WEEK 11
POINTERS

Dr. Yuzana Win

POINTERS

Objective

After completing this topic, you will be able to know

- Addresses and Pointers
- Pointers and Arrays
- Pointers and Functions
- Memory Management
- Pointer to Pointers
- the advantages and limitations of Pointers

INTRODUCTION TO POINTERS

- A pointer refers to a memory location that contains an address.
- The C++ programming language provides a special mechanism for passing variables to functions that is a little unintuitive, but extremely powerful.
- C++ provides two operators, & and *, which allow you to pass a variable instead of its value.

INTRODUCTION TO POINTERS

- A pointer must be declared and the variable type it points to must be specified.

  Example: int *ptr;
- Address Operator (&) : An address is assigned to a pointer using the address operator.

  Example: prt_v=&x;
- Indirection Operator (*): A value is assigned to a variable it points to using the indirection operator.

  Example: *ptr_v = 77;

POINTER VARIABLES

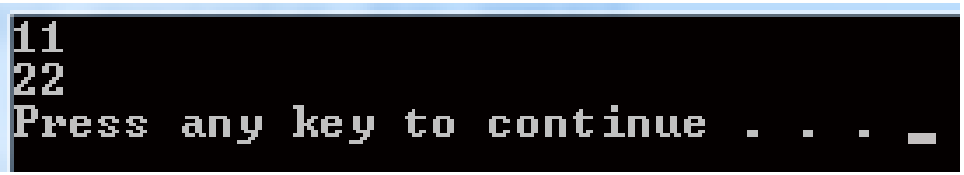
```
// ptrvar.cpp
// pointers (address variables)
#include <iostream>
using namespace std;
int main()
{
int x = 7 ;
int *px ;                    /* px is a pointer to an integer */
px = &x ;                    /* px gets the address of x */
cout << x << endl;           /* show the value of x */
cout << &x << endl ;         /* show the address of x */
cout << *px << endl;         /* show the value of what px points to */
system("pause");
return 0;
}
```

```
7
0x22ff44
7
Press any key to continue . . .
```

ACCESSING THE VARIABLE POINTED TO

```
// ptracc.cpp
// accessing the variable pointed to
#include <iostream>
using namespace std;
int main()
{
int var1 = 11;            //two integer variables
int var2 = 22;
int* ptr;                 //pointer to integers
ptr = &var1;              //pointer points to var1
cout << *ptr << endl;     //print contents of pointer (11)
ptr = &var2;              //pointer points to var2
cout << *ptr << endl;     //print contents of pointer (22)
system("pause");
return 0;
}
```

Output ???



POINTER CONVERSIONS

- One type of pointer can be converted into another type of pointer.
- There are two general categories of conversion:

void * pointers (generic pointers) and other base type pointers.

In C++, it is permissible to assign a **void * pointer** to any other type of pointer. It is also permissible to assign any other type of pointer to a void * pointer.

- Except for void *, all other pointer conversions must be performed by using an explicit cast.

POINTER CONVERSIONS

- Usage of void * pointer

- The void * pointer is used to specify a pointer whose base type is unknown.

- The void * type allows a function to specify a parameter that is capable of receiving any type of pointer argument without reporting a type mismatch.

- It is also used to refer to raw memory (such as that returned by the malloc( ) function) when the semantics of that memory are not known.

- No explicit cast is required to convert to or from a void * pointer.

POINTER TO VOID

```
// ptrvoid.cpp
// pointers to type void
#include <iostream>
using namespace std;
int main()
{
int intvar;              //integer variable
float flovar;            //float variable
int* ptrint;             //define pointer to int
float* ptrflo;           //define pointer to float
void* ptrvoid;           //define pointer to void
ptrint = &intvar;        //ok, int* to int*
// ptrint = &flovar;     //error, float* to int*
// ptrflo = &intvar;     //error, int* to float*
ptrflo = &flovar;        //ok, float* to float*
ptrvoid = &intvar;       //ok, int* to void*
ptrvoid = &flovar;       //ok, float* to void*
system("pause");
return 0;
}
```

POINTER ARITHMETIC

- Pointer Addition or subtraction is done in accordance with the associated data type.
  - int => adds 2 for every increment
  - char => adds 1 for every increment
  - float => adds 4 for every increment

- Example

```
int * ptr , i=5;
ptr= &i;        => let ptr = 1000 (location of i)
   ptr ++;      => now ptr = 1001 but it is 1002 or +2(integers)
++*ptr or (*ptr)++   => increments the value of i by 1
*ptr++               => increments the address of i by 1
```

- ++ptr and ptr++ are both equivalent to ptr + 1 (though the point in the program when ptr is incremented may be different), incrementing a pointer using the unary ++ operator, either pre- or post-, increments the address it stores by the amount sizeof(type) where "type" is the type of the object pointed to. (i.e. 4 for an integer)

POINTERS AND ARRAYS

- The address of the 1'st element (zeroth element) of an array is represented by its name
- Each time you declare an array, you also declare implicitly a pointer to the "zeroth" element. The name of this pointer is the name of the array.

```
int  a[5] ;
a        =>  &a[0]
a+1      =>  &a[0] + 1
```

(a+i); => i varies from 0 to 4 ; traverses through the address of each element in the array 'a'

```
a+i  (i=0)  =>  &a[0]
a+i  (i=1)  =>  &a[1]
a+i  (i=2) =>   &a[2]
a+i  (i=3) =>   &a[3]
a+i  (i=4) =>   &a[4]
```

POINTERS AND ARRAYS

- In C++, the standard states that wherever we might use &var_name[0] we can replace that with var_name, thus in our code where we wrote:

  int *ptr, my_array[]={10,20,30,40 };

  ptr = &my_array[0];

  – we can write:

  ptr = my_array;

- ptr is a variable, my_array is a constant. That is, the location at which the first element of my_array will be stored cannot be changed once my_array[] has been declared
- We cannot write

  my_array = ptr;

  – Or

  my_array++;

POINTERS AND ARRAYS : EXAMPLE 1

```
// arrnote.cpp
// array accessed with array notation
#include <iostream>
using namespace std;
int main()
{     //array
   int intarray[5] = { 31, 54, 77, 52, 93 };
   for(int j=0; j<5; j++)                  //for each element,
        cout << intarray[j] << endl;         //print value
   return 0;
}
```

```
31
54
77
52
93
Press any key to continue . . .
```

POINTER CONSTANTS AND POINTER VARIABLES

```
// ptrinc.cpp
// array accessed with pointer
#include <iostream>
using namespace std;
int main()
{
int intarray[] = { 31, 54, 77, 52, 93 }; //array
int* ptrint;                          //pointer to int
ptrint = intarray;                    //points to intarray
for(int j=0; j<5; j++)                //for each element,
cout << *(ptrint++) << endl;          //print value
return 0;
}
```

```
31
54
77
52
93
Press any key to continue . . .
```

POINTERS AND FUNCTIONS

```
// passref.cpp
// arguments passed by reference
#include <iostream>
using namespace std;
int main()
{
  void centimize(double&);    //prototype
  double var = 10.0;
  cout << “var = ” << var << “ inches”
   <<   endl;
  centimize(var);
  cout << “var = ” << var << “ centimeters”
<< endl;
return 0;
}
```

```
// --------------------
void centimize(double& v)
{
v *= 2.54;   //v is the same as var
}
```

```
var = 10 inches
var = 25.4 centimeters
Press any key to continue . . .
```

POINTERS AND FUNCTIONS

```
// passptr.cpp
// arguments passed by pointer
#include <iostream>
using namespace std;
int main()
{
   void centimize(double*);    //prototype
   double var = 10.0;
   cout << “var = ” << var << “ inches” <<
endl;
   centimize(&var);
   cout << “var = ” << var << “ centimeters”
<< endl;
return 0;
}
```

```
// --------------------
void centimize(double* ptrd)
{
   *ptrd *= 2.54;
   //*ptrd is the same as var
}
```

```
var = 10 inches
var = 25.4 centimeters
Press any key to continue . . .
```

PASSING ARRAYS

```
// passarr.cpp
// array passed by pointer
#include <iostream>
using namespace std;
const int MAX = 5;  //number of array elements
int main()
{
void centimize(double*);  //prototype
double varray[MAX] = { 10.0, 43.1, 95.9, 59.7, 87.3
};
centimize(varray);
for(int j=0; j<MAX; j++)  //display new array values
cout << "varray[ " << j << "]="
<< varray[j] << " centimeters" << endl;
return 0;
}
```

```
// --------------------
void centimize(double* ptrd)
{
for(int j=0; j<MAX; j++)
*ptrd++ *= 2.54;
//ptrd points to elements of
//varray
}
```

```
varray[0]=25.4 centimeters
varray[1]=109.474 centimeters
varray[2]=243.586 centimeters
varray[3]=151.638 centimeters
varray[4]=221.742 centimeters
Press any key to continue . . .
```

ORDERING WITH POINTERS

```
// ptrorder.cpp
// orders two arguments using pointers
#include <iostream>
using namespace std;
main()
{
void order(int*, int*);        //prototype
int n1=99, n2=11;        //one pair ordered, one not
int n3=22, n4=88;
order(&n1, &n2);        //order each pair of numbers
order(&n3, &n4);
cout << "n1=" << n1 << endl;
cout << "n2=" << n2 << endl;
cout << "n3=" << n3 << endl;
cout << "n4=" << n4 << endl;
}
```

```
// --------------------
void order(int* numb1, int*
numb2) //orders two numbers
{
if(*numb1 > *numb2) {
int temp = *numb1;
//swap them
*numb1 = *numb2;
*numb2 = temp;
}
}
```

```
n1=11
n2=99
n3=22
n4=88
Press any key to continue . . .
```

THE BUBBLE SORT

```
// ptrsort.cpp
// sorts an array using pointers
#include <iostream>
using namespace std;
main()
{
void bsort(int*, int);       //prototype
const int N = 10;            //array size
//test array
int arr[N] = { 37, 84, 62, 91, 11, 65, 57, 28, 19, 49 };
bsort(arr, N);               //sort the array
for(int j=0; j<N; j++)       //print out sorted array
cout << arr[j] << " ";
cout << endl; }
```

```
11 19 28 37 49 57 62 65 84 91
Press any key to continue . . .
```

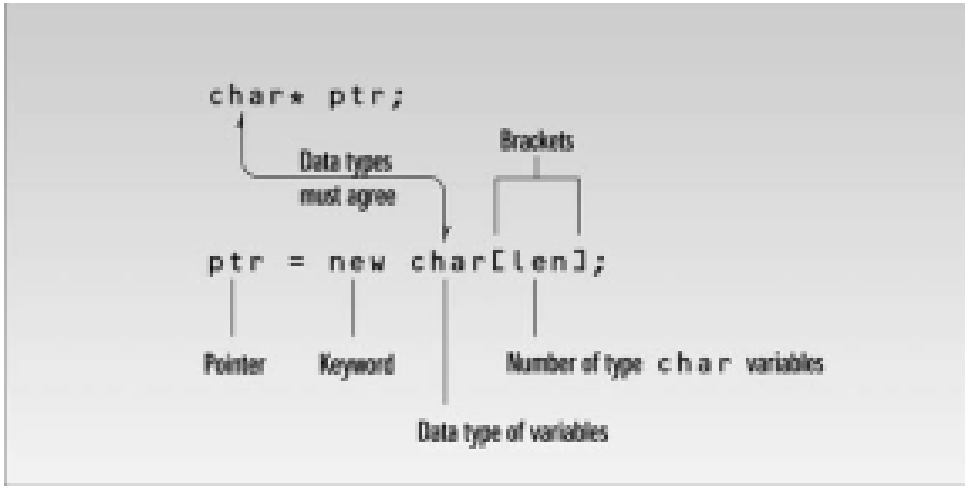
```
void bsort(int* ptr, int n)
{
void order(int*, int*);
int j, k;          //indexes to array
for(j=0; j<n-1; j++)  //outer loop
   for(k=j+1; k<n; k++)
      order(ptr+j, ptr+k);
}
//---------------------------------
void order(int* numb1, int*
numb2)  //orders two numbers
{ if(*numb1 > *numb2) {
int temp = *numb1;
*numb1 = *numb2;
*numb2 = temp;
} }
```

THE NEW & DELETE OPERATOR

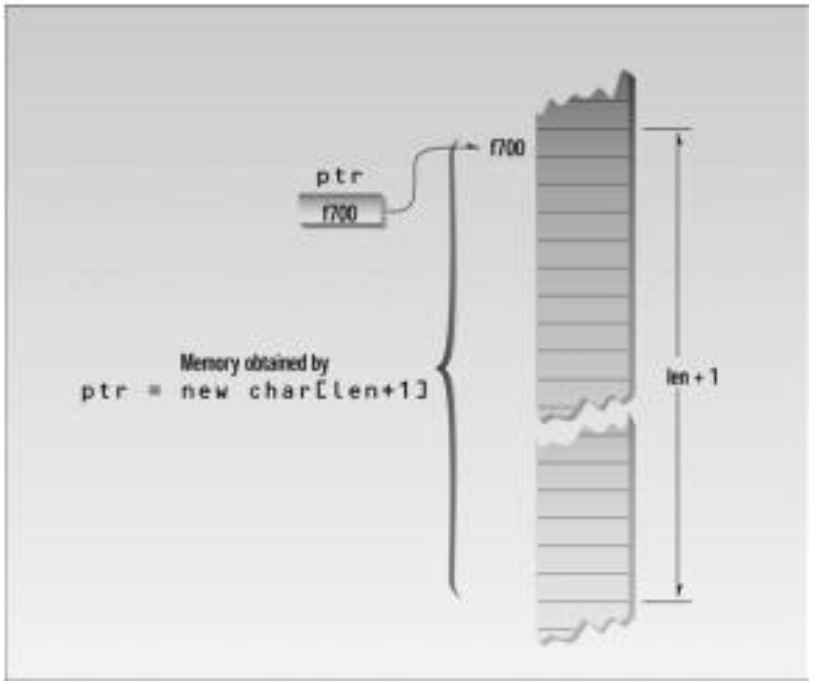
```
// newintro.cpp
// introduces operator new and delete operator
#include <iostream>
#include <cstring>          //for strlen
using namespace std;
main()
{
char* str = "Idle hands are the devil's workshop.";
int len = strlen(str);          //get length of str
char* ptr;                      //make a pointer to char
ptr = new char[len+1];     //set aside memory: string + '\0'
strcpy(ptr, str);               //copy str to new memory area ptr
cout << "ptr=" << ptr << endl;       //show that ptr is now in str
delete[] ptr;                           //release ptr's memory
}
```

```
ptr=Idle hands are the devilfs workshop.
Press any key to continue . . .
```

THE NEW OPERATOR



Syntax of the new operator



Memory obtained by the new operator

POINTER TO OBJECTS

```
// englptr.cpp
// accessing member functions by pointer
#include <iostream>
using namespace std;
class Distance              //English Distance class
{
   private:  int feet;
             float inches;
   public:
        void getdist()       //get length from user
{       cout << "\nEnter feet: ";  cin >> feet;
        cout << "Enter inches: ";  cin >> inches;
}
       void showdist()       //display distance
     {cout << feet << "\'-" << inches << '\"'; }
};
```

```
int main()
{
Distance dist;
dist.getdist();
dist.showdist();
Distance* distptr;
distptr = new Distance;
distptr->getdist();
distptr->showdist();
cout << endl;
return 0;
}
```

```
Enter feet: 10
Enter inches: 6.25
10-'6.25"
Enter feet: 6
Enter inches: 4.75
6-'4.75"
Press any key to continue . . .
```

POINTERS AND 2-D ARRAYS

- In a 2-D array each row may be thought of as an 1-D array
- When a 2-D array is declared in the following way,

  my_array[10][5];
- The expression (my_array + 0) gives the address of the 0th element of the 2-D array (row address)
- The expression *(my_array + 0) gives the base address of 0th 1-D array (column address)
- The expression *(*(my_array + 0) + 0) gives the value of the element my_array[0][0]

POINTERS AND ARRAYS : EXAMPLE 2

```
#include <iostream>
using namespace std;
int main()
{
int Arr[2][3]={{5,3,6},{7,2,4}};
int *ip;
int i,j;
ip = *Arr;
cout << "Starting address of array row index 0 = " << ip;
ip=&Arr[0][0];
cout << "\nStarting address of array row index 0 = " << ip << endl;
for(i=0;i<2;i++)
{     for(j=0;j<3;j++)
            cout << "  \t  " << &Arr[i][j];
      cout << "\n";
}
```

POINTERS AND ARRAYS : EXAMPLE 2

```
cout << "\nStarting address of array row index 0 =" << *(Arr+0); //row 0 address
cout << "\nStarting address of array row index 1 =" << *(Arr+1)<< endl; // row 1 address
for(i=0;i<2;i++)
{
      for(j=0;j<3;j++)
            cout << "\n" << *(Arr+i)+j << "  " << *(*(Arr+i)+j);
      cout << "\n";
}
system("pause");
return 0;
}
```

```
Starting address of array row index 0 = 0x22ff20
Starting address of array row index 0 = 0x22ff20
          0x22ff20          0x22ff24          0x22ff28
          0x22ff2c          0x22ff30          0x22ff34

Starting address of array row index 0 =0x22ff20
Starting address of array row index 1 =0x22ff2c

0x22ff20  5
0x22ff24  3
0x22ff28  6

0x22ff2c  7
0x22ff30  2
0x22ff34  4
Press any key to continue . . .
```

CHARACTER POINTERS

- Character pointer is a pointer, which is capable of holding the address of a character variable. Suppose if we have a character array declared as

  char name[30] = {“Data Structures”};

- If we need to store the address of this array we need a character pointer declared as follows

  char *p = NULL;

- Once the pointer is declared we need to assign the address of the array to this pointer. i.e.

  p = name;

- Now issue the following output statements and check the output

  cout << “character output = \n” << *p;

  cout << “String output =” << p;

CHARACTER POINTERS (CONT.)

- We know that when we refer to a pointer with the indirection operator ,we refer to the address of the content pointed by the pointer. The above output statements will product as follows:

  character output = D

  String output = Data Structures

- The reason for the output produced by the second *cout* statement is because of the string format specifier, which will print the string till it encounters a '\0' character.

POINTERS AND ARRAYS : EXAMPLE 3

```
#include <iostream>
using namespace std;
int main()
{
char Charr[]={'A','Y','U'};
char *cp;
int i;
/*cp=&Charr[0];
for (i=0;i<3;i++)
cout << "\n“ << cp++ << *cp; */
i=sizeof(Charr)/sizeof(char);      // no of element
for(cp=&Charr[0];cp<=&Charr[i-1];cp++)
cout << “\n” << cp << *cp;
system(“pause”);
return 0; }
```

```
AYUA

YUY

UU
Press any key to continue . . .
```

MALLOC() FUNCTION

- malloc( ) is the function used for memory allocation in C++. malloc() takes one argument that is the number of bytes to allocate. Suppose P is a pointer then

```
int *p;
p = (int *)malloc(sizeof(int));
```

- The above assignment will allocate memory for holding an integer value.

- The system allocates memory from the heap and not from the stack.

- For freeing the allocated memory the function used is,

```
free(<pointer variable>);
```

ARRAYS OF POINTERS

- Pointers can be arrayed like any other data type.
- Declaration Syntax : int *x[5];
- Can be used different columns
- x[0]=malloc(sizeof(int)); // no. of columns is 1
- x[1]= malloc (3*sizeof(int)); // no. of columns is 3
- x[2]= malloc (2*sizeof(int)); // no. of columns is 2

POINTERS AND ARRAYS : EXAMPLE 4

```
#include <iostream>
using namespace std;
int main()
{
int *Arr[5];
int i;
Arr[0]= (int *)malloc(3*sizeof(int));
Arr[1]= (int *)malloc(2*sizeof(int));
Arr[2]= (int *)malloc(4*sizeof(int));
for(i=0;i<3;i++)
{
 cout << "Enter a number : ";
 cin >> Arr[0][i];
}

for(i=0;i<2;i++)
{
 cout << "Enter a number : ";
 cin >> Arr[1][i];
}

for(i=0;i<4;i++)
{
 cout << "Enter a number : ";
 cin >> Arr[2][i];
}
cout << "\n";
for(i=0;i<3;i++)
        cout << Arr[0][i] << endl;
cout << "\n";
```

POINTERS AND ARRAYS : EXAMPLE 4

```
for(i=0;i<2;i++)
        cout << Arr[1][i] << endl;
cout << "\n";
for(i=0;i<4;i++)
        cout << Arr[2][i] << endl;
cout << "\n";
    system("pause");
    return 0;
}
```

```
Enter a number : 10
Enter a number : 24
Enter a number : 35
Enter a number : 45
Enter a number : 67
Enter a number : 78
Enter a number : 98
Enter a number : 56
Enter a number : 34

10
24
35

45
67

78
98
56
34

Press any key to continue . . .
```

POINTERS TO POINTERS

```
// persort.cpp
// sorts person objects using array of pointers
#include <iostream>
#include <string>      //for string class
using namespace std;
class person                    //class of persons
{     protected:
                  string name;              //person's name
      public:
                  void setName()            //set the name
                  { cout << "Enter name: "; cin >> name; }
                  void printName()          //display the name
                  { cout << endl << name; }
                  string getName()          //return the name
                  { return name; }
};
```

POINTERS TO POINTERS

```
int main()
{
    void bsort(person**, int);     //prototype
    person* persPtr[100];    //array of pointers to persons
    int n = 0;                    //number of persons in array
    char choice;                  //input char
    do {                          //put persons in array
        persPtr[n] = new person;    //make new object
        persPtr[n]->setName();      //set person's name
        n++;                        //count new person
        cout << "Enter another (y/n)? ";
        cin >> choice;              //   person?
}
while( choice=='y' );          //quit on 'n'
cout << "\nUnsorted list:";
for(int j=0; j<n; j++)         //print unsorted list
{ persPtr[j]->printName(); }
bsort(persPtr, n);             //sor
cout << "\n";
cout << "-----------------";
cout << "\nSorted list:";
for(int j=0; j<n; j++)         //pri
{ persPtr[j]->printName(); }
cout << endl;
system("pause");
return 0;
}  //end main()
```

POINTERS TO POINTERS

```
void bsort(person** pp, int n)    //sort pointers to persons
{
    void order(person**, person**);  //prototype
     int j, k;                       //indexes to array
     for(j=0; j<n-1; j++)            //outer loop
          for(k=j+1; k<n; k++)       //inner loop starts at outer
               order(pp+j, pp+k);    //order the pointer contents
}
void order(person** pp1, person** pp2)
 //orders two pointers
{    if( (*pp1)->getName() > (*pp2)->getName() )
       {   person* tempptr = *pp1;     //swap the pointers
           *pp1 = *pp2;
           *pp2 = tempptr;
        }
}
```

```
Enter name: Washington
Enter another (y/n)? y
Enter name: Adams
Enter another (y/n)? y
Enter name: Jefferson
Enter another (y/n)? y
Enter name: Madison
Enter another (y/n)? n

Unsorted list:
Washington
Adams
Jefferson
Madison
-----------------
Sorted list:
Adams
Jefferson
Madison
Washington
Press any key to continue . . .
```

LIMITATIONS OF POINTERS

- If memory allocations are not free properly it can cause memory leakages
- If not used properly can make the program difficult to understand

SUMMARY (POINTERS)

- A pointer is a data type whose value is used to refer to ("points to") another value stored elsewhere in the computer memory. Obtaining the value that a pointer refers to is called dereferencing the pointer.

- Pointers are variables that can hold the address of another variable

- Pointer addition and subtraction involves incrementing/ decrementing pointer depending upon the associated data type

- The difference between arrays and pointers is array name is a fixed pointer, whereas a pointer can point to any element of an array

ASSIGNMENT

- Write a program that reads a group of numbers from the user and places them in an array of type float. Once the numbers are stored in the array, the program should average them and print the result. User pointer notation wherever possible.

ASSIGNMENT

- Modify the person class in the PERSORT program in this topic so that it includes not only a name, but also a salary item of type float representing the person's salary.
- You'll need to change the setName() and printName() member functions to setData() and printData(), and include in them the ability to set and display the salary as well as the name. You'll also need a getSalary() function. Using pointer notation, write a salsort() function that sorts the pointers in the persPtr array by salary rather than by name. Try doing all the sorting in salsort(), rather than calling another function as PERSORT does. If you do this, don't forget that -> takes precedence over *, so you'll need to say

```
if( (*(pp+j))->getSalary() > (*(pp+k))->getSalary() )
    { /* swap the pointers */ }
```