CSCI 2113 Lab 2

Bo Mei

Pointer

- Pointers are special variables for storing memory addresses. The address that the pointer points to.
- A pointer has an associated type.
  - int pointer, float pointer, char pointer
- *
  - Used in declaration: Declare a pointer
  - Used in other situations: Dereference. The value that the pointer points to.
- &
  - Get the address. The address of the variable.

Normal Variable vs Pointer

	Itself	*****		**&**
Normal variable *x*	The value of *x*	N/A		The address of *x*
Pointer *p*	The address that *p* points to	**Declaration**	**Other**	The address of *p*
		Declare *p*	The value that *p* points to	

Example (Lecture 2 Worksheet)

```
int i;
int j;
int *p;
```

The value of i	
The address that p points to	
The address of j	
The address of p	
The value that p points to	

Example (Lecture 2 Worksheet)

```
int i;
int j;
int *p;
```

The value of i	i
The address that p points to	p
The address of j	&j
The address of p	&p
The value that p points to	*p

Example (Lecture 2 Worksheet)

```
int main() {
    int a = 10;
    int *ptr;
    ptr = &a;
    *ptr = 20;
    return 0;
}
```

Stack		
Address	**Name**	**Contents/Value**
10000		
10004		

Example (Lecture 2 Worksheet)

```
int main() {
    int a = 10;
    int *ptr;
    ptr = &a;
    *ptr = 20;
    return 0;
}
```

Stack		
Address	**Name**	**Contents/Value**
10000	a	20
10004	ptr	10000

Attendance Quiz (Under the “Tests” category on Blackboard)

```
int i = 100;
int j;
int *ptr;

ptr = &i;
j = *ptr;
*ptr = 200;

printf(“i=%d, j=%d, *ptr=%d\n”, i, j, *ptr);
```

Magic Pointers

- Can easily work with memory address and the value stored in the memory address.
- Can both set and get values.
- Can be used for passing references as function arguments.

```
void moveNE(int *a, int *b) {
    *a = *a + 1;
    (*b)++;
}
int main() {
    // ...
    moveNE(&x, &y);
    // ...
}
```

Exercise 2

- Relational Operation
  - <, <=, >, >=, ==, !=
  - For results, return 1 when the operation is true, and return 0 when the operation is false
  - 3 < 1
  - 1 != 2

- Logical Operation
  - !, &&, ||
  - For operands, non-0 is true and 0 is false.
  - For results, return 1 when the operation is true, and return 0 when the operation is false
  - 3 < 1 && 1 != 2
  - !(3 < 1)
  - 3 && 1
  - !3

Exercise 2

- if(expression), while(expression)
  - Expression can be a relational expression, a logical expression, **or a value**
  - For expression: non-0 is true and 0 is false
  - if(3 < 1), if(1 != 2)
  - if(3 < 1 && 1 != 2), if(!(3 < 1))
  - **if(3), if(1 << 3), if(10 / 2)**
- Pay attention to the precedence of operators in C
  - http://en.cppreference.com/w/c/language/operator_precedence

Exercise 2

- Integer division in C (/)
  - Rounded down
  - 10 / 6 = 1, 13 / 5 = 2, 5 / 10 = 0
- Remainder operation in C (%)
  - Can only be used between two integers
  - 10 % 6 = 4, 13 % 5 = 3, 5 % 10 = 5

- Increment/Decrement operation
  - i++
  - First, execute the statement without "++/--" operation
  - Then, i = i + 1
  - i = 5; a[i++] = 2;

  - ++i
  - First, i = i + 1
  - Then, execute the statement without "++/--" operation
  - i = 5; a[++i] = 2;