C Programming & Process (3th Lab)

Variables & Constants

Variable

- Memory location which can store any data (changes in value) during program execution (expressable number of cases with 8 coin: coin=bit, number of cases=ranges)
- C Basic Data Types

<table>
<thead>
<tr>
<th>Types</th>
<th>Bytes (Bits)</th>
<th>Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer Types</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Char</td>
<td>1 (8)</td>
<td>-128~127</td>
</tr>
<tr>
<td>unsigned char</td>
<td>1 (8)</td>
<td>0~255</td>
</tr>
<tr>
<td>Short</td>
<td>2 (16)</td>
<td>-32,768~32,767</td>
</tr>
<tr>
<td>unsigned short</td>
<td>2 (16)</td>
<td>0~65,535</td>
</tr>
<tr>
<td>Int</td>
<td>4 (32)</td>
<td>-2,147,483,648~2,147,483,647</td>
</tr>
<tr>
<td>unsigned int</td>
<td>4 (32)</td>
<td>0~4,294,967,296</td>
</tr>
<tr>
<td>Long</td>
<td>4 (32)</td>
<td>-2,147,483,648~2,147,483,647</td>
</tr>
<tr>
<td>unsigned long</td>
<td>4 (32)</td>
<td>0~4,294,967,296</td>
</tr>
<tr>
<td>Floating Type</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Float</td>
<td>4 (32)</td>
<td>1.175494351 E – 38</td>
</tr>
<tr>
<td></td>
<td></td>
<td>~ 3.402823466 E + 38</td>
</tr>
<tr>
<td>Double</td>
<td>8 (64)</td>
<td>2.2250738585072014 E – 308</td>
</tr>
<tr>
<td></td>
<td></td>
<td>~ 1.7976931348623158 E + 308</td>
</tr>
<tr>
<td>long double</td>
<td>12 (96)</td>
<td>3.4 E - 4932 ~ 3.4 E + 4932</td>
</tr>
</tbody>
</table>

printf() and scanf()

printf

- A command that prints out the desired data to a specific format through monitor
- printf("this year is %d.", 2014);

<table>
<thead>
<tr>
<th>Format</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>%c</td>
<td>character (문자)</td>
<td>printf(&quot;%c&quot;, 'a');</td>
</tr>
<tr>
<td>%d</td>
<td>decimal integer (10 진수)</td>
<td>printf(&quot;%d&quot;, 100);</td>
</tr>
<tr>
<td>%e</td>
<td>floating point number in scientific notation</td>
<td>printf(&quot;%e&quot;, 1.234);</td>
</tr>
<tr>
<td>%f</td>
<td>floating point number</td>
<td>printf(&quot;%f&quot;, 1.234);</td>
</tr>
<tr>
<td>%g</td>
<td>e-format 또는 f-format</td>
<td>printf(&quot;%g&quot;, 1.234);</td>
</tr>
<tr>
<td>Format</td>
<td>Meaning</td>
<td>Example</td>
</tr>
<tr>
<td>-------</td>
<td>------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>%c</td>
<td>character (문자)</td>
<td>scanf(&quot;%c&quot;, &amp;a);</td>
</tr>
<tr>
<td>%d</td>
<td>decimal integer (10 진수)</td>
<td>scanf(&quot;%d&quot;, &amp;a);</td>
</tr>
<tr>
<td>%f</td>
<td>floating point number (float)</td>
<td>scanf(&quot;%f&quot;, &amp;a);</td>
</tr>
<tr>
<td>%lf</td>
<td>floating point number (double)</td>
<td>scanf(&quot;%lf&quot;, &amp;a);</td>
</tr>
<tr>
<td>%Lf</td>
<td>floating point number</td>
<td>scanf(&quot;%Lf&quot;, &amp;a);</td>
</tr>
<tr>
<td></td>
<td>(long double)</td>
<td></td>
</tr>
<tr>
<td>%s</td>
<td>string (문자열)</td>
<td>scanf(&quot;%s&quot;, &amp;a);</td>
</tr>
</tbody>
</table>

- **scanf**
  - A command that input the data which is entered by user into specified variable.

  ```c
  scanf("format string..", argument list);
  ```

  ```c
  printf("enter your age:"); scanf("%d", &variable);
  ```

- **typedef**
  - A keyword that makes a kind of nickname to data type.

  ```c
  typedef int color
  color red, blue, green;
  → specify color type to int type.
  ```

  ```c
  typedef char uppercase
  typedef int INCHES, FEET
  typedef unsigned long size_t /* found in stddef.h */
  uppercase u;// same as char type
  INCHES length, width;// same as int type
  ```

- **The purpose of typedef**
  - To abbreviate the long declaration
  - To improve readability with determining the type name for purpose of use (if you alias the int type to color type with typedef, you can easily know this data type represents color.)
Arrays

- Data type that represents multiple values with variable of same name.

```plaintext
[Ex] int grade0, grade1, grade2, ..., grade49;
⇒ int grade[50];
```

- Syntax

```
    element-type array_name[size];
```

  - Should write the array size with positive
  - The index of array element always starts with zero
  - Above example generates the grade[0], grade[1], ..., grade[49].

- Example of array declaration

```plaintext
[Ex] int digit[10] = {0};
[Ex] #define MAX 100;
...

double student[MAX];
[Ex] float exp[13]; int a[10]; a[2] = 8;
    exp[10] = exp[11];
    exp[2+3] = 100;
    exp[a[2]] = 100;
```

- Initialization

  - assign the initial value to the array.

```plaintext
[Ex] float x[7] = { -1.1, 0.2, 33.0, 4.4, 5.05, 0.0, 7.7 };    
⇒ initialize x[0] = -1.1, x[1] = 0.2, ..., x[6] = 7.7
```

  - when initial value is smaller than the value of array element

```plaintext
[Ex] int a[100] = { 0 };    
⇒ a[0] = 0, a[1] = 0, ..., a[99] = 0 initialize all the element to zero
```

  - when the array size is not specified

```plaintext
[Ex] int a[] = { 2, 3, 5, 7 };    
```
int a[4] = { 2, 3, 5, 7 };  
→ initial value become the array size; above two example is equal

- Approach to use array
  - Note when using array

[Ex] int a[5] = { 1, 2, 3, 4, 5 };  
printf("%d", a[5]); //a[4] is end of array
→ assign from a[0] to a[4], it is likely to use a[5]

- Multi-Dimension Arrays

| Array Dimensions |
|-------------------|------------------|
| **Declarations of Arrays** | **Remarks** |
| int a[100]; | a one-dimensioanl array |
| int b[2][7]; | a two-dimensioanl array |
| int c[5][3][2]; | a three-dimensioanl array |

[Ex] two dimension array

|---------|---------|---------|---------|

- Structures
  - The difference between array and structure
    - Array
      - The element of array should be same type.
      - Access each element with index
    - Structure
      - Can be composed with the elements of different type
      - Each element has its name
      - Access each element with its name
struct Declaration

◆ Collection of Members

[Ex] struct {/* structure composed of 3 elements */
    int number;
    char name [20] ;
    int on_hand;
} part1;

Structure Tag

◆ Name to specify particular structure

[Ex] struct part {
    int number;
    char name [20] ;
    int on_hand;
};

[Ex] struct part {
    int number;
    char name [20] ;
    int on_hand;
} part1, part2; //value of part struct: part1,part2
struct part part3;

◆ Initializing Struct Variables

[Ex] struct student {
    int number;
    char name [20] ;
    int score;
} part1 = {1, "handong", 95 }, part2; //initialize part1 only

◆ Compatible Structure

◆ If the type of structure variable is same, they can mutually assign to each other
[Ex] part1 = part2; // part1 and part2 should have the same structure
→ Each member value in part2 is copied to each member value of part1

- If it is not compatible type, =, ==, != are impossible.

## Accessing Structure Members
- Struct Member Operator '.'
  - To access each member of structure, use '.'.

[Ex] struct {
    int number;
    char name[20];
    int on_hand;
} part1 = {1, "Monitor", 10};
printf ("Part number : %d \n", part1.number);
printf ("Part name : %s \n", part1.name);
printf ("Quantity on hand : %d \n", part1.on_hand);

## Strings

### Strings in C Programming
- String : consists of a series of characters(char type variable)
- In C, String is represented by One-Dimensional char array.
- Each element in string is accessible by Array Index or Pointer.
- In C, there are various build-in functions for string process
- Null Character ‘\0’ is considered by Terminate String(End of String)

#### End-of-String Sentinel ‘\0’
- All strings are ended with ‘\0'(null-String)
- String secure the memory through char Array

[Ex] char word[100]; // save a string to the array which consists of 100s of 1byte.

- The way to store a string to secured memory

[Ex] word[0] = 'a'
word[1] = 'b'
word[2] = 'c'
word[3] = '\0' // insert null char for termination of string
Initialization of Strings

[Ex] char word[4] = "abc"; // word[0] is a, word[1] is b, word[2] is c, word[3] is NULL
char word[4] = {'a', 'b', 'c', '0'};
char word[] = {'a', 'b', 'c', '0'};
char word[] = "abc"; // by compiler, create the array for 4 char automatically

String-Handling Functions in <string.h>

- char *strcat(char *s1, const char *s2);
  - At the end of string array s1, append string array s2 // s1 + s2
  - s1 should retain enough memory to store string array.

```
“abcd” + “efg”
```

- char *strcpy(char *s1, const char *s2);
  - copy string s2 to string s1 // s1 = s2
  - s1 should retain enough memory to store string array.

```
[Ex] char str1[100] = "Seoul National", str2[100] = "University";
strcat(str1, str2);
printf("%s
", str1); // Seoul NationalUniversity
strcpy(str1, str2);
printf("%s
", str1); // University
```

- int strcmp(const char *s1, const char *s2);
  - compare ASCII code the value of each ‘char’ that consists of string one by one
  - return value < 0 : The ASCII value of s1 is smaller than that of s2
  - return value = 0 : The ASCII value of s1 is equal to that of s2
  - return value > 0 : The ASCII value of s1 is bigger than that of s2

- size_t strlen(const char *s1);
  - return the length of String
[Ex] char str1[100] = "kwanak", str2[100] = "kwanak"
if(str1 == str2) // whether the address of char array which is pointed by str1, str2 are equal or not.
printf("same address \n");
if(strcmp(str1, str2) == 0) // whether the data of str1 and str2 is equal or not.
printf("equal \n");

◆ char* strchr(const char *s1, char c1);
   ● Return the pointer of char c1 that appear first in string s1
   ● If there is no character to represent c1, return the Null Pointer

◆ char* strstr(const char *s1, char* s2);
   ● similar to strchr, but search sub-string instead of character
   ● if sub-string doesn’t exist, return the Null Pointer

[Ex] if(strchr("hello", 'e' )) // if address of hello is 100, return 104
   printf("e is in hello");
   if strstr("hi, there", "hi")
      printf("found");

• Flows of Control

  □ Control Blocks

<table>
<thead>
<tr>
<th>Statement</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>if... else if... else</td>
<td></td>
</tr>
<tr>
<td>if (expression) {</td>
<td></td>
</tr>
<tr>
<td>statement1;</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>else if (expression) {</td>
<td></td>
</tr>
<tr>
<td>statement1;</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>else {</td>
<td></td>
</tr>
<tr>
<td>statement1;</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>
### switch

```
switch ( expression ) {
    case constant-expression : statements
    case constant-expression : statements
        ..................
    default : statements
}
```

### while

```
while (expr) {
    statements;
}
next statement;
```

### for

```
for ( expr1; expr2; expr3 ) {
    statements;
}
next statement;
```

### do... while

```
do {
    statements;
} while(expression);
next statement;
```
Process

- Program – executable file that is stored in disk
- Process – Program in execution

{ 연상! 집짓기 : 지을 집의 설계도(디스크 안에 있는 program), 집을 짓는 사람들(CPU), 집을 짓기 위한 재료나 도구들(resource : register, memory address, I/O device..) 전체 집을 짓는 과정 (process) }

◆ In order to execute one command, shell creates new process whenever program starts.

<table>
<thead>
<tr>
<th>Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fork</td>
<td>Create the same process like caller process</td>
</tr>
<tr>
<td>Exec</td>
<td>Replace the memory space of a process with another new program</td>
</tr>
<tr>
<td>Wait</td>
<td>Provide process synchronization. Wait until associated another process terminates.</td>
</tr>
<tr>
<td>Exit</td>
<td>Terminates process</td>
</tr>
</tbody>
</table>

fork system call

- function
- basic process creation function
- if executed successfully, create caller process(parent process) and the same new process(child process)
반환 값 (return value)

正常执行
- Execution (parent) Process: return the created (child) process id
- created (child) Process: return 0

异常执行: return the minus value

Example) fork_test.c

```c
#include <sys/types.h>
#include <unistd.h>

int main()
{
    pid_t pid; /* 부모에서 프로세스 식별번호 저장 */
    printf("Calling fork \n");
    pid = fork(); /* 새로운 프로세스 생성 */
    if (pid == 0)
        printf("I'm the child process\n");
    else if (pid > 0)
        printf("I'm the parent process\n");
    else
        printf("fork failed\n");
}
```
$gcc –o fork_test fork_test.c
$./fork_test
I'm the child process
I'm the parent process

■ exit system call
  ◆ function
  - Process termination

#include <stdlib.h>

void exit (int status);

- status: specify the termination status // delivery the status value to parent process
  - return value: normal: 0, otherwise: non zero

Example) exit_test.c
#include <stdlib.h>

main()
{
  int exit_status;

  printf("enter exit status: ");
  scanf("%d", &exit_status);
  exit(exit_status);
}
```bash
$gcc -o exit exit.c
./exit
Enter exit status: 0
$echo $? // right before terminated process save the configured status to $?
0
```

**wait** system call

- wait until the execution of child process ends.

```c
#include <sys/types.h>
#include <sys/wait.h>

pid_t wait(int *status);
```

- **status**: when the wait is returned, useful status information (exit status)
- Mostly, parent process make the child process with fork and then call wait.

```c
Example) wait_test.c
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>

main()
{
    pid_t pid;
    int status, exit_status;

    if ((pid = fork()) < 0)
        perror("fork failed");
    if (pid == 0) {
        sleep(4); /* 수행을 4초 동안 중단 */
        exit(5);
    }

    /* 부모 코드: 자식이 퇴장(exit)할 때까지 대기 */
    if ((pid = wait(&status)) == -1) {
        perror("wait failed");
        exit(2);
    }
}
```
$gcc -o wait_test wait_test.c  
$./wait_test  
....//4 second sleep  
$

**execl function**

- function
  - the execution of new program start

```c
#include <unistd.h>
int execl(const char *path, const char *arg0,...,const char *argn,(char *) 0);
```

- behavior
  - load the new program onto the memory space of caller process
  - if normally executed, it is not returned to execution process (it's different from fork())

Example) execl_test.c
```c
#include <sys/types.h>
#include <unistd.h>

main()
{
    pid_t pid;

    pid = fork();
    if (pid == 0) {
        /* 자식 프로세스가 execl 호출 */
        execl("/bin/ls", "ls", "-l", (char *)0);  
        perror("execl failed");
    } else if (pid > 0) {
        /* 자식이 끝날 때까지 수행을 일시 중단하기 위해 wait 호출 */
        wait((int *)0);
        printf("ls completed\n");
        exit(0);
    } else
        perror("fork failed");

}$gcc -o execl_test execl_test.c  
$./execl_test  
합계 104  
....//ls -l execution  
....  
is completed
Practice

- Following source code is the program which calculates factorial. In this program, child process calls the fac() that calculates the factorial, and parent process waits until the computation of child process ends, and then child calls exit(), so that takes out the results of fac(), stored in status and print it out. Fill the blank and complete the source code, and identify the result of fac() by using I/O Redirection instruction.

```c
/*factorial_child_exit.c*/
#include <sys/types.h>
#include <sys/wait.h>
int fac(int oper){
    if(oper==0) return 1;
    return fac(oper-1)*oper;
}
int main(int argc, char * argv[])
{
    pid_t pid;
    int status,exit_status;
    if((pid=fork())<0)
        perror("fork failed");
    if(pid==0){
        printf("child is computing factorial\n");
        /*black code*/
    }
    if((pid=wait(&status))==-1){
```
```c
perror("wait failed");
exit(2);
}
printf("parent received the result\n");
if(WIFEXITED(status)){ //if child process is returned normally, return true
    exit_status=WEXITSTATUS(status); //Identify the status that is a parameter of wait
    printf("%d is computed \n",exit_status);
}
return 0;
}
```

```
gcc –o test factorial_child_exit.c
./test 5
child is computing factorial
parent received the result
120 is computed
./test 3
child is computing factorial
parent received the result
6 is computed
```