



Computer Programming Pointers 8th Lecture

엄현상 (Eom, Hyeonsang)
School of Computer
Science and Engineering
Seoul National University



Outline

- Pointer Basic
- Pointer Arithmetic
- Function Pointers
- String Basic
- String Functions
- Q&A

sizeof operator

- Returns size of operand in bytes (at compile-time)
- For arrays, sizeof returns
 - (size of an element) * (number of elements)

```
int myArray[ 10 ];  
cout << sizeof( myArray );
```

sizeof operator Cont'd

- Can be used with
 - Variable names
 - Type names
 - Constant values
- Parentheses are only required if the operand is a type name

Pointer Arithmetic

- Increment/decrement pointer (`++` or `--`)
- Add/subtract an integer to/from a pointer (`+` or `+=`, `-` or `-=`)
- Pointers may be subtracted from each other
- Pointer arithmetic is meaningless unless performed on a pointer to an array

Pointer Arithmetic Cont'd

- 5-element int array on a machine using 4-byte int

```
vPtr = &v[ 0 ];
```

- vPtr points to first element v[0], at location 3000

```
vPtr += 2;
```

- sets vPtr to 3008 (3000 + 2 * 4)
- vPtr points to v[2]

■ Subtracting pointers

- Returns number of elements between two addresses

```
vPtr2 = &v[ 2 ]; vPtr = &v[ 0 ]; vPtr2 - vPtr ?
```

Pointer Arithmetic Cont'd

- Pointer can be assigned to another pointer if both are of same type
 - If not, use cast operator
 - Pointer to void (**void ***)
 - Generic pointer, represents any type
 - No casting needed to convert pointer to **void ***
 - Casting is needed to convert **void *** to any other type
 - void pointers cannot be dereferenced

Pointer Arithmetic Cont'd

■ Pointer comparison

- Use equality and relational operators
- Compare addresses stored in pointers
 - Comparisons are meaningless unless pointers point to members of the same array
- When checking whether pointer is 0 (null pointer)

■ Arrays and pointers are closely related

- Array name is like constant pointer
- Pointers can do array subscripting operations

Pointer Arithmetic Cont'd

```
int b[] = { 10, 20, 30, 40 };    for( int j = 0; j < 4; j++ )
int *bPtr = b;                    cout << "bPtr[" << j;
...                                cout << "]=\n";
                                  cout << bPtr[ j ] << endl;

for( int i = 0; i < 4; i++ )      for( int f2 = 0; f2 < 4;
    cout << "b[" << i << "]=\n";    f2++ )
    cout << b[ i ] << endl;        cout << "*(bPtr + " << f2;
                                  cout << ") = \n";
                                  cout << *(bPtr+f2) << endl;

for( int f1 = 0; f1 < 4;        ...
    f1++ )
    cout << "*(b+" << f1;
    cout << ")=\n";
    cout << *(b+f1) << endl;
```

Arrays of Pointers

```
const char *a[ 4 ] =  
    { "Hearts", "Diamonds", "Clubs",  
      "Spades" };
```

- Each element of *a* points to a **char *** (string)
- Array *a* has fixed size (4), but strings can be of any size
- Commonly used with command-line arguments to function `main`

Pointers to Functions

- Contain addresses of functions
 - Function name is starting address of code that defines function
- Passed to functions
- Returned from functions
- Stored in arrays
- Assigned to other function pointers

Calling Functions using Pointers

- Function header

```
bool ( *foo ) ( int, int )
```

- Execute function from pointer with either

```
( *foo ) ( x, y )
```

- Dereference pointer to function, or

```
foo( x, y )
```

- Use the pointer directly

- Could be confusing

Function Pointers

```
void selectionSort( int [],  
    const int,  
    bool (*)( int, int ) );
```

```
void swap( int * const,  
    int * const );  
  
bool ascending( int, int );  
bool descending( int, int );
```

```
int main()  
{  
    const int aSize = 10;  
    int order;  
    int counter;
```

```
int a[ aSize ] =  
    { 2, 6, 4, 8, 10,  
      12, 89, 68, 45, 37 };
```

```
...  
cin >> order;  
if ( order == 1 ) {  
    selectionSort( a, aSize,  
                  ascending );  
}  
else  
{  
    selectionSort( a, aSize,  
                  descending );  
}  
...  
}
```

Function Pointers Cont'd

```
void selectionSort( int w[], const int size, bool
    (*compare)( int, int ) )
{
    int smallestOrLargest;

    for ( int i=0;
        i<size - 1; i++ )
    {
        sorl = i;

        for ( int idx = i + 1;
            idx < size;
            idx++ )
            if( !(*compare)
                ( w[sorl], work[idx] ) )
                sorl = idx;

        swap(&work[sorl], &work[i]);
    }
}
```

Function Pointers Cont'd

```
void swap( int * const element1Ptr, int * const element2Ptr )
{
    int hold = *element1Ptr;

    *element1Ptr = *element2Ptr;

    *element2Ptr = hold;
}
```

```
bool ascending( int a, int b )
{
    return a < b;
}
```

```
bool descending( int a, int b )
{
    return a > b;
}
```



Arrays of Pointers to Functions

- Menu-driven systems
 - Pointers to each function stored in array of pointers to functions
 - All functions must have same return type and same parameter types
 - Menu choice determines subscript into array of function pointers

Character Constant and String

- Integer value represented as character in single quotes
 - 'z' is integer value of z
 - 122 in ASCII
 - '\n' is integer value of newline
 - 10 in ASCII
- String
 - Series of characters treated as single unit
 - String literal (string constants)
 - "I like C++"
 - Static storage class
 - Array of characters, ends with null character '\0'
 - String is constant pointer to string's first character

String Assignment

- Character array

```
char color[] = "blue";
```

```
char color[] = { 'b', 'l', 'u', 'e',  
                '\0' };
```

- Creates 5 element char array color

- Last element is '\0'

- Variable of type **char ***

```
char *colorPtr = "blue";
```

- Creates pointer colorPtr to letter b in string "blue"

- "blue" resides somewhere in memory

Reading Strings

- Assign input to character array
`word[20]`

```
cin >> word;
```

- Reads characters until whitespace or EOF
- Reads only up to 19 characters (space reserved for `'\0'`)

- String could exceed array size

```
cin >> setw( 20 ) >> word;
```

cin.getline

- Read line of text

```
cin.getline( array, size,  
            delimiter );
```

- Copies input into specified array until either
 - One less than size is reached
 - Delimiter character is input

```
char sentence[ 80 ];
```

```
cin.getline( sentence, 80, '\n' );
```

<cstring> Library

- Manipulate string data
- Compare strings
- Search strings for characters and other strings
- Tokenize strings (separate strings into logical pieces)
- Data type **size_t**
 - An unsigned integral type
 - Such as unsigned int or unsigned long
 - Defined in header file <cstring>

String Functions

■ **char *strcpy(char *s1, const char *s2)**

- Copies second argument into first argument
 - First argument must be large enough to store string and terminating null character

■ **char *strncpy(char *s1, const char *s2, size_t n)**

- Specifies number of characters to be copied from second argument into first argument
 - Does not necessarily copy terminating null character

■ **char *strcat(char *s1, const char *s2)**

- Appends second argument to first argument
 - First character of second argument replaces null character terminating first argument
 - You must ensure first argument large enough to store concatenated result and null character

■ **char *strncat(char *s1, const char *s2, size_t n)**

- Appends specified number of characters from second argument to first argument
 - Appends terminating null character to result

■ **size_t strlen(const char *s)**

- Returns number of characters in string

String Functions Cont'd

- **int strcmp(const char *s1, const char *s2)**
 - Compares character by character
 - Returns
 - Zero if strings are equal
 - Negative value if first string is less than second string
 - Positive value if first string is greater than second string
- **int strncmp(const char *s1, const char *s2, size_t n)**
 - Compares up to specified number of characters
 - Stops if it reaches null character in one of arguments
- Character codes / character sets
 - Machine dependent
 - ASCII
 - “American Standard Code for Information Interchange”
 - EBCDIC
 - “Extended Binary Coded Decimal Interchange Code”
 - Unicode

Tokenizing

■ Breaking strings into tokens

- Tokens: logical units, such as words (separated by spaces)
- Separated by delimiting characters
- "This is my string"
 - 4 word tokens (separated by spaces)

■ **char *strtok(char *s1, const char *s2)**

- Multiple calls required
 - First call contains two arguments, string to be tokenized and string containing delimiting characters
 - Finds next delimiting character and replaces with null character
 - Subsequent calls continue tokenizing
 - Call with first argument NULL
 - Stores pointer to remaining string in a static variable
- Returns pointer to current token

String Example

...

```
#include <cstring> // prototype for strtok
using std::strtok;
```

```
int main()
{
    char sentence[] = "This is a sentence with 7 tokens";
    char *tokenPtr;
    tokenPtr = strtok( sentence, " " );
    while ( tokenPtr != NULL )
    {
        cout << tokenPtr << '\n';
        tokenPtr = strtok( NULL, " " );
    }
    cout << "\nAfter strtok, sentence = " << sentence << endl;
    return 0;
}
```