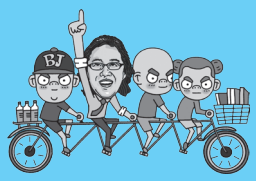
수험생의 마음으로 만든 책! 시나공 시리즈

2020 시나공

시험에 나오는 것만 공부한다!



2020년 2회 정보처리기사 실기 시험 100% 합격전략집

1 일차 | 2 일차 | 3 일차 | **4 일차** | 5 일차 | 6 일차 | 7 일차 | 8 일차 | 9 일차 | 10 일차

11 일차 | 12 일차 | 13 일차 | 14 일차 | 15 일차 | 16 일차 | 17 일차 | 18 일차 | 19 일차 | 20 일차

1장 프로그래밍 언어 활용

핵심 014 포인터

핵심 015 사용자 정의 함수

핵심 016 재귀 함수

[핵심014] 포인터

포인터와 포인터 변수

- 포인터는 변수의 주소를 말하며, C언어에서는 주소를 제어할 수 있는 기능을 제공한다.
- C언어에서 변수의 주소를 저장할 때 사용하는 변수를 포인터 변수라 한다.
- 포인터 변수를 선언할 때는 자료의 형을 먼저 쓰고 변수명 앞에 간접 연산자 *를 붙인다(예 int *a;).
- 포인터 변수에 주소를 저장하기 위해 변수의 주소를 알아낼 때는 변수 앞에 번지 연산자 &를 붙인다(예 a = &b;).
- 실행문에서 포인터 변수에 간접 연산자 *를 붙이면 해당 포인터 변수가 가리키는 곳의 값을 말한다(예 c = *a;).

포인터와 배열

- 배열을 포인터 변수에 저장한 후 포인터를 이용해 배열의 요소에 접근할 수 있다.
- 배열 위치를 나타내는 첨자를 생략하고 배열의 대표명만 지정하면 배열의 첫 번째 요소의 주소를 지정하는 것과 같다.

예 int a[5], *b;

b = a → 배열의 대표명을 적었으므로 a 배열의 시작 주소인 a[0]의 주소를 b에 저장한다.

b = &a[0] → a 배열의 첫 번째 요소인 a[0]의 주소(&)를 b에 저장한다.

	a[0]	a[1]	a[2]	a[3]	a[4]	← 배열 표기 방법
배열 a	첫 번째	두 번째	세 번째	네 번째	다섯 번째	
	*(a+0)	*(a+1)	*(a+2)	*(a+3)	*(a+4)	← 포인터 표기 방법

예제 다음 C언어로 구현된 프로그램의 출력 결과를 확인하시오.

```
main( )
{
    int a = 50; ❶ 정수형 변수 a를 선언하고 50으로 초기화한다.
    int *b; ❷ 정수형 변수가 저장된 곳의 주소를 기억할 포인터 변수 b를 선언한다.
    b = &a; ❸ 정수형 변수 a의 주소를 포인터 변수 b에 기억시킨다. b에는 a의 주소가 저장된다.

    *b = *b+20; ❹ b가 가리키는 곳의 값에 20을 더한다. b가 가리키는 곳이 a이므로 결국 a의 값도 바뀌는 것이다.
    printf("%d, %d", a, *b); ❺ 결과 70, 70
                    • ❷번과 같이 선언할 때 *는 해당 변수가 포인터 변수라는 것을 의미한다.
                    • ❹, ❺번과 같이 사용할 때 *를 붙이면 그 포인터 변수가 가리키는 곳의 값을 의미한다.
}
```

1. 다음 C언어로 구현된 프로그램을 분석하여 그 실행 결과를 쓰시오.

```
#include <stdio.h>
main() {
    int a[5], b = 1, sum = 0;
    for (int i = 4; i > -1; i--) {
```


```
        a[i] = b;
        b *= 3;
    }
    for (int i = 4; i > -1; i -= 2)
        sum += *(a + i);
    printf("%d", sum);
}
```

답 :

```
#include <stdio.h>
main() {
    int a[5], b = 1, sum = 0;

    for (int i = 4; i > -1; i--) { ❶

        a[i] = b; ❷
        b *= 3; ❸
    }
    for (int i = 4; i > -1; i -= 2) ❹

        sum += *(a + i); ❺
    printf("%d", sum); ❻
}
```

5개의 요소를 갖는 정수형 배열 a와 각각 초기값 1, 0을 갖는 변수 b, sum을 선언한다.

반복 변수 i가 4에서 시작하여 1씩 감소하면서 –1보다 큰 동안 ❷, ❸번을 반복 수행한다.

a[i]에 b의 값을 저장한다.

'b = b * 3;'과 동일하다. b에 3을 곱한 값을 저장한다.

반복 변수 i가 4에서 시작하여 2씩 감소하면서 –1보다 큰 동안 ❺번을 반복 수행한다.

sum에 (a + i)가 가리키는 곳의 값을 누적한다.

결과 91

※ Java와 Python에서는 포인터 변수를 사용할 수 없다.

※ 반복문 실행에 따른 변수들의 값의 변화는 다음과 같다.

i	b	sum	*(a+i)	a[ ] a[0]	a[1]	a[2]	a[3]	a[4]	출력
	1	0							91
4	3							1	
3	9						3	1	
2	27					9	3	1	
1	81				27	9	3	1	
0	243			81	27	9	3	1	
–1									
4		1	1						
2		10	9						
0		91	81						
–2									

정답 1. 91

[핵심015] 사용자 정의 함수

사용자가 필요한 기능을 취향대로 만들어 사용할 수 있는 함수이다. 사용자 정의 함수를 이용하면 프로그램 구조가 간단해지고 이해하기가 쉬워진다. 무엇보다 동일한 코드를 반복 입력하는 수고를 줄일 수 있다.

예제 다음 프로그램의 실행 결과를 확인하시오.

```
#include <stdio.h>
❶ void func(int i, int j);
main( )
{
❷ int a = 3, b = 12;
❸ func(a, b);
⓫ printf("%d, %d\n", a, b);
}
❹ void func(i, j)
❺ int i, j;
❻ {
❼     i *= 3;
❽     j /= 3;
❾     printf("%d, %d\n", i, j);
❿ }
```

사용할 사용자 정의 함수를 선언하는 곳이다. ❹번에서 작성하는 사용자 정의 함수를 이곳에서 정의하는 것이다. 이런 함수를 이 프로그램에서 만들어 사용하겠다는 의미이다.
- void : 사용할 함수의 리턴 값이 없음을 알려준다. 그대로 적어준다.
- func : 사용할 함수의 이름이다. ❹번에서 정의한 이름과 일치해야 한다.
- (int i ,int j) : 함수에서 사용할 인수이다. 호출하는 곳에서 보내준 인수의 순서와 자료형이 일치해야 한다. 인수로 사용하는 변수의 이름이 같을 필요는 없다.

정수형 변수 a와 b를 선언하고, 초기값으로 3과 12를 각각 할당한다.

a, b, 즉 3과 12를 인수로 하여 func 함수를 호출한다. 'func(3,12);'라는 의미이다. ❹번으로 이동한다.

결과 3, 12

func( ) 함수에서 돌려받은 값이 없으므로 원래의 a와 b의 값인 3과 12를 그대로 출력한다.

- void : 함수의 리턴 값이 없을 때 적어준다.
- func : 함수의 이름이다. 사용자가 임의로 지정하면 된다.
- (i, j) : 함수에서 사용할 인수이다. 호출하는 곳에서 보내준 인수의 순서와 자료형이 일치해야 한다. ❸번에서 'func(a, b)'라고 했으므로 i는 a의 값 3을 받고, j는 b의 값 12를 받는다.

인수로 받은 i와 j가 정수형 변수임을 선언한다. 꼭 해야 한다.

❻~❿번이 func 함수의 범위이다.

i = i * 3이므로 i는 9가 된다.

j = j / 3이므로 j는 4가 된다.

결과 9, 4

함수를 마치고 ⓫번으로 이동한다.

2020년 1회 기사 실기

1. 다음 C언어로 구현된 프로그램을 분석하여 그 실행 결과를 쓰시오.

```
#include <stdio.h>
void align(int a[]) {
     int temp;
     for (int i = 0; i < 4; i++)
          for (int j = 0; j < 4 - i; j++)
               if (a[j] > a[j+1]) {
                    temp = a[j];
                    a[j] = a[j+1];
                    a[j+1] = temp;}
               }
}
```


```
main() {
     int a[] = { 85, 75, 50, 100, 95 };
     align(a);
     for (int i = 0; i < 5; i++)
          printf("%d ", a[i]);
```

답 :

모든 C 프로그램은 반드시 main( ) 함수부터 시작해야 한다.

```
main() {
❶    int a[] = { 85, 75, 50, 100, 95 };
❷    align(a);
     for (int i = 0; i < 5; i++)
          printf("%d ", a[i]);
}
```

❶ 배열을 선언할 때 사용할 개수를 생략하고 초기값을 지정하면, 초기값으로 지정된 값의 수와 같은 크기의 배열이 선언된다.

	a[0]	a[1]	a[2]	a[3]	a[4]
배열 a	85	75	50	100	95

❷ a를 인수로 하여 함수 align( )을 호출한다. 인수로 배열의 이름을 지정하면 배열의 시작 주소가 인수로 전달된다. 그러니까 align(a)는 align(&a[0])과 같은 의미다.

```
❸ void align(int a[]) {
❹    int temp;
❺    for (int i = 0; i < 4; i++)
❻         for (int j = 0; j < 4 - i; j++)
❼              if (a[j] > a[j+1]) {
❽                   temp = a[j];
❾                   a[j] = a[j+1];
❿                   a[j+1] = temp;
⓫              }
⓬ }
```

❸ ❷번에서 'align(a)'라고 했으므로 정수형 배열 a는 main( ) 함수의 a 배열의 시작 주소를 받는다.

	a[0]	a[1]	a[2]	a[3]	a[4]
배열 a	85	75	50	100	95

❹ 정수형 변수 temp를 선언한다.
❺ 반복 변수 i가 0에서 시작하여 1씩 증가하면서 4보다 작은 동안 ❻번을 반복하여 수행한다.
❻ 반복 변수 j가 0에서 시작하여 1씩 증가하면서 4-i보다 작은 동안 ❼번을 반복하여 수행한다.
❼ a[j]가 a[j+1]보다 크면 ❽~❿번 사이의 문장을 실행한다.

반복문 실행에 따른 변수들의 값의 변화는 다음과 같다.

i	j	a[j]	a[j+1]	temp	배열 a
0	0 1 2 3 4	85 75 85 50 85 100 95	75 85 50 85 100 95 100	85 85 100	a[0] a[1] a[2] a[3] a[4] 85 75 50 100 95 75 85 85 95 100 50
1	0 1 2 3	75 50 75 85	50 75 85 95	75	a[0] a[1] a[2] a[3] a[4] 75 50 85 95 100 50 75
2	0 1 2	50 75	75 85		a[0] a[1] a[2] a[3] a[4] 50 75 85 95 100
3	0 1	50	75		a[0] a[1] a[2] a[3] a[4] 50 75 85 95 100
4					

⓫ if문의 끝이다.
⓬ 함수를 마치고 align(a) 함수를 호출했던 main( ) 함수로 제어를 옮긴다.

```
main() {
    int a[] = { 85, 75, 50, 100, 95 };
    align(a);
⓭   for (int i = 0; i < 5; i++)
⓮       printf("%d ", a[i]);
}
```

⓭ 반복 변수 i가 0에서 시작하여 1씩 증가하면서 5보다 작은 동안 ⓮번을 반복 수행한 후 프로그램을 종료한다.
⓮ a[i]의 값을 정수로 출력한 후 한 칸을 띄운다.

결과 `50 75 85 95 100`

정답 1. 50 75 85 95 100

[핵심016] 재귀 함수

자기가 자기를 호출하는 순환 프로그램을 의미한다. 재귀 함수는 순환하는 만큼 반복하여 실행하면서 변수에 저장된 값을 추적하면 결과를 이해하기 쉽다.

예제 다음은 재귀 함수를 이용해 팩토리얼(Factorial)을 구하는 프로그램이다. 실행 결과를 확인하시오.

```
#include <stdio.h>
main( )
{
   printf("%d", factorial(5));
}

factorial(int n) {
   if ( n <= 1 )
      return 1;
   else
      return n * factorial(n-1);
}
```

코드 해설

```
main()
{
❶   printf("%d", factorial(5));
}
```

❶ 5를 인수로 하여 factorial 함수를 호출한 다음 돌려받은 값을 정수형으로 출력한다.

①회

```
factorial(int n) {
❷   if ( n <= 1 )
        return 1;
    else
❸       return n * factorial(n-1);
}
```

factorial 함수가 호출될 때 5를 전달받았으므로 n은 50이다. ❷의 조건을 만족하지 않으므로 ❸을 수행한다.
❸을 수행하기 위해 factorial( ) 함수를 호출하는데, 호출할 때 전달되는 값은 factorial(n-1)이므로 factorial(4)인 상태로 호출된다.

②회

```
factorial(int n) {
❹   if ( n <= 1 )
        return 1;
    else
❺       return n * factorial(n-1);
}
```

factorial 함수가 호출될 때 4를 전달받았으므로 n은 40이다. ❹의 조건을 만족하지 않으므로 ❺를 수행한다.

❺를 수행하기 위해 factorial( ) 함수를 호출하는데, 호출할 때 전달되는 값은 factorial(n–1)이므로 factorial(3)인 상태로 호출된다.

③회

```
factorial(int n) {
❻    if ( n <= 1 )
          return 1;
     else
❼         return n * factorial(n-1);
}
```

factorial 함수가 호출될 때 3을 전달받았으므로 n은 3이다. ❻의 조건을 만족하지 않으므로 ❼을 수행한다.
❼을 수행하기 위해 factorial( ) 함수를 호출하는데, 호출할 때 전달되는 값은 factorial(n–1)이므로 factorial(2)인 상태로 호출된다.

④회

```
factorial(int n) {
❽    if ( n <= 1 )
          return 1;
     else
❾         return n * factorial(n-1);
}
```

factorial 함수가 호출될 때 2를 전달받았으므로 n은 2이다. ❽의 조건을 만족하지 않으므로 ❾를 수행한다.
❾를 수행하기 위해 factorial( ) 함수를 호출하는데, 호출할 때 전달되는 값은 factorial(n–1)이므로 factorial(1)인 상태로 호출된다.

⑤회

```
factorial(int n) {
❿    if ( n <= 1 )
⓫         return 1;
     else
          return n * factorial(n-1);
}
```

factorial 함수가 호출될 때 1을 전달받았으므로 n은 1이다. ❿의 조건을 만족하므로 ⓫을 수행한다.
'return 1;'이므로 함수의 실행을 종료하고 1을 반환하면서 제어를 ④회 factorial(n–1) 함수를 호출했던 곳으로 옮긴다.

④회

```
factorial(int n) {
     if ( n <= 1 )
          return 1;
     else
⓬        return n * factorial(n-1);
}
```

⑤회 수행 과정에서 1이 반환되었으므로
⓬ 2를 반환하면서 제어를 ③회 factorial(n–1) 함수로 옮긴다.

return n * factorial(n–1)
ⓐ ⓑ

– ⓐ : 2 ('factorial(n–1)'을 호출할 때 n은 2였으므로)
– ⓑ : 1 (⑤회 수행 과정에서 1이 반환되었으므로)

③회
```
factorial(int n) {
     if ( n <= 1 )
          return 1;
     else
❸         return n * factorial(n-1);
}
```

④회 수행 과정에서 2가 반환되었으므로
❸ 6을 반환하면서 제어를 ②회 factorial(n-1) 함수로 옮긴다.

return n * factorial(n-1)
3 2

②회
```
factorial(int n) {
     if ( n <= 1 )
          return 1;
     else
❹         return n * factorial(n-1);
}
```

③회 수행 과정에서 6이 반환되었으므로
❹ 24를 반환하면서 제어를 ①회 factorial(n-1) 함수로 옮긴다.

return n * factorial(n-1)
4 6

①회
```
factorial(int n) {
     if ( n <= 1 )
          return 1;
     else
❺         return n * factorial(n-1);
}
```

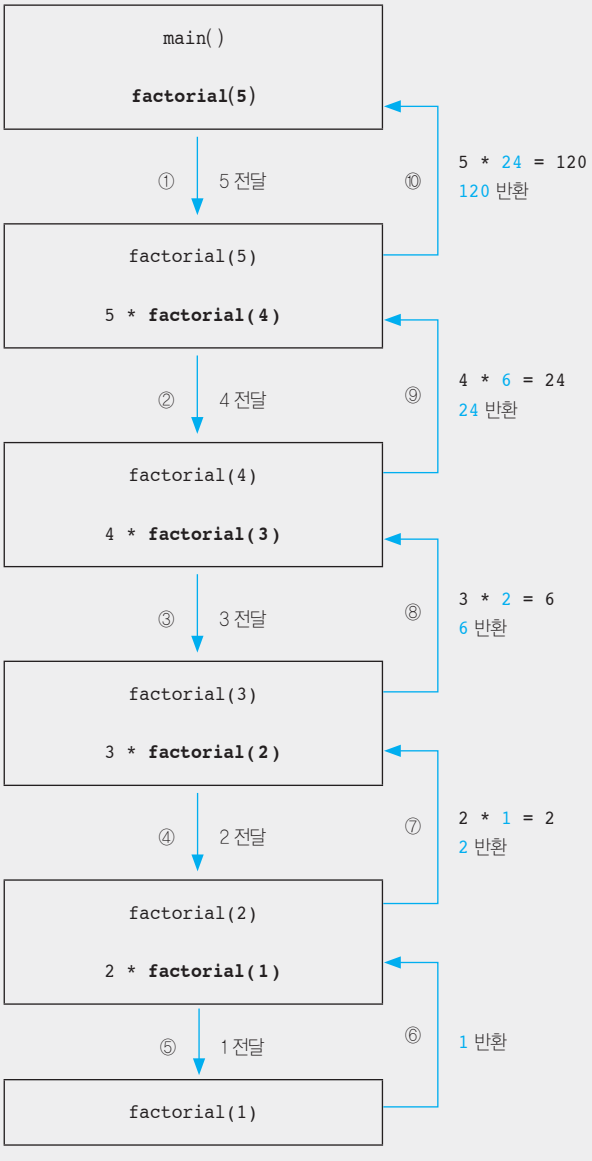
②회 수행 과정에서 24가 반환되었으므로
❺ 120을 반환하면서 제어를 처음 factorial(5) 함수로 옮긴다.

return n * factorial(n-1)
5 24

```
main()
{
❻   printf("%d", factorial(5));
}
```

❻ ①회 수행 과정에서 120이 반환되었으므로 돌려받은 값 120을 정수형으로 출력하고 프로그램을 종료한다.

지금까지의 재귀 함수 과정을 개괄적인 그림을 통해 살펴보자.



2020년 2회 기능사 실기

1. 다음 C언어로 구현된 프로그램을 분석하여 그 실행 결과를 쓰시오.

```
#include <stdio.h>
hrd(num) {
     if (num <= 0)
          return;
     printf("%d ", num);
     hrd(num-1);
}
main( ) {
     hrd(5);
     return 0;
}
```

답 :

C 언어는 main( ) 함수부터 시작해야 한다.

```
main()
{
❶   hrd(5);
    return 0;
}
```

❶ 5를 인수로 하여 hrd 함수를 호출한다.

①회

```
hrd(num) {
❷    if (num <= 0)
          return;
❸    printf("%d ", num);
❹    hrd(num-1);
}
```

hrd 함수가 호출될 때 5를 전달받았으므로 num은 5이다. ❷의 조건을 만족하지 않으므로 ❸을 수행한다.
❸ num의 값 5를 출력하고 한 칸을 띄운다.

결과 5

❹를 수행하기 위해 hrd( ) 함수를 호출하는데, 호출할 때 전달되는 값은 num-1이므로 hrd(4)인 상태로 호출된다.

②회

```
hrd(num) {
❺    if (num <= 0)
          return;
❻    printf("%d ", num);
❼    hrd(num-1);
}
```

hrd 함수가 호출될 때 4를 전달받았으므로 num은 4이다. ❺의 조건을 만족하지 않으므로 ❻을 수행한다.
❻ num의 값 4를 출력하고 한 칸을 띄운다.

결과 5 4

❼을 수행하기 위해 hrd( ) 함수를 호출하는데, 호출할 때 전달되는 값은 num-1이므로 hrd(3)인 상태로 호출된다.

③회
```
hrd(num) {
❽    if (num <= 0)
          return;
❾    printf("%d ", num);
❿    hrd(num-1);
}
```

hrd 함수가 호출될 때 3을 전달받았으므로 num은 3이다. ❽의 조건을 만족하지 않으므로 ❾를 수행한다.
❾ num의 값 3을 출력하고 한 칸을 띄운다.

결과 5 4 3

❿을 수행하기 위해 hrd( ) 함수를 호출하는데, 호출할 때 전달되는 값은 num-1이므로 hrd(2)인 상태로 호출된다.

④회
```
hrd(num) {
⓫    if (num <= 0)
          return;
⓬    printf("%d ", num);
⓭    hrd(num-1);
}
```

hrd 함수가 호출될 때 2를 전달받았으므로 num은 2이다. ⓫의 조건을 만족하지 않으므로 ⓬를 수행한다.
⓬ num의 값 2를 출력하고 한 칸을 띄운다.

결과 5 4 3 2

⓭을 수행하기 위해 hrd( ) 함수를 호출하는데, 호출할 때 전달되는 값은 num-1이므로 hrd(1)인 상태로 호출된다.

⑤회
```
hrd(num) {
⓮    if (num <= 0)
          return;
⓯    printf("%d ", num);
⓰    hrd(num-1);
}
```

hrd 함수가 호출될 때 1을 전달받았으므로 num은 1이다. ⓮의 조건을 만족하지 않으므로 ⓯를 수행한다.
⓯ num의 값 1을 출력하고 한 칸을 띄운다.

결과 5 4 3 2 1

⓰을 수행하기 위해 hrd( ) 함수를 호출하는데, 호출할 때 전달되는 값은 num-1이므로 hrd(0)인 상태로 호출된다.

⑥회
```
hrd(num) {
⓱    if (num <= 0)
⓲         return;
     printf("%d ", num);
     hrd(num-1);
}
```

hrd 함수가 호출될 때 0을 전달받았으므로 num은 0이다. ⓱의 조건을 만족하므로 ⓲을 수행한다.
'return;'이므로 함수의 실행을 종료하고 반환값 없이 제어를 ⑤회 hrd(0) 함수를 호출했던 곳으로 옮긴다.

⑤회
```
hrd(num) {
     if (num <= 0)
          return;
     printf("%d ", num);
     hrd(num-1);
} ⑲
```

⑲ 함수의 실행을 종료하고 반환값 없이 제어를 ④회 hrd(1) 함수를 호출했던 곳으로 옮긴다.

④회
```
hrd(num) {
     if (num <= 0)
          return;
     printf("%d ", num);
     hrd(num-1);
} ⑳
```

⑳ 함수의 실행을 종료하고 반환값 없이 제어를 ③회 hrd(2) 함수를 호출했던 곳으로 옮긴다.

③회
```
hrd(num) {
     if (num <= 0)
          return;
     printf("%d ", num);
     hrd(num-1);
} ㉑
```

㉑ 함수의 실행을 종료하고 반환값 없이 제어를 ②회 hrd(3) 함수를 호출했던 곳으로 옮긴다.

②회
```
hrd(num) {
     if (num <= 0)
          return;
     printf("%d ", num);
     hrd(num-1);
} ㉒
```

㉒ 함수의 실행을 종료하고 반환값 없이 제어를 ①회 hrd(4) 함수를 호출했던 곳으로 옮긴다.

①회
```
hrd(num) {
     if (num <= 0)
          return;
     printf("%d ", num);
     hrd(num-1);
} ㉓
```

㉓ 함수의 실행을 종료하고 반환값 없이 제어를 처음 hrd(5) 함수를 호출했던 곳으로 옮긴다.

```
main()
{
     hrd(5);
㉔   return 0;
}
```

㉔ main( ) 함수에서 'return 0;'는 프로그램 종료를 의미한다.

정답 1. 5 4 3 2 1