

**자료구조및알고리즘 02분반**

**서두옥 교수님**

**20211497 옹미령**

1. 이진수를 십진수로 변환하는 함수
2. 소스 코드

|  |
| --- |
| /\*  보고서 #01  연습문제 #01 - 이진수를 십진수로 변환하는 함수  \*/  #include <iostream>  using namespace std;  //입력 값이 이진수인지 판단하는 함수  bool validateBinary(int num)  {  int temp;    while(1)  {  temp = num % 10; //10으로 나눠 이진수를 한자리씩 쪼개기  if(temp!=0 && temp!=1) return 0; //이진수가 아니면 바로 리턴 0  if(num<10) return 1; //나눠버릴 수가 1 이하의 자리수가 되면 이진수로 판단 후 리턴 1    num /= 10;  }  }  //이진수를 십진수로 반환하는 함수  int binaryToDecimal(int num)  {  int degree=1;//십진수 변환 시 곱해지는 2  int temp; //쪼갠 이진수를 저장하는 변수  int sum=0; //십진수로 반환한 결과를 저장하는 변수  int cnt=0; //이진수의 숫자 개수를 저장하는 변수    while(num>0)  {  temp = num % 10;  num /= 10; //이진수를 쪼갤 때마다 입력한 수의 자릿수도 10씩 나눔  cnt++; //이진수의 개수 확인    if(cnt==1) degree=1; //가장 아래자릿수. 즉 2^0이 곱해질 수인 경우에 곱해질 수는 2^0=1  else degree\*=2; //2^0이 곱해지지 않을 경우 수가 왼쪽으로 가면서 2의 차수는 하나씩 증가함을 반영    sum += degree \* temp; //2^N 과 쪼갠 이진수 패킷의 곱의 합. 마지막 텀에서는 최종적으로 10진수로 반환한 값이 됨  }    return sum;  }  int main()  {  int input;  bool flag=1; /\*이진수 값을 입력할 때까지 반복하는  while문에 쓰일 이진수 값이 입력되면 0이 되는 flag 변수\*/    //이진수 값을 입력할 때까지 입력받는 반복문  while(flag)  {  cout << "이진수(1 또는 0): ";  cin >> input;    if(validateBinary(input))  {  cout << endl;    flag = 0; //이진수 입력 받으면 flag=0으로 while반복문 빠져나감  cout << "입력한 이진수: " << input << endl;  cout << "변환 된 십진수 : " << binaryToDecimal(input) << endl;  }  else cout << "이진수(1또는 0)만 입력 가능합니다!!!" << endl;  }    return 0;  } |

1. 실행 결과

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

1. 연구 조사

본 연습문제는 크게 어렵지 않은 무난한 예제였지만, 알고리즘에 대한 사전 지식이 필요하였다. 다음의 부분을 논리회로 시간에 공부하여 미리 알고 있었지만 복습 차 정리한다.

* 이진수 -> 십진수 반환 방법

만약 ‘1011’이라는 이진수가 있다면, 각 자리의 이진수와 2^n을 곱한 값을 모두 더한 것이 십진수로 반환한 값일 것이다. 이때에 n을 결정하는 방법은 가장 오른쪽 자리수와 곱해지는 부분이 n=0, 왼쪽으로 갈수록 n이 1씩 증가한다.

즉, 0b1011의 경우 (가장 오른쪽 자리의 수부터 10진수로 반환한다.)

1 \* 2^0 = 1

1 \* 2^1 = 2

0 \* 2^2 = 0

1 \* 2^3 = 8

1 + 2 + 0 + 8 = 11이므로, 이진수 1011을 십진수로 반환한 수는 11이다.

컴퓨터 계산기의 값과 비교해도 정확한 값임을 알 수 있다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

1. 데이터 정렬 : 정적 배열과 동적 배열
2. 소스 코드

|  |
| --- |
| /\*  연습문제 #02  \*/  #include <iostream>  #include <vector>  #include <algorithm>  #define SIZE 10  using namespace std;  //정적 배열을 정렬하는 함수 - 순차정렬 사용  void sort\_arr(int\* arr, int arr\_size)  {  int temp; //두 변수 자리 교환 시 임시로 사용되는 변수    for(int i=0; i<arr\_size-1; i++)  {  for(int j=i; j<arr\_size; j++)  {  //더 큰 수를 뒤로 보내기  if(arr[i] > arr[j])  {  temp = arr[i];  arr[i] = arr[j];  arr[j] = temp;  }  }  }  }  int main(void)  {  int arr[SIZE]; //정적 배열  vector<int> v; //벡터를 이용한 동적 배열    cout << "임의의 정수(10개)를 입력하세요..." << endl;    //배열 입력받기  for(int i=0; i<SIZE; i++)  {  cout << "arr[" << i << "] : ";  cin >> arr[i]; //정적 배열의 원소 입력  v.push\_back(arr[i]);//동적 배열 탑에 원소 추가  // cout << endl;  }    //배열 정렬  sort\_arr(arr, SIZE); //정적 배열의 주소를 인자로 넘겨주어 배열 정렬  sort(v.begin(), v.end()); //동적 배열 정렬    //정렬한 정적 배열 출력  cout << "정렬(정적 배열): " ;    for(int i=0; i<SIZE; i++)  {  cout << arr[i] << " ";  }    //정렬한 동적 배열 출력  cout << endl;  cout << "정렬(동적 배열): " ;    for(int i=0; i<v.size(); i++)  {  cout << v[i] << " ";  }    cout << endl;  return 0;  } |

1. 실행 결과

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

1. 연구 조사

해당 문제를 풀면서 정적 배열은 순차 정렬, 동적 배열 정렬에는 algorithm stl의 sort() 함수를 사용하였다.

정렬(sorting)에는 순차 정렬, 버블 정렬, 삽입 정렬, 힙 정렬 외에 다양한 방법이 있다. 내가 택한 선택 정렬은 다음의 표에 따르면, 시간복잡도가 n^2으로 시간적으로 매우 효율적인 정렬 방법이라고는 할 수 없다. 하지만 문제는 시간복잡도에 대한 별도의 조건을 건 것이 아니고, 가장 빠르고 쉽게 구현할 수 있는 방법이 순차정렬이라 생각했기 때문에 본 방법으로 정적 배열을 정렬하였다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

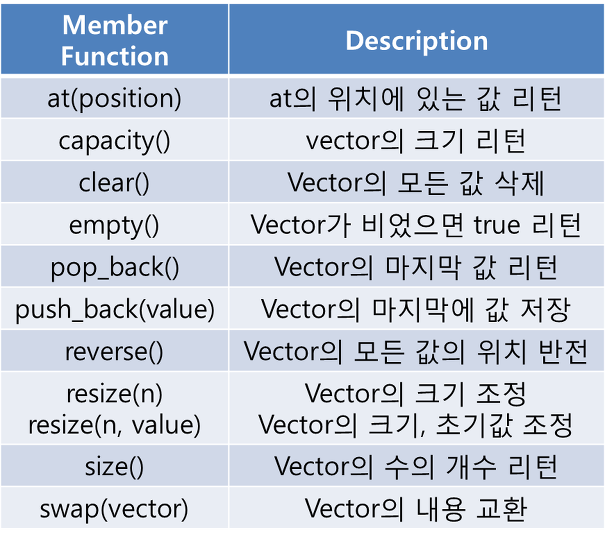
내가 선택한 순차정렬의 핵심 아이디어는 for 반복문을 여러 개 사용하여, 뒤쪽 순서의 원소들과 크기를 비교하며 오름차순 또는 내림차순을 위해 자리를 바꿔가는 것이다. for문을 보면 알 수 있듯이, 첫번째 for문을 기준으로 첫번째 for문에서 선택한 원소 바로 다음의 원소를 두번째 for문에서 선택하여 비교해가는 방식이다. 이렇게 비교하면 첫번째 for문의 값이 커질수록, 즉 for문의 반복횟수가 커질수록 정렬이 완료되어 정렬 시도 횟수가 줄어든다. 글보다는 다음의 예시를 통한 이해가 빠르다.

|  |  |
| --- | --- |
| int temp;  for(int i=0; i<SIZE-1; i++)  {  for(int j=i; j<SIZE; j++)  {  if(arr[i] > arr[j]) ①  {  temp = arr[i];  arr[i] = arr[j];  arr[j] = temp; ②  }  }  } |  |

동적 배열을 사용할 때는 c++의 STL sequence container 중 vector를 문제에 나온대로 사용하였다.

<vector> 를 헤더에 선언해주어야 사용이 가능하며, vector는 다수의 정보를 담는 역할을 하는 클래스인 STL 컨테이너의 일부로, 그중에서도 순차 컨테이너로 벡터와 동적배열을 다루는 컨테이너이다.

벡터 함수에는 다음과 같은 함수들이 있다.



또한 vector로 만든 동적 배열을 정렬할 때 STL 알고리즘을 이용하였다. vector와 마찬가지로 <algorithm> 헤더 선언 후 사용할 수 있으며, 자주 사용하는 STL 알고리즘의 함수는 다음과 같다. 이는 정렬이나 검색과 같은 알고리즘을 구현해놓은 함수이다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

1. 학생 성적 처리 : 동적 메모리 할당을 이용하여 학생들의 점수를 처리하는 프로그램을 작성하세요.
2. 소스 코드

|  |
| --- |
| /\*  연습문제 #03  \*/  #include <iostream>  using namespace std;  //총점과 평균 그리고 최댓값을 구하는 하나의 함수  void stuGrade(const double\* pArr, int num, double& rSum, double& rAve, double& rMax)  {  rMax = 0; //최댓값을 담을 변수. 비교를 위해 초기값을 0이라는 최저 점수로 설정한다.    for(int i=0; i<num; i++)  {  rSum += \*(pArr+i); //배열의 인덱스가 증가하면서 합 변수에 더해준다.    if(\*(pArr+i) > rMax) rMax = \*(pArr + i); //배열의 원소와 최댓값 변수를 비교해 최댓값을 찾는다.  }    rAve = rSum / num;  }  int main(void)  {  int num; //성적을 입력할 학생 수  double sum=0, ave, max; //총점, 평균, 최댓값값    cout << "입력할 학생(성적) 수: ";  cin >> num; //성적을 입력할 학생 수 입력  //cout << endl;    //성적을 입력할 학생 수만큼 동적 메모리 할당  double \*pArr = new double[num];    for(int i=0; i<num; i++)  {  cin >> \*(pArr + i);  }    //총점과 평균 그리고 최댓값을 구하는 하나의 함수 실행  stuGrade(pArr, num, sum, ave, max);  cout << endl << "##### 성적 결과 출력 #####" << endl << endl;  cout << "학생수: " << num << endl;  cout << "총 점: " << sum << endl;  cout << "평 균: " << ave << endl;  cout << "최댓값: " << max;    delete[] pArr;    return 0;  } |

1. 실행 결과

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. 연구 조사

C++에서는 C언어의 malloc(),free 외에도 new, delete을 이용한 동적 메모리 할당이 가능하다.

C++에서 C언어 스타일의 동적 메모리 할당을 위해서는 표준 라이브러리 헤더 파일 ‘<stdlib.h>’를 선언하면 된다. 이 라이브러리 선언 시 malloc, calloc, realloc이라는 다양한 명령어로 메모리를 할당하고 free로 메모리를 해제할 수 있다.

본 문제에서 사용한 동적 메모리 할당은 new를 이용했다.

|  |
| --- |
| //성적을 입력할 학생 수만큼 동적 메모리 할당  double \*pArr = new double[num];    for(int i=0; i<num; i++)  {  cin >> \*(pArr + i);  }  //….  delete[] pArr; |

코드의 일부에서 new를 사용한 부분이다. 문제에서 double형 배열을 할당받으라고 지시하였으므로 double 형 배열의 포인터를 이용해 num 크기 만큼의 double 메모리를 할당받는다. 포인터 변수는 동적으로 할당될 메모리에 접근하거나 해제할 수 있다.

또한 위 박스의 마지막 줄처럼 동적 메모리 할당 시 해제를 하지 않았을 때의 문제점인 메모리 leak 현상을 막고자 delete을 이용한 메모리 해제를 해준다.

또한 stuGrade(const double\* pArr, int num, double& rSum, double& rAve, double& rMax) 함수 작성 시 문제에 제시된 바로는 일부의 매개변수가 올 자리에 참조자를 사용하였다. 참조자로 선언된(int& ~) 변수의 자리에 오는 변수를 참조하여 원본 변수를 변경할 수 있다.

const가 붙은 첫번째 인자인 double \* pArr는 const에 의해 변경할 수 없는 상수가 되었으며, const가 붙은 이상 원본 pArr 포인터가 가리키는 배열을 임의로 stuGrade 함수에서 변경할 수 없다.

1. 로또 복권 : 동적 메모리 할당과 2차원 배열 자동 생성

* 로또 복권의 6개 숫자를 랜덤하게 생성하는 프로그램을 작성하세요.

• 원하는 로또 장수를 입력 받아 필요한 만큼 메모리를 할당하여 생성된 번호를 저장한다(2차원 동적 메모리 할당).

• 로또 번호는 임의의 난수 값을 발생 시켜 생성하고, 절대 중복된 값은 같은 줄에 생성되면 안된다.

1. 소스 코드

|  |
| --- |
| /\*  연습문제 #04  \*/  #include <iostream>  #include <cstdlib>  #define NUM 6 //로또 한 장당 숫자  #define RANGE 45 //나올 수 있는 로또 번호의 최댓값  #define MINIMUM 1 //나올 수 있는는로또 번호의 최솟값  using namespace std;  int main(void)  {  int cnt; //복권 장수    cout << "로또 복권을 몇 장 구입 하실래요? ";  cin >> cnt; //로또 복권 장수 입력    int\*\* pArr = new int\*[cnt]; //2차원 배열이므로 이중 포인터 사용하여 동적 메모리할당    cout << endl << "### 생성된 로또 번호 ###" << endl << endl;    for(int i=0; i<cnt; i++)  {  pArr[i] = new int[NUM]; //각 포인터에 두 번째 차원의 배열 할당    for(int j=0; j<NUM; j++)  {  int num; //난수를 저장하는 변수  bool redundancy; //중복이 아니면 0    do  {  redundancy=0; //같은 행에서 중복이 없으면 default 0인 변수    num = rand() % RANGE + MINIMUM; //1~45까지의 난수 생성    //같은 행에서의 중복 체크  for(int k=0; k<j; k++)  {  if(num == pArr[i][k])  {  redundancy = 1; //중복이면 1  break; //중복인 값이 있으면 반복문 빠져나가고 다시 난수 생성  }  }    if(!redundancy) pArr[i][j] = num;  //같은 행에서의 수와 생성된 난수가 일치하지 않는 경우, 난수로 해당 자리 할당    } while(redundancy); //중복성이 1인 경우에만 반복. 중복성이 0인 경우 반복하지 않고 뒤의 내용 수행    printf("%4d", pArr[i][j]);  }    delete[] pArr[i]; //각 행에 할당된 메모리 해제  cout << endl;  }    delete[] pArr; //메모리 해제    return 0;  } |

1. 실행 결과

텍스트, 스크린샷, 컴퓨터, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. 연구 조사
   * 제한된 범위 내에서 임의의 난수 값을 발생시키는 부분 구현

난수 발생 관련 함수인 rand(), srand(seed) 등의 함수를 내포하는 라이브러리 헤더는 C언어의 <stldlib.h>이다. C++로 이루어진 이 코드에서 해당 라이브러리를 사용하기 위해 <cstldlib>을 선언하였다.

하지만 임의의 난수 값을 발생시킬 때 이 문제에서는 절대 같은 행의 원소끼리는 같은 값을 내면 안 된다는 규칙이 있었다. 이 점을 해결하기 위해 같은 행에서의 원소끼리 중복 체크를 할 수 있도록 이중 for문 내에 중복성을 판단하는 bool 변수 redundancy(default=0)를 이용해 for문을 하나 더 써서 그 안에서 전의 원소와 생성한 난수의 값을 비교하며, 일치하지 않은 경우에는 생성된 난수가 중복되지 않은 적당한 값이라고 보고 이차원 배열의 원소에 포함시켜 행을 완성해갔다.

이때, 제한된 범위 내에서 stlib.h 헤더를 이용해 임의의 난수 값을 발생시키는 식은 다음과 같다.

num = rand() % range + mininum;

rand()는 stlib.h의 난수 발생 함수이며, 이를 나누는 range는 범위의 분량(예를 들어 1~45면 총 45개의 개수 분량이므로 45, 10~20이면 10 등), minimum은 나올 수 있는 최소 난수의 값을 의미한다.

* + 이차원 배열을 동적 할당받는 부분

이차원 배열을 동적 메모리 할당 및 메모리 해제하는 것은 일반적인 상식과는 거리가 있어 관련 지식이 필요하다.

문제 출제에서부터 이차원 배열을 메모리 할당받으라고 하였기 때문에, new 명령어를 이용하여 이차원 동적 배열을 할당받았다. 단, 이때 이중포인터를 사용하였는데, 일차적으로 포인터를 이용해 동적 메모리를 할당받는 것이 적절하기 때문이고, 이차적으로 2차원 배열이므로 이중 포인터를 사용하는 것이 적합하기 때문이다.

int\*\* pArr = new int\* [cnt];

또한 메모리 해제도 중요하다. do-while문을 수행한 후 각 행에 할당된 메모리부터 해제를 하기 위해 delete[] pArr를 사용하고 마지막에 delete[] pArr을 하여 메모리 누수를 막는다.

|  |
| --- |
| delete[] pArr[i]; //각 행에 할당된 메모리 해제  cout << endl;  }    delete[] pArr; //메모리 해제 |

1. 문자열을 숫자(정수)로 변환: atoi, stio

문자열을 숫자(정수)로 변환하는 함수를 작성하세요.

변환하고자 하는 문자열은 ‘0’ ~ ‘9’ 사이의 문자로 구성된다.

– 만약, ‘0’ ~’9’ 이외의 문자가 존재한다면 문자열을 숫자로 변환하는 작업을 중단한다.

– 첫 번째 문자로 양수와 음수를 표현하는 두 가지 문자(+, -)만을 예외로 허용한다.

1. 소스 코드

문제에서 정확히 고지하지 않았기 때문에 각 원형마다 따로 함수를 두어 코드를 작성하였다.

|  |
| --- |
| int my\_atoi (const char\* pStr); // (C) 정적 배열과 포인터 |
| #include <iostream>  #define SIZE 256  using namespace std;  int my\_atoi(const char\* pStr)  {  int sign = 1; //부호  int res = 0; //정수로 반환한 값을 저장할 변수    for(int i=0; ; i++)  {//12345  if(i==0 && \*(pStr+i)=='+') sign=1;  else if(i==0 && \*(pStr+i)=='-') sign=-1; //부호 판단  else if(\*(pStr+i)>='0' && \*(pStr+i)<='9')  {  res = res \* 10 + (\*(pStr+i)-'0'); //문자열을 정수로 반환  }  else return sign\*res; //0~9가 아닌 수가 문자열에 입력되면 바로 리턴하고 함수 종료  }    return sign\*res;  }  int main()  {  char Str[SIZE];    cout << "정수로 변환할 문자열 입력: ";  cin >> Str;  cout << "입력 문자열: " << Str << ", ";  cout << "변환된 숫자(int): " << my\_atoi(Str) << endl;    return 0;  } |
| int my\_stoi(const string& rStr); // (C++) string 클래스와 참조 |
| #include <iostream>  #include <string>  using namespace std;  int my\_stoi(const string& rStr) //cpp class와 참조를 이용한 함수  {  int sign=1; //부호  int res=0; //정수로 변환한 결과값    for(int i=0; i<rStr.length(); i++)  {  if(i==0 && rStr[0]=='+') sign=1;  else if(i==0 && rStr[0]=='-') sign=-1; //부호 처리 부분  else if(rStr[i]>='0' && rStr[i]<='9')  {  res = res \* 10 + (rStr[i] - '0'); //정수로 변환하는 부분. 이때 '0'을 빼주는 것으로 문자를 정수로 변환하여 더해주어야.  }  }    return res\*sign;  }  int main()  {  string Str;    cout << "정수로 변환할 문자열 입력: ";  cin >> Str;  cout << "입력 문자열: " << Str << ", ";  cout << "변환된 숫자(int): " << my\_stoi(Str) << endl;    return 0;  } |

1. 실행 결과

|  |  |
| --- | --- |
| int my\_atoi | int my\_stoi |
|  |  |

1. 연구 조사

* 문자열을 정수로 변환하는 logic

문자열을 for문과도 같은 반복문을 이용해 한 문자씩 쪼갠 후 str-‘0’ 과 같이 아스키코드를 이용해 문자를 숫자로 바꿔주는 방법을 사용해 한 문자씩 정수로 변환한다.

문자열의 크기(즉, 정수의 개수)에 따라 반복될 것인데, 이는 자릿수와도 같으므로 결과를 저장할 변수에 기존의 변수값과 10을 곱해준 값을 더해주고 새로 정수로 변환한 문자의 정수 변환값을 더해준다.

이 로직이 구현된 부분은 두 code의 다음과 같은 부분이다:

|  |
| --- |
| else if(rStr[i]>='0' && rStr[i]<='9')  {  res = res \* 10 + (rStr[i] - '0'); //정수로 변환하는 부분. 이때 '0'을 빼주는 것으로 문자를 정수로 변환하여 더해주어야.  } |

정수가 아닌 부분에서는 변환을 멈추고 리턴을 해야하므로 if문을 이용한 조건문에서 분해한 문자가 아스키 코드로 정수 범위에 있는지 판단하는 코드를 추가하였다.

* string class의 함수

my\_stoi를 구현할 때 string class를 사용할 필요가 있었기 때문에, string class를 헤더에 선언 후 string class의 함수를 사용하였다. 편의를 위해 atoi를 구현할 때와 최대한 비슷한 로직을 사용하고자 .length() 함수만 사용하였지만, string class에는 유용한 많은 함수들이 있고 다음과 같은 함수들이 있다:

|  |  |
| --- | --- |
| Str.append(문자열)  Str.insert(n, 문자열)  Str.replace(n, m, 문자열)  Str.clear()  Str.erase()  Str.push\_back(문자)  Str.pop\_back  getline(cin, str)  getline(cin ,str, 문자)  string str1 = “문자열1”  string str2(“문자열2”) | Str 뒤에 문자열 삽입  N 문자 뒤에 문자열 삽입  N 문자부터 m개의 문자를 문자열로 대체  문자열 모두 지움  문자열 모두 지움  Str 끝에 문자 삽입  Str 맨 뒤에 문자(top) 제거  한 줄 통째로 입력  문자전까지 입력  string 객체 생성  string 객체 생성 |

1. 은행 계좌의 인증과 출금: 응용

계좌 번호와 비밀번호의 일치 여부를 판단하여 출금 처리하는 프로그램을 작성하세요.

• 이름, 계좌번호, 비밀번호 그리고 잔액은 임의로 설정하여 객체를 생성하세요.

• 계좌번호 또는 비밀번호가 다를 경우 Account 클래스에 대한 객체를 생성하여 예외 처리하세요.

• 희망 출금액이 잔액보다 적을 경우도 별도 예외 처리하세요

소스 코드

|  |
| --- |
| #include <iostream>  using namespace std;  class Account  {  const string \_\_name; //계좌주 이름  const string \_\_account; //계좌번호  string \_\_pass; //비밀번호  int \_\_balance; //계좌 잔액  public:  //생성자 목록(문제 제시)  Account(string \_\_name, string \_\_account, string \_\_pass, int \_\_balance)  : \_\_name(\_\_name), \_\_account(\_\_account), \_\_pass(\_\_pass), \_\_balance(\_\_balance)  {}  string getName(void) const  {  return \_\_name;  };  string getAccount(void) const  {  return \_\_account;  };  string getPass(void) const  {  return \_\_pass;  };  int getBalance(void) const  {  return \_\_balance;  };  void showAccount(void)const  {  cout << "이 름: " << getName() << endl;  cout << "계좌번호: " << getAccount() << endl;  cout << "패스워드: " << getPass() << endl;  cout << "현재잔액: " << getBalance() << endl;  };  };  int main(void) {  string account, pass;  int amount, balance;    Account AC("홍길동", "111-2222-3333", "12345", 100000);    //계좌 정보 출력  cout << endl;  AC.showAccount();  cout << endl;  cout << "##### 은행 계좌 인증 #####" << endl;  cout << "계좌번호 입력: "; cin >> account;  cout << "비밀번호 입력: "; cin >> pass;    try  {  if(account != AC.getAccount() || pass != AC.getPass())  {  //계좌 정보가 일치하지 않을 경우  throw 0;  }  else  {  cout << "출금액 입력: "; cin >> amount;  if(amount > AC.getBalance())  {  //출금이 잔액을 초과한 경우  cout << "현재 잔액: " << AC.getBalance() << endl;  throw 'e';  return 0;  }  else  {  //출금 처리  balance = AC.getBalance() - amount;  cout << "현재 잔액: " << balance << endl;  cout << AC.getName() << "님의 현재 잔액은: " << balance << "입니다." << endl;  }  return 0;  }  }  catch(int exception)  {  //계좌 정보가 불일치할 경우의 예외 처리  cout << endl << "다음 입력을 다시 한 번 확인하세요!!!" <<endl;  Account AC("", account, "11111", 0);  AC.showAccount();  return 0;  }  catch(char exception)  {  //출금액이 잔액을 초과한 경우의 예외처리  cout << "희망 출금액: (" << amount << ")이 너무 많네요." << endl;  return 0;  }  } |

1. 실행 결과

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

1. 연구 조사

* try catch를 이용한 예외처리

try-catch문은 C++에서 예외처리를 할 때 사용하는 전형적인 문법이다.

C에서 사용했던 대표적인 if 조건문을 이용해서도 예외처리를 할 수 있지만, try catch문을 사용하면 누가봐도 예외처리를 위한 코드라는 점이 명확해져 유용하다.

대략적인 try catch 문법은 아래와 같다.

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

대표적으로 어떤 연산을 수행할 때 발생하는 오버플로우, 언더플로우 상황에서의 예외 처리에 유용하다. 다음은 이를 적용한 다중 예외처리 코드이다. 아래 코드에서는 오버플로우 또는 언더플로우를 방지한다.

|  |
| --- |
| #include <iostream>  #include <vector>  using namespace std;  void main() {  vector <int> v{1,2,3};  try  {  for (int i = 0; i < 5; i++) {  if (v.size() <= 0) {  throw string("Underflow!");  }    if (v.size() < i) {  throw i;  }  cout << v[i] << endl;  v.pop\_back();  }  }  catch (string error)  {  cout << error << endl;  }  catch (int index)  {  cout << "v["<< index << "]는 vector의 범위를 벗어났습니다." << endl;  }  } |

1. Point 클래스: 객체간 덧셈과 뺄셈 연산(멤버 함수, 일반 함수)

피 연산자가 모두 Point 클래스 객체인 경우 덧셈과 뺄셈이 가능하도록 연산자를 다중 정의(Overloading) 하세요. 단 일반 함수로 다중 정의하세요.

1. 소스 코드

|  |
| --- |
| /\*  연습문제 #07  \*/  #include <iostream>  using namespace std;  class Point  {  int \_\_x;  int \_\_y;  public:  //\_\_x , \_\_y 멤버 변수를 매개변수 x, y 값으로 초기화  Point(int x=0, int y=0) : \_\_x(x) , \_\_y(y) {}    void showPosition(void)  {  //(x, y) 꼴 나타내기기  cout << "(" << \_\_x << ", " << \_\_y << ")" << endl;  }    Point operator+(const Point& p) const; //객체 간 덧셈    friend Point operator-(const Point& p1, const Point& p2);  };  Point Point::operator+(const Point& p) const //객체 간 덧셈  {  Point res;    res.\_\_x = this->\_\_x + p.\_\_x;  res.\_\_y = this->\_\_y + p.\_\_y;  return res;  }  Point operator-(const Point& p1, const Point& p2) //객체 간 뺄셈  {  Point res;    res.\_\_x = p1.\_\_x - p2.\_\_x;  res.\_\_y = p1.\_\_y - p2.\_\_y;  return res;  }    int main()  {  Point p1(10, 20);  cout << "p1: ";  p1.showPosition();    Point p2(30, 40);  cout << "p2: ";  p2.showPosition();      Point p3 = p1 + p2;  cout << endl << "p3 = p1 + p2" << endl << "p3: ";  p3.showPosition();    Point p4 = p1 - p2;  cout << endl << "p4 = p1 - p2" << endl << "p4: ";  p4.showPosition();    return 0;  } |

1. 실행 결과

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. 연구 조사

* this 포인터

this 포인터는 클래스의 실제 인스턴스에 대한 주소를 가리키는 포인터이다. 위의 코드에서 operator+에서 this를 사용하는데 이는, 생성한 객체인 res나 p가 아닌 main에서 x, y를 할당한 피연산 포인터의 멤버 변수를 뜻하는 것이다. 즉, Point p3 = p1 + p2을 예로 들면, \_\_x의 this->\_\_x는 피연산 객체인 p1의 멤버변수를 말하는 것임을 알 수 있다.

이렇게 this포인터는 클래스의 멤버 변수와 매개 변수의 이름이 동일할 때나, 객체 자신의 주소를 리턴할 때 사용할 수 있다.

* friend

함수나 클래스 선언 앞에 선언할 수 있는 friend를 선언한 함수나 class에서는 접근 제어 지시자(private, public, protected)의 영향을 받지 않을 수 있다.

하나의 클래스에서 다른 클래스의 내부 데이터에 접근 해야할 경우 프렌드를 써서 권한을 줄 때 사용하면 유용하다. friend를 사용하면 자신의 개체뿐만이 아닌 다른 타입의 개체를 접근하는 것이 가능해지기 때문에 코드의 확장이 수월하게 이루어질 수 있다. 하지만 friend야말로 캡슐화에 지장을 줄 수 있기 때문에 가급적 남발하지 않고 사용해야 한다.

위의 코드에서는 문제에서 제시한 대로 operator-를 구현할 때 friend를 사용하였다. 덕분에 외부에서 일반함수로 operator-를 구현할 때 operator+와 다르게, 접근 제어 지시자의 영향을 받지 않아 private 멤버에 접근이 가능했으며, 해당 클래스의 멤버로 간주되어 외부 구현부에서 범위 지정 연산자 ::를 사용하지 않아도 되었다.

1. Person과 Student 클래스

일반적인 사람(Person) 클래스와 이를 상속 받은 학생(Student) 클래스를 작성 하세요.

1. 소스 코드

|  |
| --- |
| /\*  연습문제 #08  \*/  #include <iostream>  #include <string> //string 을 사용하므로 선언해준다  using namespace std;  class Person  {    public:  Person(const string& name, const string& gender, const string& tel, const string& addr)  : name(name), gender(gender), tel(tel), addr(addr)  {}    //이름부터 주소까지 정보 출력부분  void print\_Info() const  {  cout << "이 름: " << name << endl;  cout << "성 별: " << gender << endl;  cout << "연락처: " << tel << endl;  cout << "주 소: " << addr << endl;  }    protected:  string name;  string gender;  string tel;  string addr;  };  //Person 클래스로부터 상속받은 Student 클래스  class Student : public Person  {  public:    //생성자  Student(const string& name, const string& gender, const string& tel, const string& addr, int kor, int eng, int math)  : Person(name, gender, tel, addr), kor(kor), eng(eng), math(math)  {  calc\_tot\_ave();  }    //과목 및 총점, 평균 출력부  void print\_Scores() const  {  cout << "국 어: " << kor << endl;  cout << "영 어: " << eng << endl;  cout << "수 학: " << math << endl;  cout << "총 점: " << tot << endl;  cout << "평 균: " << ave << endl;  }  private:  int kor, eng, math, tot;  double ave;    //총점과 평균 계산  void calc\_tot\_ave()  {  tot = kor + eng + math;  ave = tot / 3.0;  }  };  int main()  {  string name, gender, tel, addr;  int kor, math, eng;    cout << "학생의 정보와 성적을 입력하세요..." << endl << endl;    cout << "이 름: ";  getline(cin, name); //공백이 입력될 수도 있으므로 getline(string) 함수를 사용한다.    cout << "성 별: ";  cin >> gender;    cout << "연락처: ";  getline(cin, tel);    cout << "주 소: ";  getline(cin, addr);    cout << "국어: ";  cin >> kor;    cout << "영어: ";  cin >> eng;    cout << "수학: ";  cin >> math;    Student student(name, gender, tel, addr, kor, eng, math);    cout << endl << "#입력 정보#" << endl << endl;    student.print\_Info(); //이름부터 주소까지 출력  student.print\_Scores(); //과목 성적과 총점, 평균 출력  return 0;  } |

1. 실행 결과

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. 연구 조사

* getline()

이름, 주소 문자열을 입력받을 때 공백을 포함한 문자열을 입력받아야 했다. 따라서 기존의 cin으로 해결할 수 없으므로 공백을 포함한 문자열을 입력할 수 있는 getline() 함수를 사용한다. 이 함수는 string 클래스 선언 후 사용이 가능하다.

* 상속의 멤버 접근 권한

해당 문제는 상속을 이용하는 문제이다.

class Student : public Person 이라는 명령어를 사용해 Student라는 클래스가 Person이라는 클래스에게 public 권한으로 상속받았다는 것을 나타낸다. public, protected, private의 3가지 접근권한으로 상속받을 수 있다.

각 접근 권한 별로 설명을 하자면,

* + - public 상속: C++에서 클래스의 상속은 기본적으로 public 상속이다. 이 경우, 기반 클래스의 public 멤버는 파생 클래스에서 public 멤버로 상속된다. 반면 protected , private 멤버는 파생 클래스 내에서 접근할 수 없다.
    - protected 상속: protected 상속을 사용하면 기반 클래스의 public 및 protected 멤버는 파생 클래스의 protected 멤버로 상속된다. private 멤버는 파생 클래스에서 접근할 수 없다.
    - private 상속: private 상속을 사용하면 기반 클래스의 모든 멤버는 파생 클래스의 private 멤버로 상속된다. 이 경우, 외부에서는 파생 클래스의 멤버로는 접근할 수 없다.

1. Person과 Student 클래스
2. 소스 코드

|  |
| --- |
| #include <iostream>  #include <string> //문자열 처리를 위한 헤더 선언언  using namespace std;  //Person 클래스 정의  class Person {  protected:  string \_\_name;  int \_\_age;  public:  //생성자 정의  Person(const string& \_name, int \_age)  : \_\_name(\_name), \_\_age(\_age) {}  //가상함수로 선언된 정보 출력 함수  virtual void print\_info() {  cout << "이름: " << \_\_name << ", 나이: " << \_\_age << endl;  }  };  // Student 클래스 정의  class Student : virtual public Person {  protected:  int \_\_admissionYear;  int \_\_graduationYear;  public:  Student(const string& \_name, int \_age, int \_admissionYear, int \_graduationYear)  : Person(\_name, \_age), \_\_admissionYear(\_admissionYear), \_\_graduationYear(\_graduationYear) {}  //추가 정보 출력 함수  virtual void print() {  cout << "이름: " << \_\_name << ", 나이: " << \_\_age << endl;  }  //정보 출력 함수를 오버라이드  void print\_info() override {  cout << "입학 년도: " << \_\_admissionYear << std::endl;  cout << "졸업 년도: " << \_\_graduationYear << std::endl;  }  };  //Worker 클래스 정의  class Worker : virtual public Person {  protected:  int \_\_workingTime;  public:  Worker(const string& \_name, int \_age, int \_workingTime)  : Person(\_name, \_age), \_\_workingTime(\_workingTime) {}    virtual void print() {  cout << "이름: " << \_\_name << ", 나이: " << \_\_age << endl;  }  void print\_info() override {    cout << "작업 시간: " << \_\_workingTime << endl;  }  };  //Albamon 클래스 정의  class Albamon : public Student, public Worker {  public:  Albamon(const string& \_name, int \_age, int \_admissionYear, int \_graduationYear, int \_workingTime)  : Person(\_name, \_age), Student(\_name, \_age, \_admissionYear, \_graduationYear), Worker(\_name, \_age, \_workingTime) {}  virtual void print() {  cout << "이름: " << \_\_name << ", 나이: " << \_\_age << endl;  }    void print\_info() override {  Student::print\_info();  Worker::print\_info();  }  };  int main() {  //각 클래스 객체 생성  Person aPerson("단군", 100);  Student bStudent("이도", 25, 2020, 2024);  Worker cWorker("이순신", 50, 100);  Albamon dAlbamon("철수", 20, 2023, 2027, 120);  aPerson.print\_info();    bStudent.print();  bStudent.print\_info();    cWorker.print();  cWorker.print\_info();    dAlbamon.print();  dAlbamon.print\_info();  return 0;  } |

1. 실행 결과

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. 연구 조사

* 가상함수

virtual function, 가상함수는 부모 클래스에서 상속받을 자식 클래스에 재정의할 것으로 기대하고 정의해놓은 함수이다. 생성된 가상함수는 파생 클래스에서 재정의하면 이전에 정의됐던 내용들은 모두 새롭게 정의된 내용들로 교체된다.

virtual이라는 예약어를 함수 앞에 붙여서 생성할 수 있으며, 이렇게 생성된 가상함수는 파생 클래스에서 재정의하면 이전에 정의되었던 내용들은 모두 새롭게 정의된 내용들로 교체된다.

컴파일러는 함수를 호출할 때 매우 복잡한, ‘정적 바인딩’이라고 하는, 컴파일러가 함수를 호출하는 코드를 컴파일 타임에 고정된 메모리 주소로 변환시키는 과정을 거치며 일반 함수는 모두 정적바인딩을 한다. 하지만 일반 함수를 오버로딩할 시 정적 바인딩으로 인해 문제가 될 수 있는데, 가상함수가 아닌 보통 함수의 경우 부모 클래스형 포인터로 멤버 함수를 호출할 때 컴파일러는 정적 타입을 보고 이 타입에 맞는 멤버 함수를 호출하기 때문이다.

이러한 상황을 막기 위해 동적바인딩을 한다. 동적 바인딩을 하려면 일반 함수들을 가상함수로 바꾸어 선언하여 포인터의 타입이 아닌 포인터가 가리키는 객체의 타입에 따라 멤버 함수를 선언할 수 있게 한다.

가상함수의 사용법은 ‘virtual 반환형 메서드명’의 형식을 가진다.

* 상속 오버라이딩(inheritance overriding)

오버라이딩은 이미 있는 함수를 무시하고 새롭게 함수를 재정의하는 것이다. 즉, overridingn은 부모 클래스와 자식 클래스의 상속 관계에서 부모 클래스에 이미 정의된 함수를 같은 이름으로 자식 클래스에서 재정의하는 것을 뜻한다. 단, 이때 부모의 멤버 함수와 원형이 같아야 한다. 또한 오버라이딩 시 부모 클래스의 함수가 모두 가려진다.

Albamon 클래스에서 print\_info 함수를 오버라이딩한 부분:

void print\_info() override {

Student::print\_info();

Worker::print\_info();

}

C++에서 함수 오버라이딩은 부모 클래스의 가상 함수를 자식 클래스에서 재정의하는 것을 의미한다. 부모 클래스 Person에서 print\_info 함수가 가상 함수로 선언되어 있으므로, 이 함수를 자식 클래스인 Student, Worker, 그리고 Albamon 클래스에서 다시 정의할 수 있다. 이런 식으로 Albamon 클래스에서 print\_info 함수를 오버라이딩하고 부모 클래스의 함수를 호출함으로서 Albamon 클래스 객체의 정보를 출력하는 동작을 정의한다. 이렇게 하면 Albamon 클래스는 Student 및 Worker 클래스의 특성을 모두 가지며 자체적 출력 형식을 가질 수 있다.

1. 다양한 자료형 출력: 함수 템플릿

배열의 원소가 다양한 자료형일 경우 하나의 출력 함수로만 처리할 수 있도록 프로그램을 작성하세요.

1. 소스 코드

|  |
| --- |
| /\*  연습문제 #10  \*/  #include <iostream>  using namespace std;  //템플릿 함수를 사용해 다양한 자료형의 배열 처리  template <typename T>  void printArr(const T\* pArr, const int num)  {  for(int i=0; i<num; i++) cout << pArr[i] << " ";  cout << endl;  }  int main()  {  char charArr[]="H i ~ C l i c k s e o";  int intArr[]={10,20,30,40,50};  double doubleArr[]={10.5, 20.5, 30.5, 40.5, 50.5};    //다양한 자료형의 배열 출력  printArr(charArr, sizeof(charArr)/sizeof(char));  printArr(intArr, sizeof(intArr)/sizeof(int) );  printArr(doubleArr, sizeof(doubleArr)/sizeof(double));    return 0;  } |

1. 실행 결과

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

1. 연구 조사

* 템플릿(Template) 함수

템플릿 함수는 템플릿 매개변수 T에 따라 컴파일러가 자동으로 해당 자료형에 맞게 코드를 생성한다. 따라서 T에 어떠한 자료형을 전달하느냐에 따라 printArr 함수는 다양한 자료형의 배열을 출력할 수 있다.

template <typename T>은 printArr을 템플릿 함수로 만들기 위한 선언이다. typename T는 템플릿 매개변수, T는 임의의 자료형이다. 이런 식으로 함수 printArr을 다양한 자료형에 대해 사용할 수 있다. void printArr(const T\* pArr, const int num) 부분은 템플릿 함수 printArr의 선언으로, T 자료형의 포인터 pArr과 배열의 원소 개수를 나타내는 정수 num을 인자로 받는다.