C언어와 PascAl언어 비교

각 언어의 특징 별 비교



학 과 : 컴퓨터공학부 컴퓨터공학전공

과 목 : 프로그래밍언어론

교수님 : 정목동 교수님

학 번 : 202030494

이 름 : 윤원재

제출일 : 2022년 4월 6일

목차

1. 개요 3

1.1. Pascal 언어 3

1.2. C 언어 3

2. 언어의 구성요소 4

2.1. 기본 문법 4

2.2. 주석 5

2.3. 식별자 5

2.4. 상수 8

2.5. 연산자 11

2.6. 자료형 13

3. 제어문 16

3.1. 조건문 16

3.2. 반복문 18

3.3. Goto문 21

4. 데이터 구조 22

4.1. 배열형 22

4.2. 구조체형 23

4.3. 열거형 25

4.4. 집합형 25

5. 포인터 26

5.1. 포인터의 개념 26

5.2. 포인터형의 선언 27

5.3. 포인터형의 연산 30

6. 참고문헌 32

1. **개요**
   1. **Pascal 언어**

1970년에 스위스의 컴퓨터 과학자 니클라우스 비르트(Niklaus Wirth)가 개발한 프로그래밍 언어이다. 깔끔한 문법 설계로 구조적 프로그래밍의 개념을 설명하기 쉽다 보니 교육용 언어로써 활용도가 높아 1990년대까지 교육현장에서 가장 널리 사용된 언어이다. 이러한 유명세를 통해 언어를 만든 니클라스우스 비르트는 1984년에 컴퓨터과학 분야의 노벨상이라 할 수 있는 튜링상을 수상하게 된다.

Pascal은 ALGOL에서 파생된 언어이다. 니클라우스 비르트는 1960년에 개발된 언어인 ALGOL 60을 개선하기 위하여 ALGOL W를 개발하였으나 인정받지 못하였다. ALGOL 60의 후속버전으로 ALGOL 68이 출시되었으나 엄격한 구문과 언어 정의의 복잡성 때문에 널리 사용되지 못하였고, 니클라우스 비르트는 ALGOL W를 개선하여 Pascal을 개발하게 된다. 절차 지향 언어이며, 자료구조와 구조적 프로그래밍, 명령형 프로그래밍을 특징으로 하고있다.

다음은 Pascal에서 ‘Hello, Woarld!’를 출력하는 예시이다.

**program** HelloWorld(output);

**begin**

writeln('Hello, World!')

**end**.

* 1. **C 언어**

1972년에 벨 연구소에서 데니스 리치(Dennis Ritchie)가 유닉스 운영체제에 사용하기 위해 개발한 프로그래밍 언어이다. 1969년에 벨 연구소에서 개발된 B언어를 개선하여 개발되었으며, 유닉스를 기반으로 하는 프로그램, 다양한 운영체제의 커널들이 C언어로 만들어졌다. 시스템 프로그램의 개발 언어로 큰 각광을 받았으며, 1989년에 ANSI C가 표준 채택 되었고, 1990년에 ISO가 ANSI C를 ISO C로 표준 채택 하며 표준화하였다. Pascal과 마찬가지로 절차 지향 언어이며, 명령형 프로그래밍을 특징으로 하고있다.

다음은 C에서 ‘Hello, World!’를 출력하는 예시이다.

**#include <stdio.h>**

**int main() {**

printf("Hello World!");

return 0;

**}**

1. **변수 및 상수**
   1. **기본 문법**

(1) pascal

Pascal은 작성된 프로그램의 여러 문장을 세미콜론(;)으로 구분한다. 이때 세미콜론(;)을 문장종료자라고 하며 문장장료자가 나타나지 않을 시 어떠한 내용이나 공백이 오더라도 그 문장은 종료되지 않은 것으로 판단한다. 이러한 문장들은 블록으로 구성되는데 프로그램 내에서 Begin~end. 안에 적혀있는 내용을 컴퓨터가 인식하여 지정한 동작을 수행하게 된다. 다음은 Pascal 프로그램의 예시이다.

**program** 프로그램이름(Input, Output);

**begin**

*//본문*

**end**.

program 뒤에 프로그램이름을 작성하고 구 컴파일러는 프로그램이름 옆에 (Input, Output)을 꼭 적어야 했지만, 현대의 컴파일러는 생략을 하여도 컴파일러가 자동으로 이를 채워준다. begin과 end.는 c언어에서의 main함수와 같은 역할을 수행한다.

(2) C언어

C언어도 작성된 프로그램의 여러 문장을 세미콜론(;)으로 구분한다. C언어 역시 블록으로 구성되는데 프로그램 내에서 {~}안에 적혀있는 문장들을 컴퓨터가 인식하여 명령을 수행하게 된다. 이때 {~} 앞에는 함수명과 함께 함수 자료형을 적어야 한다. 다음은 C 프로그램의 예시이다.

**#include <stdio.h>**

**int main(void)**

**{**

**// 본문**

**}**

#include은 전처리기 중 하나이다. 전처리기는 다른 프로그래밍 언어와 차별되는 C언어의 큰 특징 중 하나로, 프로그램을 컴파일하면 컴파일러는 전처리기의 내용을 먼저 처리한 다음 실행 프로그램을 만들게 된다. #include는 지정된 파일을 현재의 파일에 포함시키는 지시자이다. 위의 프로그램을 예시로 보면 stdio.h라는 헤더파일을 현재의 파일에 포함시키는 의미가 된다. stdio.h에는 표준입출력과 관련된 라이브러리 함수의 원형과 매크로 등이 선언 되어있다. #include <stdio.h>를 통해 현재의 파일에서 표준입출력과 관련된 함수나 명령어를 사용할 수 있다.

int는 자료형을 의미하며 이는 나중에 다룰 것이다. main()은 메인 함수를 의미하며 프로그램을 실행하면 메인 함수에 적혀있는 명령어나 함수 호출 등이 실행되게 된다.

* 1. **주석**

보통 주석문은 문서화 도구로 사용되며, 프로그램이 어떻게 동작하고 사용되는지를 설명하는 데 목적을 둔다.

(1) pascal

다중 줄 주석을 하는 방법과 줄 단위 주석을 하는 방법이 있다. 우선 다중 줄 주석을 할 때에, 괄호와 별표로 주석 처리를 할 수 있다.

**(\* 주석문 \*)**

또는 중괄호를 사용하여 주석 처리를 할 수 있다. 이 때, 컴파일러 지시자와 충돌을 일으킬 수 있으므로 주의하여야한다.

**{ 주석문 }**

줄 단위 주석을 하는 방법은 //를 사용한다. //이 사용된 시점에서 그 줄 끝까지를 하나의 주석으로 처리한다. 예를 들어 아래와 같이 ‘a := 10; // a = 10’ 이라고 하면 ‘a := 10;’은 컴파일러가 명령으로 수행을 하고 ‘// a = 10’은 주석으로 처리한다.

**a := 10; // a = 10**

(2) C언어

C언어 역시 다중 줄 주석을 하는 방법과 줄 단위 주석을 하는 방법이 있다. 다중 줄 주석을 할 때에, 슬래시와 별표로 주석처리를 할 수 있다.

**/\* 주석문 \*/**

줄 단위 주석을 하는 방법은 Pascal과 똑같은 방식인 //를 사용한다. //이 사용된 시점에서 그 줄 끝까지를 하나의 주석으로 처리한다. 예를 들어 아래와 같이 ‘a = 20; // a = 20’ 이라고 하면 ‘a = 20;’은 컴파일러가 명령으로 수행을 하고 ‘// a = 20’은 주석으로 처리한다.

**a = 20; // a = 20**

* 1. **식별자**

Pascal과 C언어의 식별자 규칙 중에서 공통적인 부분을 하나로 통합하여 설명하고자 한다. 식별자는 프로그램의 객체인 변수, 함수 등에 유일한 이름을 부여하기위해 사용되며, 영문자, 숫자, 특수 문자인 밑줄 문자(\_)로 구성된다. 이 때 무조건 문자 또는 밑줄 문자로 시작해야 하며, 영문자에서 대문자와 소문자는 구별된다.

(1) Pascal

Pascal의 식별자는 반드시 알파벳으로 시작해야하며, 63자로 글자수 제한이 존재한다. 또한 예약어를 사용할 수 없다. 다음은 Pascal에서 사용되는 예약어다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| and | end | interface | raise | uses |
| array | except | is | record | var |
| as | export | label | repeat | while |
| asm | file | library | resourcestring | with |
| begin | finalization | mod | set | xor |
| case | finally | nil | shl |  |
| class | for | not | shr |  |
| const | function | object | string |  |
| constructor | goto | of | then |  |
| destructor | if | or | threadvar |  |
| dispinterface | implementation | out | to |  |
| div | in | packed | try |  |
| do | inherited | procedure | type |  |
| downto | initialization | program | unit |  |
| else | inline | property | until |  |

<표 2-1> Pascal에서 사용되는 예약어

지시어는 식별자로 사용이 가능하나 권장되지는 않는다. 다음은 Pascal에서 사용되는 지시어다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| absolute | dynamic | message | private | resident |
| abstract | export | name | protected | safecall |
| assembler | external | near | public | stdcall |
| automated | far | nodefault | published | stored |
| cdecl | forward | overload | read | varargs |
| contains | implements | override | readonly | virtual |
| default | index | package | register | write |
| deprecated | library | pascal | reintroduce | writeonly |
| dispid | local | platform | requires |  |

<표 2-2> Pascal에서 사용되는 지시어

예약어에 대해 한가지 더 추가하자면, Pascal에선 <표 2-1>에 있는 예약어를 제외하고 사용자가 임의로 예약어를 추가할 수 없다.

(2) C언어

C언어는 Pascal보다는 그나마 자유로운 식별자 규칙을 가지고 있다. 알파벳이나 밑줄 문자로 시작되어야 하며 글자 수의 제약을 받지 않는다. 다만, 컴파일러에 따라서 글자 수의 제약이 있을 수 있다. 다음은 C언어에서 식별자 규칙에 맞는 식별자의 예시다.

**d**

**\_id**

**identifier12**

**This\_is\_identifier121**

두번째 ‘\_id’의 경우 Pascal에서는 사용이 불가능하다.

Pascal에서 예약어를 식별자로 사용할 수 없었던 것처럼 C언어 역시 키워드를 식별자로 사용할 수 없다. C언어에서는 예약어와 키워드가 같기때문에 같은 의미로 통용된다. 아래 표는 C언어에서 사용되는 키워드다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| auto | do | goto | signed | unsigned |
| break | double | if | sizeof | void |
| case | else | int | static | volatile |
| char | enum | long | struct | while |
| const | extern | register | switch |  |
| continue | float | return | typedef |  |
| default | for | short | union |  |

<표 2-3> C언어에서 사용되는 키워드

키워드 뿐만 아니라 라이브러리 함수명, main, 미리 정의된 상수명 등도 사용해선 안된다. 예를 들어 printf의 경우 표준 입출력 라이브러리에 포함되어있는 함수이므로 printf라는 이름의 식별자를 사용해선 안된다.

(3) 식별자 정의

그렇다면 식별자는 왜 문자(c언어의 경우 밑줄 문자 포함)로 시작되어야 하는 것일까? 이는 언어 문법이 이렇게 정의되어있기 때문이다. 아래는 C언어의 식별자 문법의 정의이다.

***identifier*:  
    *nondigit*  
    *identifiernondigit*  
    *identifierdigit***

***nondigit: 다음 중 하나  
    \_ a b c d e f g h i j k l mn o p q r s t u v w x y z  
    A B C D E F G H I J K L MN O P Q R S T U V W X Y Z***

***digit: 다음 중 하나  
    0 1 2 3 4 5 6 7 8 9***

nondigit는 알파벳과 밑줄 문자(\_)로, digit는 숫자로 구성되어있다. 식별자(identifier)는 nondigit, 식별자+nondigit, 식별자+digit로 정의된다. 결국 식별자는 알파벳이나 밑줄 문자로 시작 되어야하고 숫자 앞에는 반드시 식별자 또는 nondigit가 와야 한다고 문법적으로 정의가 되어있는 것이다. 따라서 식별자는 숫자로 시작될 수 없다.

* 1. **상수**

(1) Pascal

const를 사용하여 상수 선언을 할 수 있다.

**program** exConst

**const**

pi = 3.141592;

valid\_age = 25;

p = NIL;

character = ‘+’;

name = ProgrammingLanguages’

**begin**

**end.**

위의 코드처럼 상수 값은 자연수, 실수, 문자열, 포인터 등으로 꼭 숫자일 필요는 없다. 아래의 코드는 상수를 실제로 사용하는 코드이다.

**program** CircleArea

**const**

PI = 3.14;

**var**

R, S : real;

**begin**

writeln(‘반지름 값을 입력하시오’);

read(R);

S := R \* R \* PI;

writeln(‘원 넓이 :’ , S);

readln;

**end.**

3.14가 저장되어있는 PI라는 상수와 사용자 입력을 받은 R이라는 변수를 통해 원넓이 S를 구하는 코드이다. 이 코드를 통해 알 수 있는 점은 상수와 변수의 쓰임이 같다는 것이다.

(2) C언어

앞서 소개한 Pascal의 상수 선언처럼 사용하기 위해선 #define이라는 전처리기를 사용한다.

**#include <stdio.h>**

**#define** PI 3.14;

**int main(void)**

**{**

**float** r, s;

**printf(**“반지름 값을 입력하시오.”**);**

**scanf(**“%f”, &r**);**

s = r \* r \* PI;

**printf(**“원 넓이 : %.2f \n”, s**);**

**return 0;**

**}**

원 넓이를 구하는 Pascal의 코드를 C언어에 맞게 바꾸어 보았다. #define을 통해 PI라는 상수가 3.14로 된 것을 확인할 수 있다. C언어에서는 const 키워드 또한 사용할 수 있다.

**const int a = 4;**

이 문장은 int 자료형인 a에 4라는 값을 넣은 후 변경하지 않겠다는 뜻이다. 다시 말해 a값은 변경이 불가능하며, 변경이 되면 오류를 반환하게 된다.

뿐만 아니라 C언어는 정수 상수, 문자 상수, 열거 상수 등이 있다. 정수 상수는 8진 정수 상수, 10진 정수 상수, 16진 정수 상수로 구분된다. 8진 정수 상수는 0으로 시작하고 16진 정수 상수는 0x로 시작한다.

**printf(“%d, %#x, %#o” , 017, 017, 017); // 각각 15, Oxf, 017이 출력**

문자 상수는 작은따옴표로 둘러싸인 한 개의 문자이다. ‘a’, ‘1’, ‘+’ 처럼 표현하며 키보드에 없어 표현할 수 없는 문자 상수는 탈출 기법을 사용하여 표현한다.

|  |  |
| --- | --- |
| 탈출 문자 | 의미 |
| \a | 경고 |
| \\ | 역슬래시 |
| \b | 백스페이스 |
| \r | 캐리지 리턴 |
| \” | 큰따음표 |
| \f | 폼피드 |
| \t | 수평 탭 |
| \n | 개행 |
| \0 | 널 문자 |
| \’ | 작은따옴표 |
| \v | 수직 탭 |

<표 2-4> C언어에서 사용되는 탈출 문자

**printf(**“재미있는\n \”프로\b로그래이\b밍언어론\” 수업!!”**);** // **재미있는**

// **“프로그래밍 언어론” 수업**이 출력된다.

위 코드는 탈출 문자 사용의 예시이다. \n으로 개행(줄바꿈)이 이루어지고 \”로 큰따옴표를 표현하며 \b의 사용으로 ‘로’와 ‘이’가 지워졌음을 알 수 있다. 탈출 문자를 소개하면서 문자열 상수를 사용하였는데 위 코드처럼 작은따옴표가 아닌 큰따옴표로 싸인 문자을 문자열 상수로 표현한다.

끝으로 열거 상수는 enum 키워드를 사용하여 항을 열거하며 표현한다.

**enum day {**MON, TUE, WEN, THU, FRI, SAT, SUN**} d1;**

day라는 태그 이름으로 각 요일이 열거 되어있고 이를 변수 d1로 표현한다. d1은 enum day형 변수가 되는 것이다.

**d1 = TUE;**

다음과 같이 d1에 TUE를 배정하게 되면 TUE는 두번째에 있으므로 2가 배정된다.

* 1. **연산자**

Pascal과 C언어 간의 중복되는 부분은 한번에 설명하고 차이가 있는 부분은 그때마다 설명하고자 한다. 연산자는 대입 연산자, 부호 연산자, 산술 연산자, 논리 연산자, 비트 연산자, 비교 연산자 등이 있다.

(1) 대입 연산자

Pascal은 대입을 하기 위해 ‘:=’를 사용한다. a라는 변수에 5라는 값을 대입할 경우 ‘a := 5;’로 표현한다. C언어는 대입을 하기 위해 ‘=’을 사용한다. 앞서 말한 ‘a:=5;’ 를 C언어에선 ‘a=5;’로 표현한다.

(2) 부호 연산자

Pascal과 C언어에서 사용되는 부호 연산자는 동일하다. 양수인 경우 ‘+’를, 음수인 경우 ‘-‘를 사용하여 표현한다. 양수 5는 ‘+5’, 음수 5는 ‘-5’로 표현한다.

(3) 산술 연산자

덧셈, 뺄셈, 곱셈은 동일하나 나눗셈에서 차이를 보인다. 우선 덧셈, 뺄셈, 곱셈은 각각 ‘+’, ‘-‘, ‘\*’를 사용하며 ‘2+3’, ‘a-b’, ‘a\*5’ 와 같이 사용된다. 나눗셈의 경우 pascal에서는 ‘/’, ‘div’, ‘mod’를 사용하여 표현한다. ‘/’의 경우 실수의 나머지를 표현하며, ‘div’는 몫을, ‘mod’는 나머지를 구하는 연산자이다.

**var**

int: Integer;

float: real;

**begin**

int := 26 div 7; // int = 3

int := 26 mod 7; // int = 5

float := 10 / 4; // float = 2.5

{int := 10 / 2;} // /연산자는 실수값을 반환하므로 대입할 수 없다.

int := trunc( 26 / 7 ); // int = 3, 소수점 이하 버림

**end.**

/연산자는 실수를 반환하므로 실수형 변수에 대입하여야 하며, trunc을 사용하면 실수형 연산에서 소수점 이하를 버릴 수 있다.

C언어에서 나눗셈 연산은 ‘/’와 ‘%’를 사용한다.

**#include <stdio.h>**

**int main(void)**

**{**

printf(“5 / 4 = %d”, 5 / 4); // 출력 : 5 / 4 = 1

printf((“5 %% 4 = %d”, 5 % 4); // 출력 : 5 % 4 = 1

printf(“5 / 4 = %f”, 5 / 4); // 출력 : 5 / 4 = 1.25000

return 0;

**}**

‘/’ 는 몫을 반환하며, ‘%’는 나머지를 반환한다. 이때, ‘/’의 연산에서 피연산자를 정수가 아닌 실수로 계산을 하면 소수점 아래까지 표현된다는 것을 알 수 있다.

(4) 논리 연산자

Pascal에서 논리연산자를 통한 결과는 항상 Boolean형이며, 피연산자 또한 Boolean형이여야 한다. ‘and’, ‘or’, ‘not’, ‘xor’이 사용되며 아래의 표와 같다.

|  |  |
| --- | --- |
| 논리 연산자 | 의미 |
| and | 논리곱, 양쪽이 참이여야 true값을 반환 |
| or | 논리합, 둘 중 하나만 참이여도 true값을 반환 |
| not | 부정, true일 경우 false값을, false일 경우 true값을 반환 |
| xor | 베타 논리합, 양쪽이 다를 경우에만 true값을 반환 |

C언어에서의 논리연산자는 꼭 Boolean형 피연산자로 계산될 필요가 없으며, 정수 1과 정수 0을 각각 true와 flase로 취급하여 사용한다. ‘%%’, ‘||’, ‘!’이 사용되며 아래의 표와 같다.

|  |  |
| --- | --- |
| 논리 연산자 | 의미 |
| && | 논리곱, 양쪽이 참이여야 true값을 반환 |
| || | 논리합, 둘 중 하나만 참이여도 true값을 반환 |
| ! | 부정, true일 경우 false값을, false일 경우 true값을 반환 |

(5) 비트 연산자

Pascal에서 논리연산자는 항상 Boolean형을 사용해야한다고 하였는데, Boolean형이 아닌 정수를 사용하게 되면 비트 연산자로 동작하게 된다. 또한, 시프트 연산을 지원하는데 ‘shl’, ‘shr’를 사용하여 수행한다.

|  |  |
| --- | --- |
| 비트 연산자 | 의미 |
| and | 비트 논리곱, 양쪽이 정수의 논리곱을 반환 |
| or | 비트 논리합, 양쪽 정수의 논리합 반환 |
| not | 비트 부정, 오른쪽 정수의 보수를 반환 |
| xor | 비트 베타 논리합, 양쪽 정수의 베타 논리합을 반환 |
| shr | 비트 오른쪽 밀기, 왼쪽 정수를 오른쪽 자연수 비트 만큼 오른쪽으로 밀어낸다. |
| shl | 비트 왼쪽 밀기, 왼쪽 정수를 오른 쪽 자연수 비트 만큼 왼쪽으로 밀어낸다. |

C언어에서 비트 연산자는 ‘&’, ‘|’, ‘~’, ‘^’연산자가 사용된다. C언어 역시 시프트연산을 지원하는데 ‘>>’ , ‘<<’를 사용하여 수행한다.

|  |  |
| --- | --- |
| 비트 연산자 | 의미 |
| & | 비트 논리곱, 양쪽이 정수의 논리곱을 반환 |
| | | 비트 논리합, 양쪽 정수의 논리합 반환 |
| ~ | 비트 부정, 오른쪽 정수의 보수를 반환 |
| ^ | 비트 베타 논리합, 양쪽 정수의 베타 논리합을 반환 |
| >> | 비트 오른쪽 밀기, 왼쪽 정수를 오른쪽 자연수 비트 만큼 오른쪽으로 밀어낸다. |
| << | 비트 왼쪽 밀기, 왼쪽 정수를 오른 쪽 자연수 비트 만큼 왼쪽으로 밀어낸다. |

(6) 비교 연산자

Pascal과 C언어의 비교연산자는 같다. 단, 등가 기능에 대해선 차이가 있다.

|  |  |
| --- | --- |
| 비교 연산자 | 의미 |
| = | (Pascal) 왼쪽과 오른쪽 값이 같으면 true값을 반환 |
| == | (C언어) 왼쪽과 오른쪽 값이 같으면 true값을 반환 |
| > | 왼쪽의 값이 오른쪽 값보다 크면 true값을 반환 |
| < | 왼쪽의 값이 오른쪽 값보다 작으면 true값을 반환 |
| => | 왼쪽의 값이 오른쪽 값보다 크거나 같으면 true값을 반환 |
| =< | 왼쪽의 값이 오른쪽 값보다 작거나 같으면 true값을 반환 |

위의 표와 같이 Pascal에서 등가 기호는 ‘=’이며, C언어에서 등가기호는 ‘==’이다. C언어에서 ‘=’기호는 대입 연산자이다.

* 1. **자료형**

(1) 변수의 자료형 설정

Pascal에서 변수의 자료형 설정은 다음과 같다.

**var**

int: Integer; // Integer자료형의 int 변수 선언

float: real; // real자료형의 float 변수 선언

var을 통해 변수 선언을 하며, 변수 선언을 할 때 ‘변수명: 자료형’의 형식으로 자료형을 지정할 수 있다.

C언어에서의 자료형 선언은 다음과 같다

**int** num1; // int형의 변수 num1 선언

**float** num2; // float형의 변수 num2 선언

자료형 뒤에 변수명을 입력하여 해당 변수의 자료형을 설정한다..

(2) 정수형

Pascal에서 사용되는 정수형은 ‘Integer’와 ‘Cardinal’이 있다. ‘Integer’은 일반적인 정수를 표현하며, ‘Cardinal’은 0 이상의 정수, 다시 말해, 0을 포함한 양의 정수를 표현한다. 선언이나 연산으로 ‘Cardinal’ 자료형 변수가 음수를 표현하게 되면 컴파일 오류가 발생한다.

**var**

int1: Integer;

int2: Cardinal;

그 밖에 정수를 표현하는 자료형은 아래 표와 같다.

|  |  |  |
| --- | --- | --- |
| Pascal 정수 자료형 | 크기 | 범위 |
| Byte | 1바이트 | 0~255 |
| word | 2바이트 | 0~65535 |
| Shortint | 1바이트 | -128 ~ 127 |
| Integer | 2바이트 | -32768 ~ 32768 |
| Longint | 4바이트 | -2147483648 ~ 2147483647 |

<표 2-5> Pascal에서 사용되는 정수 자료형

C언어에서 사용되는 정수형은 ‘char’, ‘Int’, ‘long’, ‘short’ 등이 있다. 부호 키워드(signed, unsigned)와 자료형 크기(int, long 등)으로 표현하며 ‘sighned’는 생략이 가능하다. 다음은 C언어에서 사용하는정수 자료형을 정리한 표이다.

|  |  |  |  |
| --- | --- | --- | --- |
| C언어 정수 자료형 | 크기 | 범위 | 비고 |
| char  singed char | 1바이트, 8비트 | -128~127 |  |
| unsigned char | 1바이트, 8비트 | 0~255 |  |
| short  short int | 2바이트, 16비트 | -32.768~32.767 | int 생략가능 |
| unsigned short  unsigned short int | 2바이트, 16비트 | 0~65.535 | int 생략가능 |
| int  signed int | 4바이트, 32비트 | -2.147.483.648~2.147.483,647 |  |
| unsigned  unsigned int | 4바이트, 32비트 | 0~4.294.967.295 | int 생략가능 |
| long  long int  signed long  signed long int | 4바이트, 32비트 | -2,147,483,648~2,147,483,647 | int 생략가능 |
| unsigned long long  unsigned long long int | 4바이트, 32비트 | 0~4.294.967.295 | int 생략가능 |
| long long  long long int  signed long long  signed long long int | 8바이트, 64비트 | -9.223.372.036.854.775.808~  9,223,372,036,854,775,807 | int 생략가능 |
| unsigned long long  unsigned long long int | 8바이트, 64비트 | 0~18.446.744.073.709.551.615 | int 생략가능 |

<표 2-6> C언어에서 사용되는 정수 자료형

int 자료형은 컴퓨터 CPU에 따라 크기가 달라질 수 있는데, 64비트 CPU의 경우 크기가 64비트이다.

(3) 문자형

Pascal과 C언어 모두 ‘char’를 사용하여 문자형을 표현한다. ‘char’는 1바이트(=8비트)의 크기를 가지며 1개의 문자를 가진다.

(4) 논리형

Pascal과 ‘Boolean’을 사용하여, C언어는 ‘bool’을 사용하여 논리형을 표현한다. 크기는 1바이트이며, Pascal의 ‘Boolean’에는 ‘true’ 또는 ‘false’라는 상수가, C언어의 ‘bool’에는 ‘true’ 또는 ‘false’라는 상수를 포함하여 1 또는 0을 넣을 수 있는데 1과 0은 각각 ‘true’와 ‘false’를 의미한다.

(5) 실수형

Pascal에서 사용되는 실수 자료형은 아래의 표와 같다.

|  |  |  |
| --- | --- | --- |
| Pascal 실수 자료형 | 크기 | 범위 |
| Real | 6바이트 | 소수점 이하 11~12자리 |
| Single | 4바이트 | 소수점이하 7~8자리 |
| Double | 8바이트 | 소수점 이하 15~16자리 |
| Extended | 10바이트 | 소수점 이하 19~20자리 |
| Comp | 8바이트 | 소수점 이하 19~20자리 |

<표 2-7>Pascal에서 사용되는 실수 자료형

C언어에서 사용되는 C언어 자료형은 아래의 표와 같다.

|  |  |  |
| --- | --- | --- |
| C언어 실수 자료형 | 크기 | 범위 |
| float | 4바이트 | ±3.4 ×10^-37이상 ± 3.4 × 10^38이하 |
| double | 8바이트 | ±1.7 ×10^-307이상 ± 3.4 × 10^308이하 |
| long double | 8바이트 이상 | double 이상의 표현범위 |

<표 2-8> C언어에서 사용되는 실수 자료형

1. **제어문**
   1. **조건문**

(1) Pascal

Pascal에서 조건문은 if문과 case문으로 표현할 수 있다. 우선 if문은 아래의 코드와 같다.

**if** boolean변수 **then**

명령

**else**

명령;

boolean형 변수가 true일 경우 해당 if문이 수행된다. 이때 세미콜론(;)이 마지막 명령 뒤에 붙어있는 것을 확인할 수 있는데, 이는 if문의 명령이 하나의 문장이기 때문이다. 다시 말해, if~세미콜론(;)까지가 하나의 문장으로 취급되는 것이다. 만약 else를 사용하지 않는다면 다음과 같이 표현한다.

**if** boolean변수 **then**

명령;

뿐만 아니라 여러가지 명령을 수행하고자 할 때, begin~end.를 사용하여 다음과 같이 표현한다.

**if** boolean변수 **then**

**begin**

명령1;

명령2;

명령3;

**end**

**else**

**begin**

명령1;

명령2;

명령3;

**end**;

Pascal의 case문은 주어진 조건과 일치할 경우, 작성된 명령을 수행하는 구문이다. case 뒤에 있는 서수형 자료를 비교하여 해당 경우에 맞게 수행된다. 이때 서수형은 정수형, 문자형, 논리형 등 이다. 만약 해당되는 경우가 없다면 else 뒤에 작성된 명령을 수행하게된다. if문과 마찬가지로 case~end까지를 하나의 문장으로 보기때문에 end뒤에 세미콜론(;)을 붙인다. 예시 코드는 아래와 같다.

**case** 서수형 **of**

**경우1 :**

명령1;

**경우2 :**

명령2;

**경우3 :**

명령3;

**else**

명령;

**end;**

(2) C언어

C언어에서 조건문은 ‘if문’, ‘switch문’이 있다. 우선 ‘if문’의 코드는 다음과 같다.

**if(조건식)**

**{**

명령1;

**} else if(조건식)**

**{**

명령2;

**} else**

**{**

명령3;

**}**

if 뒤 괄호의 조건식을 작성한다. 예를 들어 ‘a==5’라고 하였을 때, a의 값이 5이면 true로, 아니면 false로 인식한다. true일 경우 {}안의 명령어가 수행되며 false일 경우 else if 뒤 괄호의 조건식을 비교한다. else if가 아닌 else 뒤 {} 안의 명령은 앞에 작성된 if의 조건에 맞지 않을 경우 수행되는 명령이다.

switch문의 경우 다음과 같다.

**switch(조건식)**

**{**

**case 조건1:**

명령1;

break;

**case 조건2:**

명령2;

break;

**default:**

명령3;

**}**

switch 뒤에 있는 조건식의 값이 case 뒤 조건과 일치할 경우 각 case에 맞는 명령이 수행된다. 이 때, 각 조건에는 break문을 넣어야 하는데 break를 작성하지 않을 경우 원하는 case의 명령을 수행하고도 switch문을 빠져나오지않고 뒤의 코드가 계속 수행된다. 위의 코드에서 break를 모두 지운 경우엔 case 조건1, case 조건2, default의 모든 명령이 수행된다. switch문의 괄호에는 정수 수식이 와야 하고, case 다음에는 정수 상수 수식이 와야 한다.

* 1. **반복문**

(1) Pascal

Pascal에서 반복문은 ‘for문’, ‘while문’, ‘repeat-until문’으로 표현할 수 있다. 아래의 코드는 Pascal에서 쓰이는 ‘for문’이다.

**begin**

**for** 정수형지시자 := 시작 **to** 끝 **do**

명령;

**end**

정수형 변수 a를 지정한 다음 ‘a := 1 to 20 do’라고 하고 명령으로 a값을 출력하게 되면 1부터 20까지의 정수들이 출력된다. 이때 ‘to’를 ‘downto’라고 하고 1과 20의 자리를 바꿔서 ‘a := 20 downto 1 do’로 하게 되면 20부터 1까지의 정수들이 출력된다. 다시 말해, ‘to’는 증가를, ‘downto’는 감소를 의미한다. 이때 1씩 증가와 감소가 이루어지고 1이외의 차로는 for문을 설정할 수 없다.

다음은 ‘while문’이다.

**while** 조건식 **do**

명령;

조건식이 참일 경우 명령을 수행하게 된다. 조건식에 ‘a<100’(a는 Integer형 변수)를 넣고 a를 1씩 증가하는 명령을 작성하면 a가 1씩 증가하다가 100이 되면 while문을 탈출하게된다.

끝으로 ‘repeat-until문’이다.

**repeat**

명령1;

명령2;

**until** 조건식;

일단, 명령1과 명령2를 수행하고 until 뒤의 조건이 false라면 repeat과 until 사이의 명령들이 한번 더 수행된다. true일 경우 반복문을 빠져나가게 된다.

‘while문’과 ‘repeat-until문’의 차이점은 while문의 경우 조건을 비교하여 명령의 수행 여부를 결정하지만, ‘repeat-until문’의 경우 일단 명령을 수행하고 조건을 비교하는 차이가 있다. 또한 조건이 참일 경우 ‘while문’에선 명령이 수행되지만 ‘repeat-until문’에서는 반복문을 빠져나오게 된다. 끝으로 ‘while문’에서는 1개의 명령만 수행 가능하지만 repeat-until문’에서는 여러 개의 명령이 수행 가능하다.

(2) C언어

C언어에서 반복문은 ‘for문’, ‘while문’, ‘do-while문’으로 표현할 수 있다. 우선 for문의 코드는 다음과 같다.

**for(초기화; 조건식; 증감문){**

명령1;

명령2;

**}**

for뒤의 괄호 안에 3가지 형식의 수식이 있다. 첫번째는 초기화이다. for문 내에서 사용할 루프 변수를 초기화 하는 작업을 수행한다. 이 때, 사용할 변수는 for문을 만나기 전에 미리 선언 되어야한다. 두번째는 조건식이다. for문 안의 명령을 수행하기전에 조건식의 참, 거짓을 비교하여 참일 경우 명령을 수행하고 거짓일 경우 for문을 탈출하게 된다. 끝으로 증감문이다. 증감문은 for문의 명령이 수행되고 실행된다. 보통 루프 변수를 증가하거나 감소시키는데 사용한다. 다음은 for문의 예시이다.

**int i;**

**for(I = 0; i<10; i++)**

**{**

명령1;

명령2;

**}**

위의 코드는 i가 0으로 초기화되어 0부터 9까지, 다시 말해, 10번 수행되는 for문이다.

다음은 ‘while문’이다.

**while(**조건식)

**{**

명령1;

명령2;

**}**

while뒤의 괄호 안에 있는 조건식의 참, 거짓 여부를 비교하여 참일 경우 {}안의 명령을 수행한다. 이때, 조건식이 언제나 true인 경우 while문이 무한 반복을 하게 된다. 따라서 조건식은 항상 ‘true’가 되지않게 조절하여 작성할 필요가 있다.

끝으로 ‘do-while문’이다. 작동 원리는 Pascal의 ‘repeat-until문’과 동일하다.

**do**

**{**

명령1;

명령2;

**}while(**조건식**)**

Pascal과 C언어의 ‘while문’ 차이점은 명령의 개수 여부이다. Pascal의 ‘while문’은 명령 문장을 한 문장만 수행하는 반면 C언어의 ‘while문’은 {}안의 모든 명령을 수행하게 된다. Pascal의 ‘while문’에서 여러 문장의 명령을 수행하고 싶으면 ‘begin’과 ‘end’로 묶어 주어야한다.

(3) 공통 반복문 프로시저

Pascal과 C언어에서 공통적으로 ‘continue’와 ‘break’가 사용된다. 반복문을 실행하던 도중 ‘continue’를 만나면 ‘continue’ 다음 문장부터 반복문의 끝 문장까지의 모든 명령을 건너뛰고 다시 반복문의 조건을 검사하게 된다. 특정 조건에 따라 반복문 명령을 건너뛰고 싶을 때 유용하게 사용된다. 두번째로 반복문을 실행하던 도중 ‘break’를 만나게 되면 그 즉시 반복 조건과 상관없이 반복문을 빠져나오게 된다. 반복을 실행하다가 특정 조건을 만족할 경우 반복문을 빠져나오게 하고 싶을 때 사용된다. 이러한 ‘break’문은 C언어에서 ‘switch문’에서도 사용되며 앞서 사용의 예를 보인 바 있다.

* 1. **goto문**

(1) Pascal

goto문을 사용하기 위해선 지표 이름을 선언해야한다. 지표 이름은 ‘label’을 사용하여 선언한다.

**program exGoto;**

**label 20;**

**begin**

writeln('Programming');

**goto 20;**

writeln('Algorithms');

**20:**

writeln('Languages');

**end.**

위의 코드는 Pascal에서 goto문의 쓰임을 표현한 예시 코드이다. ‘label’을 사용하여 ‘20’이라는 지표를 설정하였다. 그 후 프로그램을 수행하며 ‘Programming’을 출력하고 ‘goto 20;’을 만나서 ‘20:’으로 이동한다. 그런 다음 ‘Languages’를 출력하고 프로그램이 종료된다. ‘goto 20;’으로 지표 이름 20까지 건너뛰었음으로 ‘Algorithms’의 출력은 수행되지 않는다.

(2) C언어

C언어의 경우 Pascal처럼 지표 이름을 선언하지는 않는다.

**#include <stdio.h>**

**int main(void)**

**{**

printf(“Programming”);

**goto read;**

printf(“Algorithms”);

**read:**

printf(“Languages”);

return 0;

**}**

위의 코드는 C언어에서 goto문의 쓰임을 표현한 예시 코드이다. C언어에서는 ‘goto 레이블’을 통해 레이블(label)을 지정한다. 이 레이블은 식별자라고 하는데 식별자 뒤에 콜론(:)을 붙이고 문장을 작성하면 이는 레이블 문장이 된다. 위의 코드에선 ‘read’가 레이블이고 ‘printf(“Languages”);’가 레이블 문장이 된다. 출력은 Pascal과 마찬가지로 ‘Algorithms’를 건너뛰어 ‘ProgrammingLanguages’가 출력된다.

goto문은 코드를 복잡하게 하기 때문에 Pascal에서나 C언어에서나 사용을 지양하는 편이다.

1. **데이터 구조**
   1. **배열형**

(1) Pascal

Pascal에서 배열을 사용하기 위해선 우선 배열형을 만들어야 한다.

**type**

배열형이름: **Array[**시작값..끝값**]** **of** 자료형;

----------- 예시 -----------------------------------

**type**

ArrInt: **Array[**1..10**]** **of** Integer;

위의 코드는 배열형 선언 코드와 그 예시이다. 예시와 같이 선언을 하면 ArrInt라는 배열형이 만들어지고 자료형은 정수형(Integer)로, 배열의 크기는 10이다. 배열형의 사용은 다음과 같다.

**type**

ArrInt: **Array[**1..10**]** **of** Integer;

**var**

nArr: ArrInt;

count: Integer;

**begin**

**for** count:=1 **to** 10 **do**

nArr[count] = 0;

**end.**

위의 코드는 ‘ArrInt’라는 배열형으로 ‘nArr’라는 10칸짜리 배열을 생성한 다음 각 배열에 0을 넣는 프로그램이다. 다차원 배열을 만드는 방법은 다음과 같다.

**type**

dim1arrInt = **Array**[1..10] **of** Integer;

dim2ArrInt1 = **Array**[1..10][1..10] **of** Integer;

dim2ArrInt2 = **Array**[1..10] **of** **Array** [1..10] **of** Integer;

dim2ArrInt3 = **Array**[1..10] **of** dim1ArrInt;

dim2ArrInt4 = **Array**[1..10, 1..10] **of** Integer;

(2) C언어

C언어는 Pascal보다는 간단하게 배열 선언이 가능하다. 아래 코드는 배열 선언의 예시이다.

int nArr[10]; // 자료형 배열이름[배열 크기]

nArr라는 이름의 10칸짜리 배열이 위의 코드를 통해서 생겼다. []안에 들어가는 숫자를 ‘인덱스’라고 하는데, 여기서 주의해야할 점은 C언어에서 배열의 인덱스는 0부터 시작한다는 점이다. 위의 코드로 10칸짜리 배열은 nArr[0], nArr[1], …, nArr[9]로 참조된다. 인덱스는 정수 수식으로도 사용 가능한데, nArr[1+3]로 접근을 하면 nArr[4]로 접근하는 것과 같다. 다음은 Pascal의 배열 예시를 C언어 코드로 표현한 것이다.

**#include <stdio.h>**

**int main(void)**

**{**

int i, nArr[10];

for(i = 0; i<10, i++){

nArr[i] = 0;

}

return 0;

**}**

위 코드를 실행하면 nArr[0], nArr[1], …, nArr[9]에 0을 넣는 명령을 수행한다. C언어에서 다차원 배열을 만드는 방법은 다음과 같다.

int dim2Arr[2][4]; // 자료형 배열이름[배열 크기][배열 크기] 2차원 배열

int dim3Arr[2][4][5]; // 자료형 배열이름[배열 크기][배열 크기][배열 크기] 3차원 배열

역시나 배열 인덱스는 0부터 시작한다. 따라서 위의 배열의 경우 dim2Arr[0][0]~dim2Arr[1][4], dim3Arr[0][0][0]~dim3Arr[1][3][4]가 참조된다.

* 1. **구조체형**

(1) Pascal

Pascal에서 구조체를 사용하기위해선 구조체형을 만들어야 한다.

**type**

이름 = **Record**

필드1 : 자료형;

필드2 : 자료형;

**end;**

----------- 예시 -----------------------------------

**type**

Student = **Record**

grade, age : Integer;

name : **Array[**0..8**]** **of** Char;

**end;**

위의 코드는 Pascal의 구조체형 선언 코드와 그 예시이다. Student라는 구조체를 만들고 그 안에는 grade, age라는 정수형 변수와 name이라는 문자형 배열이 있다. 위 코드로 선언된 구조체를 사용하기 위해선 변수 선언을 구조체형으로 하면 된다. 다시 말해, 구조체는 사용자가 만들어낸 자료형인 것이다.

**var**

Student1, Student2 : Student;

이렇게 하면 Student 구조체형인 Student1, Student2 변수가 만들어진다. 각 변수 내부에는 grade, age, name이 존재하며 이를 멤버 변수라고 한다. 멤버 변수에 접근하기 위해서는 점(.)을 사용하여 접근한다. 아래의 코드는 Student1의 멤버 변수 age에 23을 대입하는 코드이다.

Student1.age := 23;

(2) C언어

C언어에서 구조체는 ‘struct’를 사용하여 선언한다.

**struct** 구조체\_테그\_이름 {

자료형 멤버1;

자료형 멤버2;

} 변수명;

----------- 예시 -----------------------------------

**struct** Student {

int age, grade;

char name[8];

} student1, student2;

위의 코드는 C언어의 구조체형 선언 코드와 그 예시이다. student1, student2 변수가 Student 구조체형으로 선언되었으며 멤버 변수로는 정수형인 age, grade변수와 char형인 name[8] 배열이 있다. 구조체 변수 선언은 구조체형 선언과 분리하여 선언하여도 된다.

**struct** Student student1, student2; // Student형 변수 student1, student2 선언

Pascal과 마찬가지로 멤버 변수에 접근하기 위해서는 점(.)을 사용하여 접근한다.

student1.age = 23;

* 1. **열거형**

(1) 열거형

Pascal은 집합의 연산이 가능하여 열거형의 데이터 구조 사용하여 집합형를 표현할 수 있다. 집합형을 보기 전에 열거형을 우선 살펴보기로 한다.

**type**

day = (MON, TUE, WEN, THU, FRI, SAT, SUN);

**var**

today : day;

위의 코드는 열거형 타입인 day를 선언하고 today라는 변수를 day타입으로 선언 한 코드이다. today변수는 열거된 day값들 중 하나의 값을 가질 수 있다. 열거형을 사용하는 예는 아래 코드와 같다.

**type**

day = (MON, TUE, WEN, THU, FRI, SAT, SUN);

**var**

today : day;

**begin**

today := MON;

**end.**

(2) 부분 범위형(subrange형)

Pascal에서 사용하는 독특한 표현 방법이며 순서가 정해진 형(ordinal형)의 범위를 표현하는 데에 사용한다. 여기서 ordinal형은 정수형, 문자형, 논리형, 열거형, 부분 범위형이 있다. 부분 범위형은 ‘..’을 사용하여 표현하며 1부터 100까지를 표현하고 싶으면 ‘1..100’라고 하면 된다.

* 1. **집합형**

(1) Pascal

Pascal에서 사용하는 독특한 표현 방법이며 다음과 같이 사용한다.

**type**

이름 **= Set of** 자료형; //단 자료형의 선택 가능한 수는 256 이하여야 한다.

자료형의 수가 256 이하여야 하기때문에 ‘SmallInt’, ‘LongInt’ 등은 사용될 수 없다.

**type**

num **= Set of** 1..20;

**var**

set1, set2, set3 : num;

check : Boolean;

**begin**

set1 := {1, 2, 7, 10, 12, 16};

set2 := {3, 5, 7, 9, 10, 13, 16};

set3 := set1 + set2; // 합집합, set3 = [1, 2, 3, 5, 7, 10, 12, 13, 16]

set1 := set1 – set2; // 차집합, set1 = [1, 2, 12]

set1 := set1 + [7, 10, 16];

set3 := set1 \* set2; // 교집합, set3 = [7, 10, 16]

check := 3 in set1; // check = False

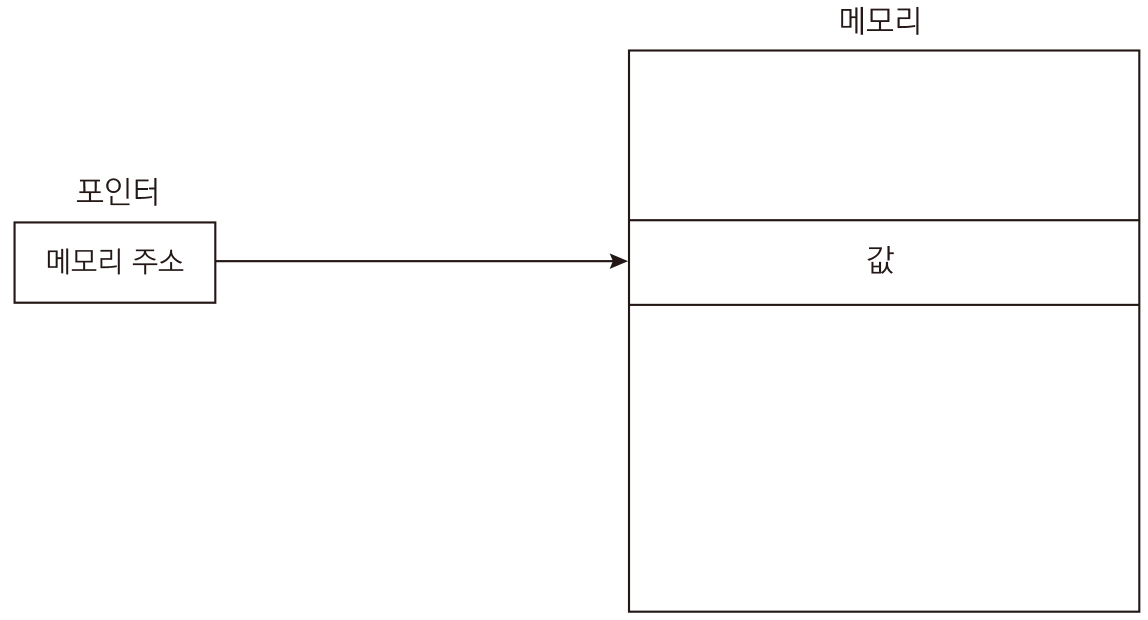
check := 3 in set2; // check = True

**end.**

Pascal의 집합형은 위의 코드와 같이 쓰인다. 주석으로 세부 설명을 하였으므로 별도의 설명은 붙이지 않겠다.

1. **포인터**
   1. **포인터의 개념**

포인터의 개념을 매우 자세하게 설명하면 따로 리포트를 작성해야 할 정도이니 여기선 간단하게 설명한다. 포인터는 변수의 메모리 주소를 가리키는 변수이다. 변수를 선언하게 되면 메모리에 변수의 공간이 생긴다. 이 공간은 변수의 자료형에 따라 크기가 지정되고 값이 저장된다. 이러한 공간은 정수값 주소를 가지고 있는데 이를 가리키는 것이 포인터이다. 포인터에는 메모리 주소가 저장되며 포인터의 연산은 메모리 주소의 연산과 같다. 예를 들어 int형 포인터 p가 선언되고 p에 메모리 주소 100이 있다고 치자. 포인터 p에 +1을 하게 되면, p의 결과는 101이 아니라 104가 된다. int형의 크기가 4바이트이기 때문이다.



<그림 5-1> 포인터와 메모리 구조

* 1. **포인터형의 선언**

(1) Pascal

Pascal에서 포인터는 다음과 같이 선언된다.

**type**

포인터\_이름 = ^기본\_변수\_자료형;

포인터는 다른 변수의 주소를 표현하는 변수이다. int형 변수 num의 주소를 Iptr라는 포인터 변수로 표현한다고 하면 다음과 같다.

**type**

iPtr = ^Integer;

**var**

p1 : iPtr

num : Integer;

**begin**

num := 100;

p1 := @num;

**end.**

정수형 변수인 num에 100 값을 넣고 포인터형 변수 p1이 변수 num을 가리키게 하였다. 포인터 변수 뒤에 캐럿(^)문자를 사용하면 역참조된다.

**type**

iPtr = ^Integer;

**var**

p1 : iPtr

num : Integer;

**begin**

num := 100;

p1 := @num;

p1^ := 200; // num := 200과 같다.

**end.**

Addr()함수를 사용하면 변수의 주소 또는 포인터가 가리키고 있는 주소를 얻을 수 있다.

**type**

iPtr = ^Integer;

**var**

p1 : iPtr

num : Integer;

**begin**

num := 100;

p1 := Addr(num);

p1^ := 200; // num := 200과 같다.

**end.**

할당 해야할 정확한 주소가 없는 경우엔 NIL포인터를 사용한다. 예를 들어 리스트를 표현할 때, 리스트의 끝이 알려지지 않은 경우 NIL포인터를 사용한다. 또한, 프로그램상의 수행 오류를 표현할 때도 사용된다. nil포인터는 특정 변수를 가리키는게 아닌, 아무 곳이나 가리키게 된다. 이때, nil포인터의 값을 출력하면 0으로 표현된다. nil포인터의 사용은 다음과 같다.

**type**

iPtr = ^Integer;

**var**

p1 : iPtr

num : Integer;

**begin**

num := 100;

p1 := nil;

y1 := Addr(p1);

writeln('p1의 값은 : ', y1^); // ‘p1의 값은 0’ 이 출력된다.

**end.**

(2) C언어

C언어에서 포인터는 다음과 같이 선언된다.

자료형 \*포인터변수이름;

포인터변수 = &변수

일반적으로 변수가 선언되는 형식에서 변수명 앞에 별표(\*)를 앞에 붙여 포인터 변수를 선언한다. 이러한 포인터 변수에 해당 변수의 주소를 배정하기 위해선 &기호를 사용한다. 아래의 코드는 C언어의 포인터 변수 사용 예시이다.

위 코드의 출력은 다음과 같다.

i의 주소 : 0x23a3fc4 // 0x23a3fc4는 i의 주소 (컴퓨터마다 다름)

i의 값 : 50

p의 값 : 0x23a3fc4

\*p의 값 : 50

**#include <stdio.h>**

**int main(void)**

**{**

int i = 50;

int \*p = NULL;

p = &i;

printf(“i의 주소 : %p\n“, &i);

printf(“i의 값 : %d\n“, i);

printf(“p의 값 : %p\n“, p);

printf(“\*p의 값 : %d\n“, \*p);

return 0;

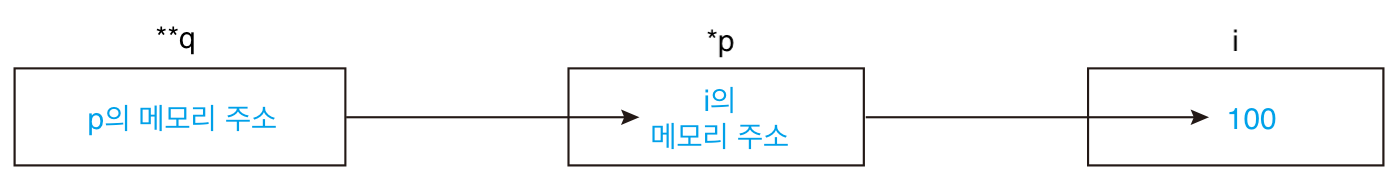
**}**

포인터가 가리키고 있는 값에 접근하기 위해선 연산자 \*를 사용하여 역참조한다. 위의 예시에서 \*p를 출력하게 되면 포인터 p가 가리키고있는 변수 i에 접근되어 변수 i의 값이 출력된다. 포인터 p를 선언할 때 NULL로 초기화 하였는데, NULL로 초기화된 포인터 p를 출력하게 되면 0을 출력한다.

포인터 변수를 가리키는 포인터 변수를 만들 수도 있는데 이를 ‘이중포인터’라고 하고 아래의 코드처럼 표현할 수 있다.

int i = 100, \*p = &i, \*\*q = &p

포인터 변수 또한 주소를 가지고 있기 때문에 포인터의 포인터로 포인터 변수 q를 선언하였다. 위의 구조를 그림으로 그리면 다음과 같다.



\*\*q는 i를 의미하고, \*q는 p를 의미한다.

* 1. **포인터형의 연산**

(1) Pascal

Pascal에서 포인터형 연산은 증분, 감소, +, -로 표현할 수 있다. 포인터의 연산은 일반 정수형 연산과는 차이가 있는데, 주소 1000를 가르키는 정수형 포인터 ptr를 Inc(ptr);로 증분을 해보자. 그럼 ptr은 1001이 되는 것이 아니라 1004가 된다. 정수형의 크기가 4(32비트 기준)이기 때문이다. 이와 같은 연산은 배열에서 자주 사용되는데, Pascal에서 배열은 증감 연산이 불가능하다. 따라서 배열의 주소를 포인터에 넣어서 증감 연산을 할 수 있다.

**var**

arr: array [1..3] of integer = (10, 100, 200);

i: integer;

iptr: ^integer;

**begin**

iptr := @arr[1];

**for** i := 1 **to** 3 **do**

**begin**

writeln('arr[', i, '] = ' , iptr^ );

inc(iptr);

**end;**

**end.**

-------결과 -------

arr[1] = 10

arr[2] = 100

arr[3] = 200

iptr에 배열의 주소가 배정되고 이 iptr를 inc()를 사용하여 arr[1]부터 arr[3]의 모든 배열의 주소를 참조하는 것을 확인 할 수 있다. 만약 증분이 아닌 감소를 한다면 for문의 조건을 3부터 downto 되게 한 다음에 dec()을 사용하여 감소할 수 있다.

(2) C언어

C언어에서의 포인터 연산은 산술 연산과 차이가 있지만 형식은 비슷하다. 다음은 C언어의 포인터 연산 예시 코드이다.

**#include <stdio.h>**

**int main(void)**

**{**

int arr[10], \*p = NULL, \*q = NULL;

p = &arr[3];

q = p + 4;

printf(“q – p = %d\n”, q – p);

printf(“(int) q – (int) p = %d\n”, (int) q – (int) p);

return 0;

**}**

-------결과 -------

q – p = 4

(int) q – (int) p = 16

10칸 배열 arr이 선언되고 포인터 p는 배열의 4번째 원소(c언어에서 배열의 인덱스는 0부터 시작)를, 포인터 q는 배열의 8번째(4+4) 원소를 가리키고있다. 이때, q-p를 출력하면 각 포인터가 가리키고 있는 원소들 사이의 개수인 4가 출력된다. 이 연산은 포인터끼리의 연산이므로 포인터 연산이다. 포인터 연산을 산술 연산으로 바꾸기 위해서 (int)로 형변환을 하여 계산을 하였다. 그 결과, 16이 출력되었는데 이는 int형의 크기가 4이기 때문이다. 다시 말해, int형 배열의 주소는 int형의 크기인 4바이트씩 배정받았는데, arr[0]의 주소가 1000이라하면 arr[1]의 주소는 1004가 된다. 주소 값을 int형으로 형변환을 하여 계산을 하였기때문에 16이 출력된 것이다. 만약 배열과 포인터가 int형이 아닌 double형으로 선언되었다면 q - p의 값은 4 \* 8(double형 크기) = 32가 출력되었을 것이다.

1. **참고문헌**

* 김명호, (2014), “소프트웨어 개발자를 위한 C프로그래밍”, 홍릉과학출판사
* 조정호, 전현민, 김응주, 장문수, & 김영진, (2018), ‘알기쉬운 C프로그래밍”, 光文閣
* Beck, Leland L, (1994), “Pascal : a guided tour”, Addison-Wesley
* Kawai Satoru, (1998), “프로그램 작성법(Pascal)”, 홍릉과학출판사
* Learn Pascal Programming, (n.d.), tutorialspoint, <https://www.tutorialspoint.com/pascal/index.htm>
* Nell B. Dale, (1997), “Programming in Pascal”, Jones & Bartlett Learning