CODE COMPLETE 스터디
16.2~16.4 루프

컨텐츠검색플랫폼팀
조영일

16 루프 제어

- 16.2 루프의 제어
- 16.3 루프를 쉽게 작성하는 방법 – 안에서부터 밖으로
- 16.4 루프의 배열의 대응

16.2 루프의 제어

- 루프의 문제
  - 잘못된 초기화 또는 생략
  - 연산자 혹은 루프와 관련된 변수들의 초기화 생략
  - 부적절한 중첩
  - 잘못된 루프 종료
  - 루프 변수를 증가시키는 것을 잊거나 잘못 증가시킴
  - 배열 요소를 잘못 인덱스하는 것

16.2 루프의 제어

- 루프를 블랙박스로 보자!

```
while ( !inputFile.EndOfFile() && moreDataAvailable ) {
    [redacted]
}
```

- while (true) ... break 기법을 사용하면 종료 조건이 안쪽에 위치하므로 이득이 없음

16.2 루프의 제어

- 루프 진입하기
  - 한 위치에서만 루프에 진입
  - 시작하기 직전에 초기화 코드를 입력
  - 무한 루프는 while (true) 를 사용

```
#define FOREVER    for (;;)
...
FOREVER {
    ...
}
```

    - 위와 같이 대체하는 기법을 찾지 말 것
- 적절할 때 for 루프를 선택
  - incremental index access

16.2 루프의 제어

- 루프 진입하기
  - while 루프가 더 적절할 때 for 루프를 사용하지 않음

```
// read all the records from a file
for ( inputFile.MoveToStart(), recordCount = 0; !inputFile.EndOfFile();
   recordCount++ ) {
   inputFile.GetRecord();
}
```

```
recordCount = 0;
for ( inputFile.MoveToStart(); !inputFile.EndOfFile(); inputFile.GetRecord() ) {
   recordCount++;
}
```

```
// read all the records from a file
inputFile.MoveToStart();
recordCount = 0;
while ( !inputFile.EndOfFile() ) {
   inputFile.GetRecord( &inputRec[ recordCount ], MAX_CHARS );
   recordCount++;
}
```

16.2 루프의 제어

- 루프의 중간 부분 처리하기
  - 언제나 중괄호를 사용하여 묶을 것
  - 빈 루프를 피할 것

```
while ( ( inputChar = cin.get() ) != '\n' ) {
    ;
}
```

```
do {
    inputChar = cin.get();
} while ( inputChar != '\n' );
```

16.2 루프의 제어

- 루프의 중간 부분 처리하기
  - 설것이 작업은 루프의 시작이나 끝에 배치

```
stringIndex = 1;
totalLength = 0;
while ( !inputFile.EndOfFile() ) {
   // do the work of the loop
   inputFile >> inputString;
   strList[ stringIndex ] = inputString;
   ...

   // prepare for next pass through the loop--housekeeping
   stringIndex++;
   totalLength = totalLength + inputString.length();
}
```

  - 루프가 하나의 기능만 수행

16.2 루프의 제어

- 루프의 종료
  - 종료 확인
  - 조건을 명확하게
  - 종료하기 위해서 인덱스를 조작하지 말 것

```
for ( int i = 0; i < 100; i++ ) {
   // some code
   ...
   if ( ... ) {
      i = 100;
   }

   // more code
   ...
}
```

16.2 루프의 제어

- 루프의 종료
  - 마지막 인덱스 값에 의존하는 코드는 피할 것
    - 플래그를 이용하여 재작성

```
for ( recordCount = 0; recordCount < MAX_RECORDS; recordCount++ ) {
   if ( entry[ recordCount ] == testValue ) {
      break;
   }
}
// lots of code
...
if ( recordCount < MAX_RECORDS ) {
   return( true );
}
else {
   return( false );
}
```

16.2 루프의 제어

- 루프의 종료
  - 안전한 카운터 사용

```
do {
   node = node->Next;
   ...
} while ( node->Next != NULL );
```

```
safetyCounter = 0;
do {
   node = node->Next;
   ...
   safetyCounter++;
   if ( safetyCounter >= SAFETY_LIMIT ) {
      Assert( false, "Internal Error: Safety-Counter Violation." );
   }
   ...
} while ( node->Next != NULL );
```

16.2 루프의 제어

- 루프의 종료
  - 루프 일찍 종료하기
    - break / Exit-For / Exit-Do / goto
    - 탈출하기 위해 플래그를 사용하지 말고 break 이용
    - break는 최소로, 구조적으로 사용

```
do {
   ...
   switch
      ...
      if () {
         ...
          break;
         ...
      }
      ...
} while ( ... );
```

16.2 루프의 제어

- 루프의 종료
  - 루프 일찍 종료하기
    - 루프 안 앞쪽에서 테스트를 위해 continue 사용

```
while ( not eof( file ) ) do
   read( record, file )
   if ( record.Type <> targetType ) then
      continue

   -- process record of targetType
   ...
end while
```

    - 언어가 continue를 지원하지 않으면 레이블 break 이용

```
do {
   ...
   switch
      ...
      CALL_CENTER_DOWN:
      if () {
         ...
          break CALL_CENTER_DOWN;
         ...
      }
      ...
} while ( ... );
```

16.2 루프의 제어

- 루프의 종료
- 루프 일찍 종료하기
  - break와 continue를 신중하게 사용
    - brea를 사용하면 루프를 블랙박스로 취급할 수 없음
    - 읽기 어려워짐
    - 심사숙고해서 사용할 것

16.2 루프의 제어

- 종결점 확인
  - 루프 설계 시, 첫번째 경우, 임의로 선택된 중간 경우, 마지막 경우를 따져 볼 것
- 비효율적인 프로그래밍 방식
  - < 기호를 <=로 바꾼다든가
  - 루프 인덱스에 1을 더하거나 빼거나
  - 우연에 의존
  - 오류를 더 복잡하게 만들고 감추는 부작용 발생

16.2 루프의 제어

- 루프 변수 사용
  - 루프 한계를 위해 서수나 열거형을 사용
    - 부동소수점은 위험
  - 의미있는 변수 이름 사용

```
for ( int payCodeIdx = 0; payCodeIdx < numPayCodes; payCodeIdx++ ) {
   for (int month = 0; month < 12; month++ ) {
      for ( int divisionIdx = 0; divisionIdx < numDivisions; divisionIdx++ ) {
         sum = sum + transaction[ month ][ payCodeIdx ][ divisionIdx ];
      }
   }
}
```

```
for ( i = 0;  i < numPayCodes; i++ ) {
   // lots of code
   ...
   for ( j = 0; j < 12; j++ ) {
      // lots of code
      ...
      for ( i = 0; i < numDivisions; i++ ) {
         sum = sum + transaction[ j ][ i ][ k ];
      }
   }
}
```

16.2 루프의 제어

- 루프 변수 사용
  - 인덱스 변수 범위를 루프 자체로 제한
    - Ada 설계에서는 엄격하게 제한
    - C++과 Java는 비슷하게

```
for ( int recordCount = 0; recordCount < MAX_RECORDS; recordCount++ ) {
   // looping code that uses recordCount
}

// intervening code

for ( int recordCount = 0; recordCount < MAX_RECORDS; recordCount++ ) {
   // additional looping code that uses a different recordCount
}
```

  - 다만, C++ 컴파일러 구현에 따라 다르게 동작함

16.2 루프의 제어

- 루프가 얼마나 길어야 할까?
  - 한 눈에 볼 수 있도록
  - 중첩을 3수준으로 제한
  - 긴 루프의 내부 루프를 루틴으로 이동
  - 길이가 긴 루프는 특히 명료하게 작성

16.3 루프를 쉽게 작성하는 방법 – 안에서부터 밖으로

- 보험회사의 총 보험료 계산 루틴 작성
- 1단계

```
-- get rate from table
-- add rate to total
```

- 2단계

```
rate = table[ ]
totalRate = totalRate + rate
```

- 3단계

```
rate = table[ census.Age ][ census.Gender ]
totalRate = totalRate + rate
```

- 4단계

```
For person = firstPerson to lastPerson
   rate = table[ census.Age, census.Gender ]
   totalRate = totalRate + rate
End For
```

16.3 루프를 쉽게 작성하는 방법 – 안에서부터 밖으로

- 5단계

```
For person = firstPerson to lastPerson
   rate = table[ census[ person ].Age, census[ person ].Gender ]
   totalRate = totalRate + rate
End For
```

- 6단계

```
totalRate = 0
For person = firstPerson to lastPerson
   rate = table[ census[ person ].Age, census[ person ].Gender ]
   totalRate = totalRate + rate
End For
```

16.4 루프와 배열의 대응

- 루프는 배열을 위해 존재

```
for ( int row = 0; row < maxRows; row++ ) {
   for ( int column = 0; column < maxCols; column++ ) {
      product[ row ][ column ] = a[ row ][ column ] * b[ row ][ column ];
   }
}
```

```
Product <- a x b
```

- 특정 언어에서는 그렇지 않음
  - APL