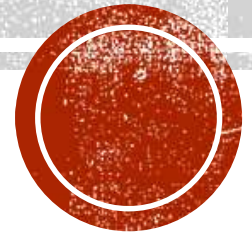


PROGRAMMING LANGUAGES 1

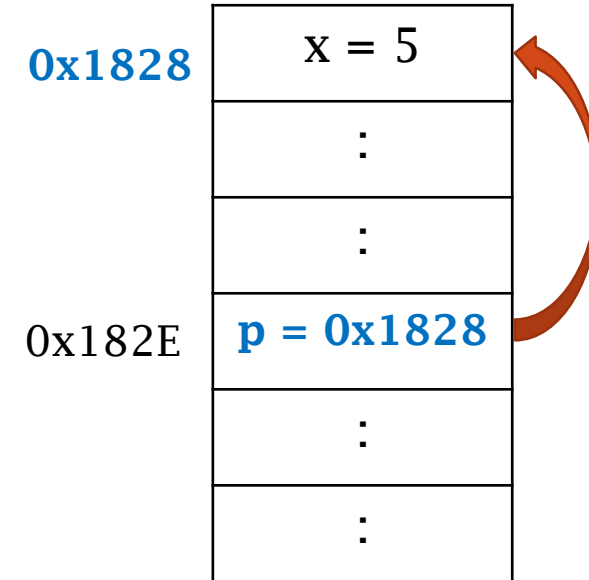
Pointers & References



POINTERS

A pointer holds the address of (*points to*) another variable

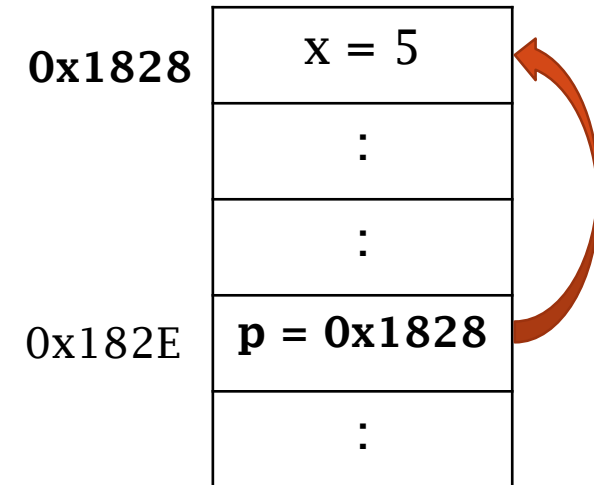
```
int x = 5;  
int* p; // define p as a pointer to int  
p = &x; // pointer p holds address of (&) x
```



POINTERS

```
int x = 5;  
int * p = &x;
```

```
cout << "Value of x = " << x << endl;  // 5  
cout<< "Address of x = " << &x << endl; // 0x1828  
cout<< "Value of p = " << p << endl;    // 0x1828 ← address of x  
cout<< "Value of *p = " << *p << endl;  // 5 ← Value of x  
cout<< "Address of p = " << &p << endl; // 0x182E  
*p = 6;    // x = 6  
cout << "Value of x = " << x << endl;  // 6
```



POINTER ARITHMETIC

- Operators +, - , ++, -- can be applied to pointers
- ++/-- operator moves pointer to next/previous address
- + and - operators can be used to access location relative to current address

POINTER ARITHMETIC - EXAMPLE

```
int x = 5, y = 3; // assume y follows x in memory
```

```
int * p = &x;
```

```
cout<< "Value of *p = " << *p << endl; // 5
```

```
p++;
```

```
cout<< "Value of *p = " << *p << endl; // 3
```

```
cout<< "Value of *(p-1) = " << *(p-1) << endl; // 5
```

0x1828

x = 5

y = 3

:

0x182E

p = 0x1828

:

:

DANGLING POINTER

- A dangling pointer is a pointer that has not been set/initialized
 - Has random value
 - Points to random location in memory

```
double * p;
```

NULL POINTER

- A null pointer has a value of zero, ie. does not point to any location

```
double * p = 0;
```

POINTERS & ARRAYS

- Arrays are actually implemented as pointers in C and C++
- Both notations can be used interchangeably

```
int a[5] = {0,1,2,3,4};  
int *p = a;  
cout << a[3] << endl;  
cout << *(p+3) << endl; // a[3]  
cout << *(a+3) << endl; // a[3]  
cout << p[3] << endl; // a[3]  
cout << *p << endl; // a[0]  
p++;  
cout << *p << endl; // a[1]  
cout << *a << endl; // a[0]
```

DYNAMIC ARRAYS

- Static (constant-size) arrays are defined as described before:

```
const int n = 5;  
int x[n]; // n must be constant
```

- Pointers are used to create **dynamic** (variable-size) arrays:

```
int n;  
cin >> n;  
int * x = new int[n];
```

- Dynamic arrays must be deleted by the programmer:

```
delete x;
```

DYNAMIC 2-D ARRAYS

- A 2-D array is a pointer to pointers
- A for-loop is used to create & delete dynamic 2-D arrays

```
int rows, cols;  
cin >> rows >> cols;  
int ** x = new int *[rows];  
for (int i = 0; i < rows; i++) x[i] = new int[cols];  
// Do something with array  
for(int i = 0; i < rows; i++) delete * x[i];  
delete [] x;
```

DYNAMIC 2-D ARRAY - EXAMPLE

```
int main()
{
    int rows, cols;
    cout << " Enter number of rows and columns"
    cin >> rows >> cols;
    int ** x = new int *[rows];
    for(int i = 0; i < rows; i++) {
        x[i] = new int[cols];
        for(int j = 0; j < cols; j++) {
            cin >> x[i][j]; // read element at row i and column j
        }
    }
}
```

DYNAMIC 2-D ARRAY - EXAMPLE (CONT'D)

```
// Do something with array data
for(int i = 0; i < rows; i++) {
    delete x[i];
}
delete [] x;
}
```

REFERENCES

A references is an alias (that *refers* to) another variable

```
int x;  
int & r = x;  // define reference r  
cout << "Value of x is " << x << endl;  // 5  
cout<< "Value of r is " << r << endl;    // 5  
cout << "Address of x = " << &x << endl;  // 0x1828  
cout << "Address of r = " << &r << endl;  // 0x1828  
r = 10;  
cout << "Value of x = " << x << endl;  // 10  
cout << "Value of r = " << r << endl;  // 10
```

REFERENCES

- Similar to pointers, except:
 - A reference must be initialized
 - A reference cannot refer to another variable once initialized
 - A reference cannot have a null (zero) value

LECTURE CODE

<https://github.com/ProgrammingLanguagesClass/Lecture-7>

SUMMARY

- Pointers
- Dynamic Arrays
- References