Passing by Reference

- o A reference to the variable is passed to the function.
- o If the variable is passed by pointer or by reference then any change in the function will change the original variable in the calling routine .

Full Example

```
#include<iostream.h>
void fun1(int x); //passing by value
void fun2(int *y); //passing by pointer
void fun3(int &z); //passing by reference

void main()
{
int a =1 , b = 2 , c = 3;
cout<<"Before calling"<<endl;
cout<<" a= "<<a<<endl;
cout<<" b= "<<b<<endl;
cout<<" c= "<<c<<endl;
fun1(a); fun2(&b);fun3(c);
cout<<"After calling"<<endl;
cout<<" a= "<<a<<endl;
cout<<" b= "<<b<<endl;
cout<<" c= "<<c<<endl;
}
```

Output of Example

```
void fun1(int x){ x++;} //passing by value
void fun2(int *y){ (*y)++;} //passing by pointer
void fun3(int &z){ z++;} //passing by reference
```

Output

```
Before calling
a= 1
b= 2
c= 3
After calling
a= 1
b= 3
c= 4
```

Default function arguments

- C++ provides default values to be assigned to the function parameters within the function prototype.
- The Default values are used when they are not provided by the function call

Example on **Default function arguments**

```
#include<iostream.h>
void funct1( int n1=4, int n2= 5, int n3 = 6);
int main( )
{
funct1(1,2,3);
funct1(1,2);
funct1(1);
funct1( );
return 0;
}
void funct1( int n1, int n2, int n3 ){
cout<<"n1= "<<n1<<" n2= "<<n2<<" n3= "<<n3<<endl;
}
```

Example on **Default function arguments**

Output:

n1= 1 n2= 2 n3= 3

n1= 1 n2= 2 n3= 6

n1= 1 n2= 5 n3= 6

n1= 4 n2= 5 n3= 6

Inline Function

- Function call overhead may be eliminated, using inline keyword which indicates that a new copy of the function should be placed in the compiled code at each point of reference.

Inline Function, Example

```
#include<iostream.h>
inline bool odd( int x)
{
return (x%2);
}
int main( )
{
if(odd(10)) cout<<"10 is odd"<<endl;
if(odd(11)) cout<<"11 is odd"<<endl;
return 0;
}
```

Function Overloading: (C++ feature)

- Same name with different number or different type of arguments.

Not available in C.

Function Overloading. Example

```
Ex:
        #include<iostream.h>
        #include<math.h>
        int my_abs(int x)
        {
                return abs(x);
        }
        double my_abs(double x)
        {
                return fabs(x);
        }

        int main( )
        {
                int z = -3;
                double w = -4.5;
                cout<<my_abs(z)<<endl;  // 3
                cout<<my_abs(w)<<endl;  // 4.5
                return 0;
        }
```

- The same name
- The same number of arguments
- Different types

Function Overloading. Example

Ex:

```
#include<iostream.h>
#include<math.h>
void out(int x){
        cout<<"function with single argument"<<endl;
        cout<<x<<endl;
}
void out(int x, int y){
        cout<<"function with two arguments"<<endl;
        cout<<x<<y<<endl;
}
void main( ){
        out(3);
        out(4,5);
}
```

- The same name
- The same type
- Different number of arguments

Function Overloading, Example

Note: the type of returned data can not be used to differentiate between the overloaded functions:

Ex:

```
int add(int x, int y);
float add(int x, int y);
```

- The same name
- The same number of arguments with the same type
- Different returned data types