

Algonquin College

Pointers

Created by Rex Woollard

Use PageUp and PageDn to move from screen to screen. Online, use Microsoft iExplorer in slide show mode.

Click on speaker to play sound.



Addresses of Variables

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

```
void main()
{
  int n1 = 11;
  int n2 = 22;
  int n3 = 33;
```

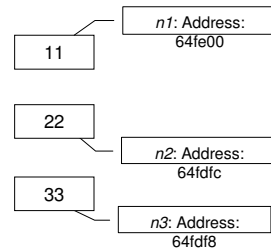
```
  cout << &n1 << endl;
  cout << &n2 << endl;
  cout << &n3 << endl;
}
```

Output

```
0x0064fe00
0x0064fdfc
0x0064fdf8
```

Ampersand asks for the **address of** something.

Memory Map



Pointer Variables (address variables): 1

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

```
void main()
{
  int n1 = 11;
  int n2 = 22;
  cout << &n1 << endl << &n2 << endl << endl;
}
```

```
int* pnPtr;

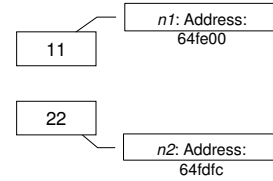
pnPtr = &n1;
cout << pnPtr << endl;

pnPtr = &n2;
cout << pnPtr << endl;
}
```

Output

```
0x0064fe00
0x0064fdfc
```

Memory Map



Accessing the Variable Pointed To: 1

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

```
void main()
{
  int n1 = 11;
  int n2 = 22;
  cout << &n1 << endl << &n2 << endl << endl;
}
```

```
int* pnPtr;

pnPtr = &n1;
cout << *pnPtr << endl;

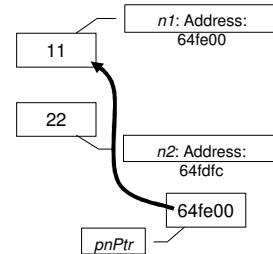
pnPtr = &n2;
cout << *pnPtr << endl;
}
```

Output

```
11
```

Asterisk accesses the content of memory pointed to by `pnPtr`. The variable `pnPtr` points at `64fe00`. The content of that location is `11`

Memory Map



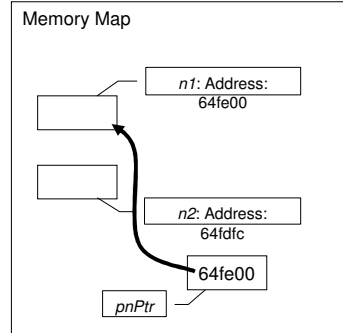
Using Pointers to Change Values: 1

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

void main()
{
    int n1, n2;
    int* pnPtr;
    pnPtr = &n1;
    *pnPtr = 37;
    n2 = *pnPtr;

    cout << n2 << endl;
}
```

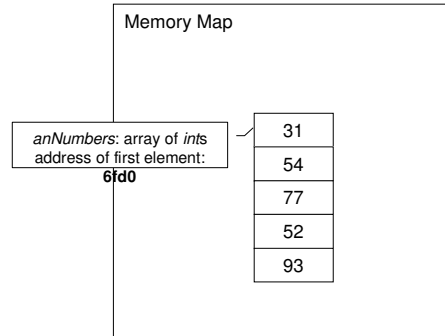
Assign address of *n1* to *pnPtr*.



Other Access Using Pointers: 1a

```
void main() // accessed with pointer
{
    int anNumbers[] = { 31, 54, 77, 52, 93 };
    int* panNumbers = anNumbers;

    for (int j=0; j<5; j++) {
        cout << *panNumbers << endl;
        panNumbers++;
    }
}
```



Other Access Using Pointers: 2a

```
void main() // accessed with pointer
{
  int anNumbers[] = { 31, 54, 77, 52, 93 };

  int* panNumbers = anNumbers;

  for (int j=0; j<5; j++) {
    cout << *panNumbers << endl;
    panNumbers++;
  }
}
```

Output

31
54
77
52
93

Memory Map

anNumbers: array of ints
address of first element:
6fd0

31
54
77
52
93

```
void main() // accessed with pointer notation
{
  int anNumbers[5] = { 31, 54, 77, 52, 93 };

  for (int j=0; j<5; j++)
    cout << *(anNumbers+j) << endl;
}
```

Output

31

anNumbers: use of name without [] means address of first element: 6fd0

First time through j has the value 0.

Address of anNumbers + j is 6fd0. Apply the dereference operator to get the contents and output 31



Other Access Using Pointers: 3

```
void main() // accessed with pointer
{
  int anNumbers[] = { 31, 54, 77, 52, 93 };

  int* panNumbers = anNumbers;

  for (int j=0; j<5; j++) {
    cout << *panNumbers << endl;
    panNumbers++;
  }
}
```

Output

31
54
77
52
93

Memory Map

anNumbers: array of ints
address of first element:
6fd0

31
54
77
52
93

```
void main() // accessed with pointer notation
{
  int anNumbers[5] = { 31, 54, 77, 52, 93 };

  for (int j=0; j<5; j++)
    cout << *(anNumbers+j) << endl;
}
```

Output

31
54
77
52
93

Traditional array notation to access elements.

```
void main() // accessed with array notation
{
  int anNumbers[5] = { 31, 54, 77, 52, 93 };

  for (int j=0; j<5; j++)
    cout << anNumbers[j] << endl;
}
```

Output

31
54
77
52
93



Reference Arguments

```
void main() // by reference
{
    double dVar = 10.0;
    cout << "var = " << dVar << " in" << endl;

    centimize(dVar);
    cout << "var = " << dVar << " cm" << endl;
}

void centimize(double& rdVar)
{
    rdVar = rdVar * 2.54;
}
```

Output

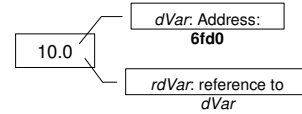
```
var = 10.0 in
var = 25.4 cm
```

```
void main() // by pointer
{
    double dVar = 10.0;
    cout << "var = " << dVar << " in" << endl;

    centimize(&dVar);
    cout << "var = " << dVar << " cm" << endl;
}

void centimize(double* pdVar)
{
    *pdVar = *pdVar * 2.54;
}
```

Memory Map



Pointer Arguments: 1

```
void main() // by reference
{
    double dVar = 10.0;
    cout << "var = " << dVar << " in" << endl;

    centimize(dVar);
    cout << "var = " << dVar << " cm" << endl;
}

void centimize(double& rdVar)
{
    rdVar = rdVar * 2.54;
}
```

Output

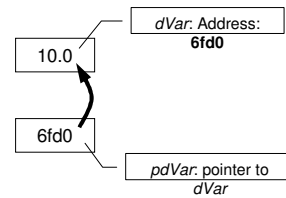
```
var = 10.0 in
var = 25.4 cm
```

```
void main() // by pointer
{
    double dVar = 10.0;
    cout << "var = " << dVar << " in" << endl;

    centimize(&dVar);
    cout << "var = " << dVar << " cm" << endl;
}

void centimize(double* pdVar)
{
    *pdVar = *pdVar * 2.54;
}
```

Memory Map



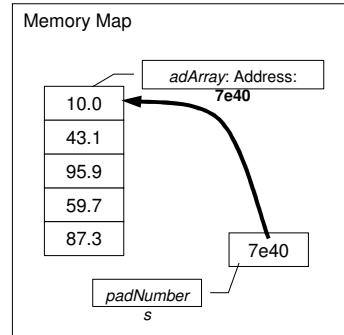
Array Passed as Pointer: 1

```
void main()
{
double adArray[MAX] = { 10.0, 43.1, 95.9, 59.7, 87.3 };

centimize(adArray, MAX);

for (int j=0; j<MAX; j++)
cout << "adArray[" << j << "]=" << adArray[j] << " cm" << endl;
}

void centimize(double* padNumbers, int nSize)
{
for (int j=0; j<nSize; j++) {
*padNumbers = *padNumbers * 2.54;
++padNumbers;
}
}
```



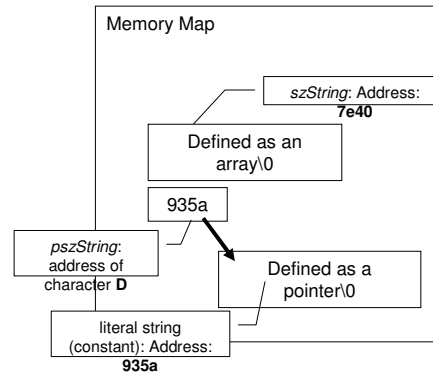
Strings Using Array and Pointer Notation: 1

```
void main()
{
char szString[] = "Defined as an array";
char* pszString = "Defined as a pointer";

cout << szString << endl;
cout << pszString << endl;

// szString++; // can't do this; szString is constant
pszString++; // this is OK, pszString is a pointer

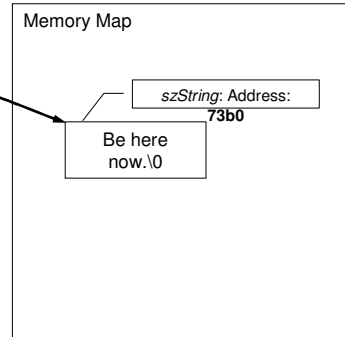
cout << pszString << endl;
}
```



Displays a String using Pointer Notation: 1

```
void main()
{
char szString[ ] = "Be here now.";
DisplayString(szString);
}

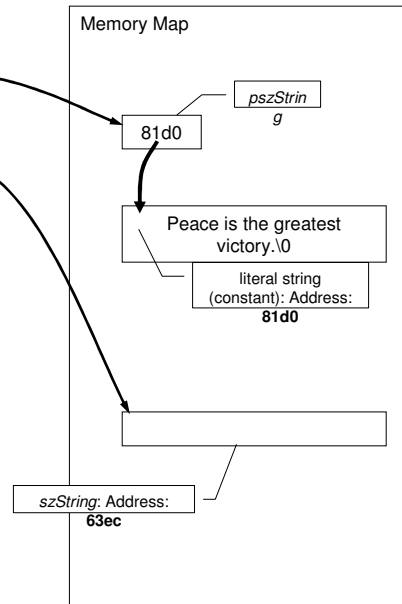
void DisplayString(char* psz)
{
while (*psz != '\0') {
cout << *psz;
psz++;
}
cout << endl;
}
```



Copies one String to Another Using Pointers: 1

```
void main()
{
char* pszString = "Peace is the greatest victory.";
char szString[80];
CopyString(szString, pszString);
cout << szString << endl;
}

void CopyString(char* pszDest, const char*
pszSource)
{
while (*pszSource != '\0') {
*pszDest = *pszSource;
pszDest++;
pszSource++;
}
*pszDest = '\0';
}
```

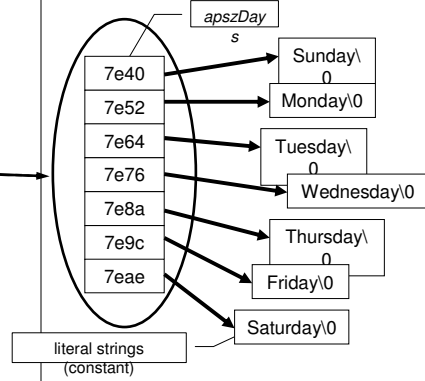


An Array of Pointers to Strings: 1

```
const int DAYS = 7;
void main()
{
  char* apszDays[DAYS] = { "Sunday", "Monday",
    "Tuesday",
    "Wednesday", "Thursday",
    "Friday", "Saturday" };

  for (int j=0; j<DAYS; j++)
    cout << apszDays[j] << endl;
}
```

Memory Map



The new Operator: 1

```
void main()
{
  char* pszString1 = "Idle hands are the devil's
  workshop.";
  int nLen = strlen(pszString1);

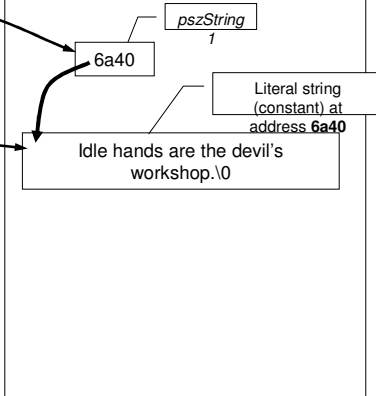
  char* pszString2;
  pszString2 = new char[nLen+1];

  strcpy(pszString2, pszString1);

  cout << "pszString2=" << pszString2 << endl;

  delete [ ] pszString2;
}
```

Memory Map



Using **new** to Get Memory for Strings: 1

```
class String {
private:
    char* pszString;
public:
    String(char* psz)
    {
        int nLength = strlen(psz);
        pszString = new char[nLength + 1];
        strcpy(pszString, psz);
    }
    ~String() { delete [] pszString; }
    void Display() { cout << pszString << endl; }
};

void main()
{
    String s1 = "One ring to rule them all.";
    cout << "s1=";
    s1.Display();
}
```

Memory Map



An Array of Pointers to Objects: 1

```
class Person {
protected:
    char szName[40];
public:
    void SetName()
    { cout << "Enter szName: "; cin >> szName; }
    void DisplayName()
    { cout << "\n Name is: " << szName; }
};

void main()
{
    Person* persPtr[14];
    int n = 0;
    char cChoice;
    do {
        persPtr[n] = new Person;
        persPtr[n]->SetName();
        n++;
        cout << "Enter another (y/n)? ";
        cin >> cChoice;
    } while (cChoice == 'y');

    for (int j=0; j<n; j++) {
        cout << "\nPerson number " << (j+1);
        persPtr[j]->DisplayName();
    }

    for (j=0; j<n; j++)
        delete persPtr[j];
}
```

Memory Map

