|  |  |  |  |
| --- | --- | --- | --- |
| 2022/2 『자료구조』과제 보고서 | | | |
| 제목 | 2장 실습( ) 과제( O ) | 제출일자 | 2022.   09.  27 . |
| 학번 | 201911608 | 이름 | 김지환 |

|  |
| --- |
| **실습 2장** |
| 02 순환 호출을 하였을 경우에 활성 레코드들이 저장되는 위치는 어디인가?  (1) 순환호출 함수내부 (2) 변수  (3) 배열 (4) 스택  -> 순환호출은 자기 자신을 호출하여도 A에서 A`로 다른 함수를 호출하는 것과 같으므로 A가 먼저 실행이 되고 A``가 실행이 되고 A```가 실행이 되면 A``` 함수의 내부처리가 끝난 후 A로 돌아온다. 즉 Last In First Out 이다. Stack에 저장된다.  4번 스택 |
| 04 하나의 함수가 호출할 수 있는 순환호출의 개수는?  (1) 1번 (2) 2번  (3) 스택이 허용하는 한도 (4) 무제한  -> 순환호출은 스택처럼 쌓이므로 스택이 허용하는 한도이다. 순환 호출에 따른 스택 크기는 계산이 어려우니 1만번을 초과하지 않는 것이 가장 좋다.  3번 스택이 허용하는 한도 |
| 06 다음의 순환호출 함수에서 잘못된 점은 무엇인가?  int recursive(int n) {  printf("recursive(%d)\n", n);  return n \* recursive(n - 1);  }  -> 조건이 지정되어 있지 않고 return에서 계속 recursive(n-1)을 호출 하므로  main 함수에서 recursive(0)을 실행 하였을 경우 0\*-1\*-2\*-3\*-4\*-5\*......\*-1만 이하 ( 스택의 한도 )까지 순환호출 될 것이다. 그래서 if (n==1) return 1; 등 조건을 추가해야한다. |
| 08 다음 함수를 recursive(5)로 호출하였을 때, 화면에 출력되는 내용과 함수의 반환 값을 구하라.  int recursive(int n) {  printf("%d\n", n);  if (n < 1) return 2;  else return(2 \* recursive(n - 1) + 1);  }  -> 위 코드에서 n을 출력 후 n 값이 0이 아닐 경우 n-1을 순환한다. recursive(5)의 경우 출력 내용은 5, 4, 3, 2, 1, 0 일 것이다. 반환 값은 recursive(1)일 경우 2\*recursive(0)+1 = 5이다.   |  |  |  | | --- | --- | --- | | 함수 | 출력 | 반환 값 | | recursive(0) | 0 | return 2 ==> 2 | | recursive(1) | 1, 0 | return 2\*2+1 ==> 5 | | recursive(2) | 2, 1, 0 | return 2\*5+1 ==> 11 | | recursive(3) | 3, 2, 1, 0 | return 2\*11+1 ==> 23 | | recursive(4) | 4, 3, 2, 1, 0 | return 2\*23+1 ==> 47 | | recursive(5) | 5, 4, 3, 2, 1, 0 | return 2\*47+1 ==> 95 | |
| 10. 다음 함수를 recursive(5)로 호출하였을 때, 화면에 출력되는 내용을 쓰시오.  int recursive(int n) {  if (n != 1) recursive(n - 1);  printf("%d\n", n);  }  -> 위 코드에서는 순환 후 출력을한다. 순환은 스택구조이므로 n이 5->4->3->2->1 순서로 함수에 진입하게 되며 마지막에 진입한 1부터 출력된다. 즉, 출력 값은 1->2->3->4->5 일 것이다. |
| 12 다음과 같은 함수를 호출하고 “recursive” 문자열을 입력한 다음, 엔터키를 눌렀다면 화면에 출력되는 것은?  int unknown() {  int ch;  if ((ch = getchar()) != '\n')  unknown();  putchar();  }  -> 다음과 같은 함수는 오류가 발생할 것이다. putchar(ch)로 수정해야한다.  putchar(ch)로 수정 할 경우 엔터를 입력받기 전 까지 unknown() 함수를 순환후 출력하기 때문에 스택구조로 출력된다. 즉 recursive를 입력했을 경우 reverse 된 evisrucer이 출력 될 것이다. |
| 14 다음을 계산하는 순환적인 프로그램을 작성하시오.  1 + 1/2 + 1/3 + ... + 1/n  #include <stdio.h>  double f\_sigma(int n) {  if (n == 1) return 1.0;  return ((1 / (double)n) + f\_sigma(n - 1));  }  int main() {  printf("%f",f\_sigma(2));  return 0;  } |
| 16 다음의 순환적인 프로그램을 반복 구조를 사용한 비순환적 프로그램으로 바꾸시오.  int sum(int n) {  if (n == 1) return 1;  else return n + sum(n - 1);  }  ->  int main() {  int sum = 0, n = 10;  for (int i = 1; i <= n;i++) sum += i;  printf("%d", sum);  return 0;  } |
| 18. Ackermann 함수는 다음과 같이 순환적으로 정의된다.  A (x, y) = { 1. y+1, if x = 0 2. A(x-1, 1), if y =0 3. A(x-1, A(x,y-1)), otherwise }  (a) A(3,2)와 A(2,3)의 값을 구하시오.   |  |  |  |  |  | | --- | --- | --- | --- | --- | | x y | 0 | 1 | 2 | 3 | | 0 | A(0, 0) = 1 | A(0, 1) = 2 | A(0, 2) = 3 | A(0, 3) = 4 | | 1 | A(0, 1) = 2 | A(0, A(1, 0)) = 3 | A(0, A(1, 1)) = 4 | A(0, A(1, 2)) = 5 | | 2 | A(1, 1) = 3 | A(1, A(2, 0)) = 5 | A(1, A(2,1)) = 7 | A(1, A(2,2)) = 9 | | 3 | A(2, 1) = 5 | A(2, A(3,0)) = 13 | A(2, A(3,1)) = 29 |  |   -> A(3, 2) = 29, A(2, 3) = 9  (b) Ackermann 함수를 구하는 순환적인 프로그램을 작성하시오.  #include <stdio.h>  int Ackermann(int x, int y) {  if (x == 0) return y + 1;  if (y == 0) return Ackermann(x - 1, 1);  return Ackermann(x - 1, Ackermann(x, y - 1));  }  int main() {  printf("A/B\t0\t1\t2\t3\n");  for (int i = 0; i < 4; i++) {  printf(" %d\t", i);  for (int j = 0; j < 4;j++) {  printf("%d\t", Ackermann(i,j));  }  printf("\n");  }  return 0;  }  (c) 위의 순환적인 프로그램을 반복구조를 사용한 비순환적 프로그램으로 바꾸시오.  #include <stdio.h>  int Ackermann(int x, int y) {  int acm[100000];  int i = 0;  acm[i++] = x, acm[i] = y;  while (i != 0) {  if (acm[i - 1] == 0) {  acm[i - 1] = acm[i] + 1;  i -= 1;  }  else if (acm[i] == 0) {  acm[i - 1] = acm[i - 1] - 1;  acm[i] = 1;  }  else {  acm[i + 1] = acm[i] - 1;  acm[i] = acm[i - 1];  acm[i - 1] = acm[i - 1] - 1;  i += 1;  }  }  return acm[i];  }  int main() {  printf("A/B\t0\t1\t2\t3\n");  for (int i = 0; i < 4; i++) {  printf(" %d\t", i);  for (int j = 0; j < 4;j++) {  printf("%d\t", Ackermann(i,j));  }  printf("\n");  }  return 0;  }  실행결과 |
| 20 순환호출에서는 순환호출을 할 때마다 문제의 크기가 작아져야 한다.  (1) 팩토리얼 계산 문제에서 순환호출이 일어날 때마다 문제가 어떻게 작아지는가?  -> n!이라고 할 시 1이 될 때까지 n-1만큼 작아진다. 3!(순환) -> 3\*2!(순환) -> 3\*2\*1  (2) 하노이의 탑에서 순환호출이 일어날 때마다 문제가 어떻게 작아지는가?  -> n 개의 원판이 있으면 2\*2^(n-1)-1개의 이동이 필요하므로 log2 만큼 작아진다.  -> 4개의 원판 -> 2 \* 3개의 원판 이동횟수 + 1 -> 2 \* ( 2 \* 2개의 원판의 이동횟수 + 1) ... |