KAIST
EE 209: Introduction to Programming Systems
Pointer-Related Operators

Key

```
p, p1, p2      Pointer variables

i              An integral expression
```

Operators Meaningful for Any Pointer Variable

Dereference Operator

```
*p The contents of the memory referenced by p.
```

Equality and Inequality Relational Operators

```
p1 == p2 1 if p1 is equal to p2, and 0 otherwise.
p1 != p2 1 if p1 is unequal to p2, and 0 otherwise.
```

Assignment Operator

```
p1 = p2 Side effect: Assign p2 to p1. The new value of p1.
```

Operators Meaningful for Pointers that Reference Array Elements

Arithmetic Operators

```
p + i          The address of the ith element after the one referenced by p.
i + p          The address of the ith element after the one referenced by p.
p - i          The address of the ith element before the one referenced by p.
p++            Side effect: Increment p to point to the next element.
               The previous value of p.
++p            Side effect: Increment p to point to the next element.
               The new value of p.
p--            Side effect: Decrement p to point to the previous element.
               The previous value of p.
--p            Side effect: Decrement p to point to the previous element.
               The new value of p.
```

Arithmetic Operators

```
p1 - p2 The "span" of p1 and p2.
```

Relational Operators

```
p1 < p2 1 if p1 is less than p2, and 0 otherwise.
p1 <= p2 1 if p1 is less than or equal to p2, and 0 otherwise.
p1 > p2 1 if p1 is greater than p2, and 0 otherwise.
p1 >= p2 1 if p1 is greater than or equal to p2, and 0 otherwise.
```

Assignment Operators

```
p += i          Side effect: Increment p so its value is the address of
                the ith element after the one referenced by p.
                The new value of p.
p -= i          Side effect: Decrement p so its value is the address of
                the ith element before the one referenced by p.
                The new value of p.
```

Disallowed

```
p1 + p2
i - p
i += p
i -= p
p == i
```

Array Subscripting Operator

```
p[i]            *(p + i), that is, the contents of memory at the address
                that is i elements after the address referenced by p.
```

KAIST
EE 209: Programming Structures for EE
Kinds of Function Parameters rd

Kind of Parameter	Example	Implementation	C Construct
in	IntMath_gcd() (both params)	call by value	ordinary parameter
out	quorem() (3rd param) scanf() (2nd param)	call by reference	pointer parameter
inout	swap() (both params)	call by reference	pointer parameter

KAIST
EE 209: Programming Structures for EE
The "const" Keyword with Pointers

Pointer to Constant

```
1: const int i1 = 100;
2: const int i2 = 200;
3: const int *pi = &i1;          /* pi is a "pointer to a constant." */
4: i1 = 300;                     /* Error. Cannot change i1. */
5: i2 = 400;                     /* Error. Cannot change i2. */
6: pi = &i2;                     /* OK. */
7: *pi = 500;                    /* Error. Cannot change *pi. */
```

Constant Pointer

```
1: int i1 = 100;
2: int i2 = 200;
3: int *const pi = &i1;          /* pi is a "constant pointer." */
4: i1 = 300;                     /* OK. */
5: i2 = 400;                     /* OK. */
6: pi = &i2;                     /* Error. Cannot change pi. */
7: *pi = 500;                    /* OK. */
```

Constant Pointer to Constant

```
1: const int i1 = 100;
2: const int i2 = 200;
3: const int *const pi = &i1;    /* pi is a "constant pointer to a constant." */
4: i1 = 300;                     /* Error. Cannot change i1. */
5: i2 = 400;                     /* Error. Cannot change i2. */
6: pi = &i2;                     /* Error. Cannot change pi. */
7: *pi = 500;                    /* Error. Cannot change *pi. */
```

Disallowed Mismatch

```
1: const int i1 = 100;
2: const int i2 = 200;
3: int *pi = &i1;        /* Error. Subversive. Subsequently changing *pi would change i1. */
```

Disallowed Mismatch in Function Calls

```
1: void f(int *pi)
2: {
3: ...
4: }
...
5: const int i1 = 5;
6: const int *pi2 = &i1;
7: f(pi2);       /* Error. Subversive. If f() changes *pi, then *pi2 also would change. */
```

Allowed Mismatch

```
1: int i1 = 100;
2: int i2 = 200;
3: const int *pi = &i1;  /* OK, even though subsequently changing i1 would change *pi. */
4: i1 = 300;             /* OK. Also changes *pi. */
5: i2 = 400;             /* OK. */
6: pi = &i2;             /* OK, even though subsequently changing i2 would change *pi. */
7: *pi = 500;            /* Error. Cannot change *pi. */
```

Allowed Mismatch in Function Calls

```
1: void f(const int *pi)
2: {
3: ...
4: }
...
5: int i1 = 5;
6: int *pi2 = &i1;
7: f(pi2);               /* OK. *pi2 is protected against accidental change by f(). */
```

KAIST
EE 209: Programming Structures for EE
Manipulating C Strings

String Operation	String in Stack	String in Rodata Section

Allocating memory for a string	`{`
	`char acStr[5];`
	`...`
	`}`
		`{`
		`...`
		`...`
		`...`
		`}`

Initializing a string	`{`
	`char acStr1[3] = {'h', 'i', '\0'};`
	`char acStr2[] = {'h', 'i', '\0'};`
	`char acStr3[3] = "hi";`
	`char acStr4[] = "hi";`
	`char acStr5[2] = "hi"; /* truncation */`
	`char acStr6[10] = "hi";`
	`...`
	`}`
		`{`
		`...`
		`..."hi"...`
		`...`
		`}`

Computing the length of a string	`{`
	`char acStr[10] = "hello";`
	
	`... strlen(acStr) ...`
	`/* Evaluates to 5 */`
	
	`... sizeof(acStr) ...`
	`/* Evaluates to 10 */`
	`}`
		`{`
		`char *pcStr = "hello";`
		`... strlen(pcStr) ...`
		`/* Evaluates to 5 */`
		`... sizeof(pcStr) ...`
		`/* Evaluates to 4 */`
		`}`

Changing the characters of a string	`{`
	`char acStr[10] = "hi";`
	
	`acStr = "bye"; /* compiletime error */`
	
	`acStr[0] = 'b';`
	`acStr[1] = 'y';`
	`acStr[2] = 'e';`
	`acStr[3] = '\0';`
	
	`strcpy(acStr, "bye");`
	`/* Danger of memory corruption. */`
	`}`
		(Runtime error to attempt to change the characters of a string that resides in the rodata section)

Concatenating characters onto a string	`{`
	`char acStr[10] = "hi";`
	
	`acStr += "bye"; /* compiletime error */`
	
	`acStr[2] = 'b';`
	`acStr[3] = 'y';`
	`acStr[4] = 'e';`
	`acStr[5] = '\0';`
	`strcat(acStr, "bye");`
	`/* Danger of memory corruption. */`
	`}`
		(Runtime error to attempt to change the characters of a string that resides in the rodata section)

Comparing one string with another	{
	char acStr1[] = "hi";
	char acStr2[] = "bye";
	
	if (acStr1 < acStr2) ...
	/* Legal, but compares addresses!!! */
	
	if (strcmp(acStr1, acStr2) < 0) ...
	/* Compares strings */
	}
		(Same as string in stack)

Reading a string	{
	char acStr[10];
	
	iConvCount = scanf("%s", acStr);
	/* Reads a word as a string.
	Grave danger of memory corruption. */
	
	iRet = gets(acStr);
	/* Reads a line as a string,
	removing the \n character.
	Grave danger of memory corruption. */
	
	iRet = fgets(acStr, 10, stdin);
	/* Reads a line as a string,
	retaining the \n character. */
	}
		(Runtime error to attempt to change the characters of a string that resides in the rodata section)

Writing a string	{
	char acStr[] = "hi";
	
	iCharCount = printf("%s", acStr);
	/* Writes a string. */
	
	iSuccessful = puts(acStr);
	/* Writes a string, appending a \n
	character. */
	
	iSuccessful = fputs(acStr, stdout);
	/* Writes a string. */
	}
		(Same as string in stack)

Converting a string to another type	{
	char acStr[] = "123";
	int i;
	long l;
	double d;
	iConvCount = sscanf(acStr, "%d", &i);
	i = atoi(acStr);
	l = atol(acStr);
	d = atof(acStr);
	}
		(Same as string in stack)

Converting another type to a string	{
	char acStr[10];
	int i = 123;
	
	iCharCount = sprintf(acStr, "%d", i);
	/* Danger of memory corruption. */
	}
		(Runtime error to attempt to change the characters of a string that resides in the rodata section)