- 문항 선택

1 2 3 4
5 6 7 8

한 문제씩 검토
검토 완료

강의실 홈

강의정보
- 교수계획표 (국문)
- 교수계획표 (영문)

성적/출석관리
- 동영상이수현황
- 스마트출석부
- 성적부

수강생 알림
- 쪽지 보내기

기타 관리

학습활동

/ 프로그래밍원리와실습 (CB1600702-003) / [Q7:8]

시작 일시	2022-11-01 17:36
진행 상황	종료됨
완료 일시	2022-11-07 13:25
소요시간	5 일 19 시간
성적	최고 8.00점 중 8.00점 (100%)

문제 **1**
정답
총 1.00 점에서 1.00 점 할당
문제 표시

Fill in the blanks with an appropriate word.

- In terms of scope, variables can be classified into two types.
  - In the case of a variable declared in a function area, it is valid within the function and is called a [1] local ✓ variable
  - If a variable is declared outside the function, it is valid in the entire function of the source file and is called a [2] global ✓ variable
- In terms of lifetime/extent, variables can be classified into two types.
  - Variables are created and allocated in memory when the program is started and remain until the end of the program. All [2] variables are [3] static ✓ variables. Variables added with the keyword "[3]" are also [3] variables.
  - Variables are created and allocated in memory when a function is called and disappear when the function exits. Without the [3], all [1] variables are [4] automatic ✓ variables.

문제 **2**
정답
총 1.00 점에서 1.00 점 할당
문제 표시

Variables have attributes. Fill in the following blanks with an appropriate word respectively.

- [1] Identifier ✓ : A name for a variable that is used to access the variable in your code
- [2] Value ✓ : A variable stores [2]. You can read or change the [2] of the variable in the code.
- [3] Type ✓ : A variable has a "data [3]" that determines how to interpret the binary data stored in memory.
- [4] Memory Size ✓ : A variable has a certain amount of memory space to store the [2]. The amount of memory space required depends on the [3] of data. You can use the sizeof() operator to determine it.
- [5] Scope ✓ : Valid ranges of variable names in source code.
- [6] Lifetime ✓ : There is a period in which a variable is maintained by occupying a certain amount of memory space.
- [7] Address ✓ : A variable has a/an [7] for its memory space. It tells the location of the memory space for the variable in the memory cells. It is related to the concept of "pointer".

문제 **3**
정답
총 1.00 점에서 1.00 점 할당
문제 표시

Compare the following 2 programs.

Write down the output of each program.

Execution Result-Program1	Execution Result-Program2
The sum is 5 ✓	The sum is 15 ✓

Program 1

```
#include <stdio.h>
int add(int n) {
        int sum = 0;
        sum += n;
        return sum;
}

int main(void) {
        int ns[] = {1,2,3,4,5,-1};
        int i=0, sum;

        while (ns[i] > 0) {
                sum = add(ns[i++]);
        }
        printf("The sum is %d\n", sum);

        return 0;
}
```

Program 2

```
#include <stdio.h>
int add(int n) {
        static int sum = 0;
        sum += n;
        return sum;
}

int main(void) {
        int ns[] = {1,2,3,4,5,-1};
        int i=0, sum;

        while (ns[i] > 0) {
                sum = add(ns[i++]);
        }
        printf("The sum is %d\n", sum);

        return 0;
}
```

문제 **4**
정답
총 1.00 점에서 1.00 점 할당
문제 표시

The basic function call mechanism of the C language is Call by Value ✓ .

In this mechanism, the value of the variable/expression used in the function call is copied into the corresponding argument variable of the function. Generally, the change in the argument variable does not change the value of the variable/expression used in the function call.

Write the output of the following program.

Execution Result

ns [5 ✓ , 5 ✓ , 5 ✓]
Sum of ns:6

```
#include <stdio.h>
#define END_MARK                -1
int sum_int_array(int nums[]) {
        int i=0, sum=0;
        while (nums[i] != END_MARK) {
                sum += nums[i];
                nums[i++] = 5;
        }
        return sum;
}
int main(void) {
        int ns[4] = {1,2,3,END_MARK};

        int sum = sum_int_array(ns);

        printf("ns [%d, %d, %d]\n",
                ns[0],ns[1],ns[2]);

        printf("Sum of ns:%d\n", sum);
        return 0;
}
```

문제 **5**
정답
총 1.00 점에서 1.00 점 할당
문제 표시

Refer to a reference for the following C standard library functions declared in <math.h>

ceil(), floor(), round(), trunc()

What is the output of the following program?

FYI, The FSF **"%.nlf"** means that print out a double with *n* digits after ".". For example, **"%.2lf"** specifies the number of digits after "." is 2. If the value to be printed is 1.200000, the output will be 1.20.

Execution Result

ceil(1.80)= 2 ✓ , ceil(1.20)= 2 ✓ , ceil(-1.20)= -1 ✓
floor(1.80)= 1 ✓ , floor(1.20)= 1 ✓ , floor(-1.20)= -2 ✓
round(1.80)= 2 ✓ , round(1.20)= 1 ✓ , round(-1.20)= -1 ✓
trunc(1.80)= 1 ✓ , trunc(1.20)= 1 ✓ , trunc(-1.20)= -1 ✓

```
#include <stdio.h>
#include <math.h>
int main(void) {
        double x = 9.0/5;
        double y = 6.0/5;
        double z = -6.0/5;

        printf("ceil(%.2lf)=%.0lf, ceil(%.2lf)=%.0lf, ceil(%.2lf)=%.0lf\n",
                        x,ceil(x),y,ceil(y),z,ceil(z));
        printf("floor(%.2lf)=%.0lf, floor(%.2lf)=%.0lf, floor(%.2lf)=%.0lf\n",
                        x,floor(x),y,floor(y),z,floor(z));
        printf("round(%.2lf)=%.0lf, round(%.2lf)=%.0lf, round(%.2lf)=%.0lf\n",
                        x,round(x),y,round(y),z,round(z));
        printf("trunc(%.2lf)=%.0lf, trunc(%.2lf)=%.0lf, trunc(%.2lf)=%.0lf\n",
                        x,trunc(x),y,trunc(y),z,trunc(z));

        return 0;
}
```

문제 **6**
정답
총 1.00 점에서 1.00 점 할당
문제 표시

We want to write a function, dice(), which simulates a random dice throw using the rand() function. That is, whenever it is called, the dice() returns a random values in [1,2,3,4,5,6];

What is a proper code for the [A]?

○ rand() ○ rand()%6 ● rand()%6+1 ✓ ○ rand()/RAND_MAX ○ rand()/(RAND_MAX+1)

총 1.00 점에서 1.00 점 할당

정답 : rand()%6+1

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define N_DICE 10
int dice() {
        return ([A]);
}

int main(void) {
        int i, diceval;
        srand(time(0));
        for (i=0; i<N_DICE; i=i+1) {
                diceval = dice();
                printf("%dth Dice Value : %d\n", i, diceval);
        }
        return 0;
}
```

문제 **7**
정답
총 1.00 점에서 1.00 점 할당
문제 표시

In the lecture, some essential programming guidelines are explained.

Fill in the blanks with an appropriate word respectively.

- Use good indentation ✓ . Bodies of functions, loops, if-else statements, etc. should be indented, and statements within the same body-level should be indented the same amount.
- Write short ✓ functions. Because it is easier to write, read, and test.
- Use descriptive ✓ names for variables and functions.

문제 **8**
정답
총 1.00 점에서 1.00 점 할당
문제 표시

There are some practices for writing compound words or phrases. Among them, snake_case in which each word is separated by an underscore, and CamelCase in which each word begins with an uppercase letter are usually used in the programming. The snake_case is usually used in C programming.

The CamelCase is used in the following source code. What the names would be if the snake_case is applied?

Write an appropriate snake_case name for each following CamelCase name.

- getDiscountRate → get_discount_rate ✓
- discountRate → discount_rate ✓
- isSpring → is_spring ✓
- isSummer → is_summer ✓
- isFall → is_fall ✓

```
float getDiscountRate(int month) {
        float discountRate = 0.0F;
        int isSpring = month >= 3 && month <= 5 ;
        int isSummer = month >= 6 && month <= 8 ;
        int isFall = month >= 9 && month <= 11 ;
        if ( isSpring )
                discountRate = 0.2F;
        else if ( isSummer )
                discountRate = 0.5F;
        else if ( isFall )
                discountRate = 0.2F;
        else
                discountRate = 0.1F;
        return discountRate;
}
```

검토 완료