1. C Review

- Storage classes and scopes
- Tips for C programming
- Review of pointers

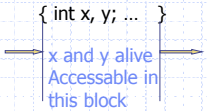
Declaration:

- int v = 0; /* declared at some place in your program */
- What can you see from the declaration?

storage class	:	Scope and life time: auto, register, static, extern
type	:	value domain
value	:	current value ∈ value domain
name	:	symbolic identifier
location	:	memory address
size	:	how many bytes it occupies

Storage class:

- Auto: declared inside a block, exists only when the block is entered, and disappears when execution leaves the block.



- Static: accessible in the block where it is declared, exists and retains its value in whole program cycle.

```
void f(){
        int x = 0;
        printf("%d\n", x++);
}

int main(){
        f(); // 0
        f(); // 0
        f(); // 0
}
```

```
void f(){
        static int x = 0;
        printf("%d\n", x++);
}

int main(){
        f(); // 0
        f(); // 1
        f(); // 2
}
```


- Extern: accessible and exists in whole program file and program cycle. Define a global variable exactly once and use external declarations everywhere else.

```
/* f1.c */

int x;

void f(){
        x++;
}

void g(){
        x++;
}
```

```
/* f2.c */

extern int x;

void h(){
        x++;
}
```


- Register: frequently used variables for efficiency purpose. Restrictions:
  - (1) can not take the address of a register variable,
  - (2) can not declare global register variables,
  - (3) a register variable must fit into a single machine word,
  - (4) the compiler may ignore register declaration.

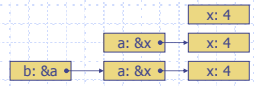
```
/* search the given table to find the given key;
   return the index if found or –1 otherwise      */

int table_search(int a[], register int n, register int key){
        register int j;
        for (j = 0; j < n && a[j] != key; j++);
        return (j != n) ? j : -1;
}
```

Tips for C Programming:

- Do not change a loop variable inside a for loop block.
- All flow control primitives (if, else, while, for, do, switch, and case) should be followed by a block, even if it is empty.
- Statements following a case label should be terminated by a statement that exits the switch statement.
- All switch statements should have a default case.
- Use break and continue instead of goto.
- Do not have overly complex functions.
- Indent to show program structure (better readability).
- Parenthesize to resolve ambiguity.

Review of pointers:

```
int x = 4;
int *a = &x;
int **b = &a;
```



x	4	a	addr(x)	b	addr(a)
&x	addr(x)	&a	addr(a)	&b	addr(b)
*x	illegal	*a	4	*b	addr(x)
*(&x)	4	**a	illegal	**b	4
				***b	illegal

*b == a == &x
**b == *a == x

The unary operators * and &

```
int x, y, *px;

x = 10;
px = &x;
y = *px;
```

- The unary operator & gives the address of an object.
- It can be used only to variables and array elements. &(y+2) and &7 are illegal.
- The unary operator * treats its operand as the address of a memory cell, and accesses the cell to get the contents.

The unary operators * and &

```
int x, y;
int *px;
x = 10;
px = &x;
*px += 1;
(*px)++;
```

- Declaration int *px; means that *px is an int, or px is a memory cell containing a pointer to a variable of int.
- *px can be on the left side of an assignment.
- In (*px)++, the parentheses are required.

Function Arguments and Pointers

```
int x, y;
 int *px;
 scanf("%d", &x);
 scanf("%d", px);
```

- In C, invocation of functions is "call by value".
- In order for the called function to change the value of a variable in the calling function, we pass the address of the variable.

Pointers and Arrays

```
int x[5] = {12, 23, 34, 45, 56};
 int *px;
 int y, z;
 px = &x[0];        //Set px to point to x[0].
 y = *px;           //Assign the content of x[0] to y.
 px = x;            //Set px to point to x[0], which is the
                    //beginning of x.
 z = *(px+1);       // same as z = x[1].
```

- Note: x = px or x++ or px = &x are illegal.

Address Arithmetic

```
int x[5] = {12, 23, 34, 45, 56};
int *px;
int y, z;
px = &x[0];        // Set px to point to x[0].
z = *(px+1);       // Same as z = x[1].
y = *(px+4);       // Same as y = x[4];
```

- When p is a pointer to an array, p+1 points to the second element, p+2 points to the third element, …
- For p+n, the compiler scales n to the size of the object p points to.
- For p and q to elements of the same array, they can be compared using ==, <, <=, >, >=, !=.

Character pointers

```
char *text0, *text1;
text0 = "Hello world!"; //Assign to text0 a pointer to the string.
text1 = text0;          //Assign the same pointer to text1.
```

- text0+n points to the (n-1)th character in the string.
- *text0++ is the character text0 points to before it is incremented.
- *++text0 is the character text0 points to after it is incremented.

Relationship of -> and . :

```
typedef struct {
                char name[20];
                int grade;
        } student;

student s;
student *sp = &s;

s.grade = 97;          // direct structure field selection
sp -> grade = 97;      // indirect structure field selection
(*sp).grade = 97;      // (*sp).grade == sp->grade
(&s) -> grade = 97;    // s.grade == (&s)->grade
```


Structures and Operators:

- sizeof() operator determines the number of bytes used by a structure.

  sizeof(s) // using a student variable

  sizeof(student) // or using a type
- Don't assume the size of a structure is the sum of the size of its fields.

  sizeof(s.name) + sizeof(s.grade)

  ≠ sizeof(s)

Structures and Operators:

- Assignment operator = applies to structures. It copies the contents of one structure into another.

  student new_student = s;
- No operators for comparing structures.

  (new_student == s) // wrong