제10장 기초 Perl 프로그램밍

10.1 Perl?

Perl 언어는 C 언어와는 달리 컴파일러가 필요하지 않은 **인터프리트(Interpret) 언어**이다. 이는 90년대 컴퓨터 교육의 광풍이 불었을 때 많이 가르치던 GW-BASIC 과 같이 프로그램을 입력하면 바로 실행이 되는 형태를 띈다. Perl의 또 다른 장점은 운영체제(Operating System)에 독립적이라는 점이다. 즉, 윈도우즈에서 실행한 Perl 프로그램은 특별한 예외(시스템 관련된 코드는 운영체제의 특성에 영향을 받는다)를 제외하고는 바로 UNIX에서 실행이 가능하다(그림 10.1).

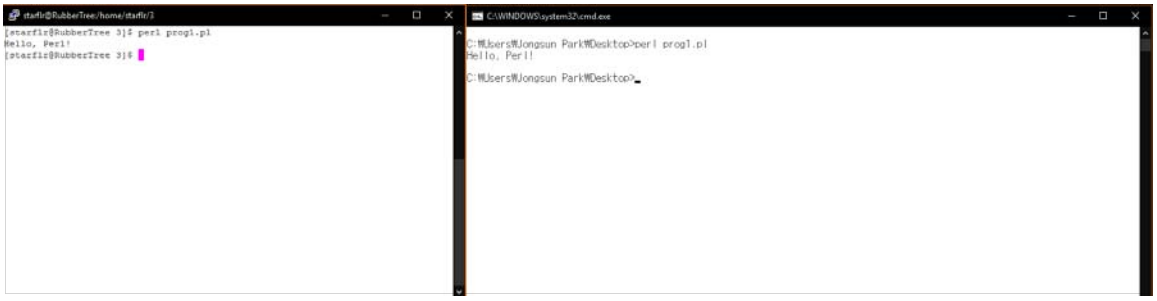


그림 10.1 같은 프로그램 (prog1.pl)을 Linux와 윈도우에서 실행한 화면

Perl 언어의 문법은 C와 비슷하면서도 변수를 선언할 때 변수의 타입 (예, 숫자, 문자열 등)을 설정하지 않고 바로 사용이 가능하기 때문에 초보자로 쉽게 프로그래밍을 할 수 있다. 이에 반해서 C 프로그램의 경우는 변수를 사용하기 전에 해당 형식을 설정해야 하고, 설정이 되고 나면 변경하는 것이 불가능하다. 또한 다른 형태의 변수 간 데이터 이동을 위해서 형변환에 대한 고려를 해야 하는 어려움이 있다. 마지막으로, C 프로그램을 배울 때 가장 많은 사람이 포기하는 부분이 포인터(pointer)인데, 포인터를 이해하지 못하면 C 프로그램을 효율적으로 만들 수 없는, 가장 어려운 관문이 된다(그림 10.2). 이에 반해서 Perl은 포인터 없이도 다양한 문자열 처리가 가능하도록 설계가 되었다. 이와 같은 장점이 많은 사람들로 하여금 Perl을 사용하고, 생물정보학에서도 초기에는 Perl 프로그램을 많이 사용하였다.

그림 10.2 C의 포인터 개념. 포인터는 메모리의 주소이다.

Perl 코드는 사람에 따라서 일반적인 프로그램처럼 짤 수도 있지만, 간결하게 코드를 짤 수 있는 것으로도 유명하다. 원래 프로그램 언어를 개발할 때 간결하게 프로그래밍을 할 수 있도록 설계를 하였고, 같은 프로그램도 다양한 형태로 표현될 수 있도록 고려를 하였다. 가장 극단적인 예로는 Perl로 시를 쓰는 경우도 있는데, 아래 그림이 그 예이다(그림 10.3).

No 5.5: I just want to siiiing! by Petruchio Nov 22 2000 rep:180+

Monks:
He's a Perl Hack, and he's okay,
He hacks all night and he sleeps all day!

Petruchio:
I write my code, I take lunch breaks,
I go to the Monastery!
Sometimes I post my homework,
And merlyn yells at me!

Monks:
He writes his code, he takes lunch breaks,
He goes to the Monastery!
Sometimes he posts his homework,
And merlyn yells at... him.

All:
He's a Perl Hack, and he's okay,
He hacks all night and he sleeps all day!

Petruchio:
I debug code, I call in sick,
I stay home and play Doom!
I write annoying letters
About MonkMail to vroom!

그림 10.3 Perl로 쓰여진 시 예제 (http://www.perlmonks.org/?node_id=1111395)

마지막으로 Perl의 강점은 문자열 처리에 활용하는 **정규 표현식(Regular Expression)**이다. 텍스트 파일을 읽어서 패턴분석을 하는 데에는 정규표현식을 이용해서 매우 간단하게 프로그램을 만들 수 있을 뿐 아니라, 매우 빠르게 그 처리를 진행할 수 있다. 그런데 이 표현식은 익숙해지지 않으면 해독이 안 되는 암호화된 문자열처럼 보인다(그림 10.4). 생물정보학에서도 이 부분 때문에 여러 연구자, 개발자들이 Perl을 애용하고 있다.

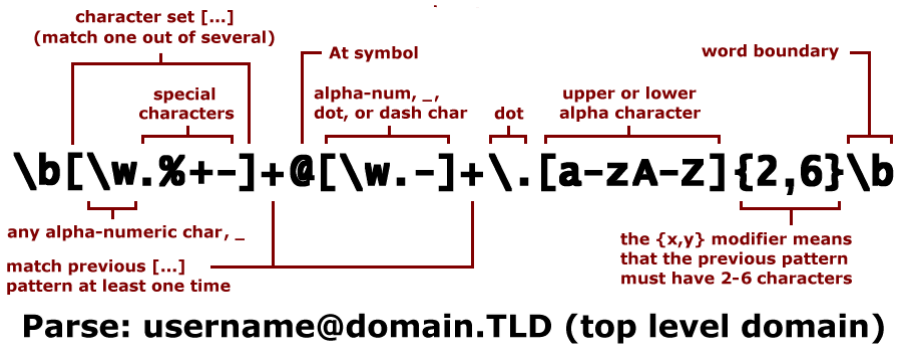


그림 10.4. Perl 정규 표현식의 예제

Box 10.1. Regular expression (정규 표현식)

그림 10.4에서 보이는 것처럼 엄청나게 복잡한 암호처럼 보이는 문자열을 정규표현식(Regular Expression)이라고 한다. Perl의 가장 큰 강점은, 복잡한 문자열 패턴을 단 한줄의 정규표현식으로 정리를 할 수 있다는 점이다. 또한 이 정규표현식의 실행속도는 C로 복잡하게 프로그램을 짠 것과 대등한 속도를 낸다. 이 때문에 텍스트 자료 처리 등에 Perl 의 정규표현식이 자주 사용되곤 하였다.

생물정보학(Bioinformatics)의 경우는 전형적으로 text 파일을 많이 다루는 분야이다. 계속적으로 소개가 되지만, 대부분의 결과물은 텍스트 형태로 저장이 되며(일부 용량 때문에 압축하는 경우 제외), 이 결과에서 원하는 정보를 추출할 일을 자주 수행하게 된다. 왜냐하면 대부분의 생물정보학 분석은 한 번에 끝나는 것이 아니고 수 십 단계를 거쳐서 진행이 되기 때문에, 결과로부터 원하는 정보를 빠르게 추출해서 다음 단계에서 분석하는 작업을 거치게 된다.

10.2 Perl 프로그램 시작하기

Perl 프로그램을 만들기 위해서는 Perl 인터프리터(Interpreter)가 있어야 한다. 대부분의 UNIX/Linux 서버에는 Perl 인터프리터가 기본적으로 설치되어 있다. 윈도우에서 Perl 프로그램을 수행하기 위해서는 ActivePerl (http://www.activeperl.com/)에서 Perl 인터프리터를 다운받아서 설치하면 된다(그림 10.5).

Perl 인터프리터가 있는지 확인하기 위해서 UNIX/Linux에서는 perl -v를 입력해서 Perl 버전이 나오면 설치가 되어 있는 것이다. 윈도우즈 역시 명령 프롬프트(cmd)에서 같은 명령을 입력하면 확인 가능하다(그림 10.6, 10.7).

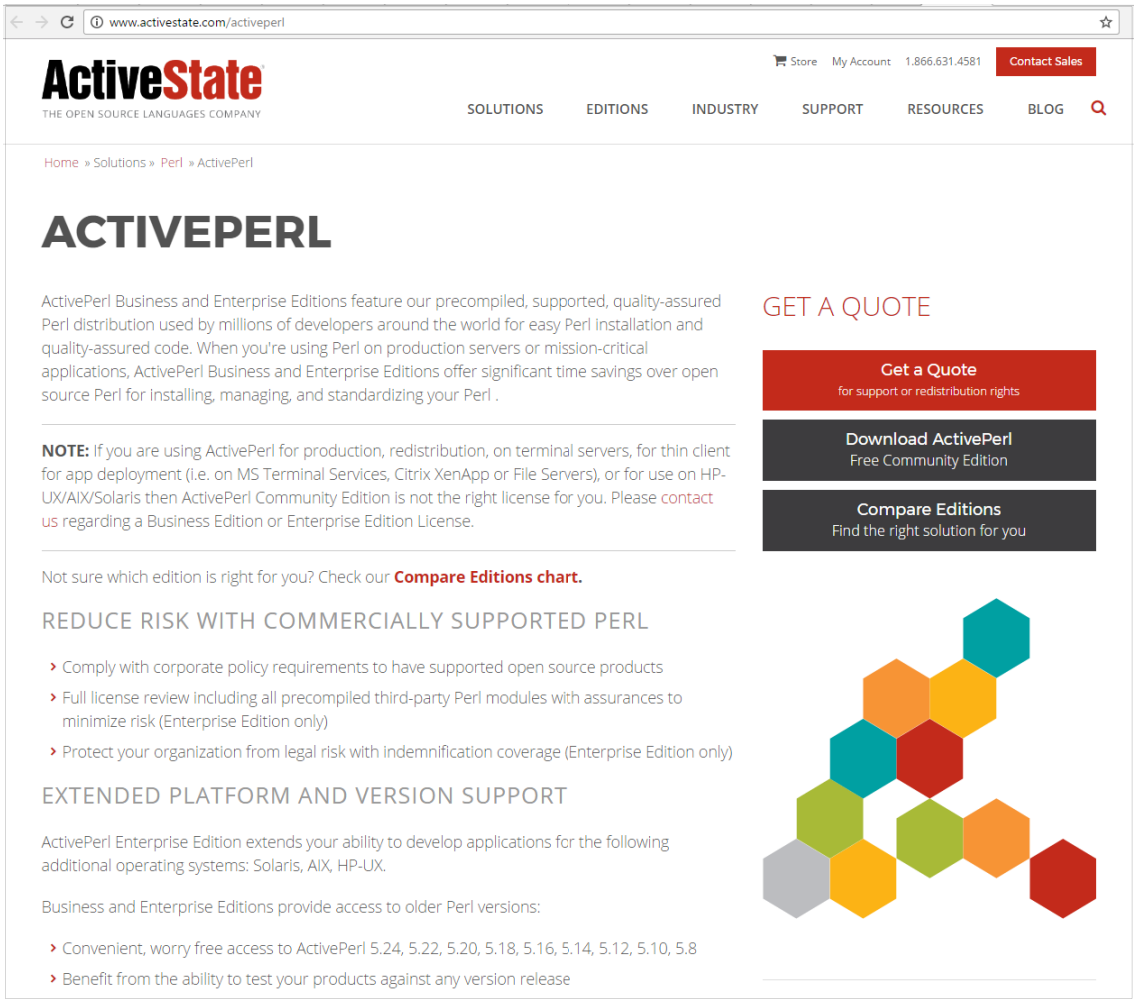


그림 10.5 http://www.activeperl.com 웹 사이트. 무료로 Perl interpreter를 다운로드 받을 수 있다.

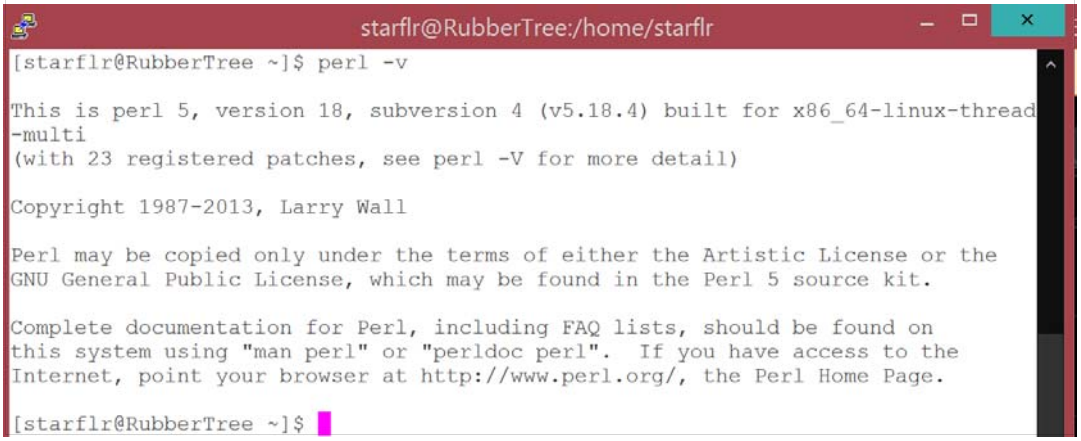


그림 10.6. UNIX/Linux 서버에서 Perl 인터프리터를 확인한 화면.

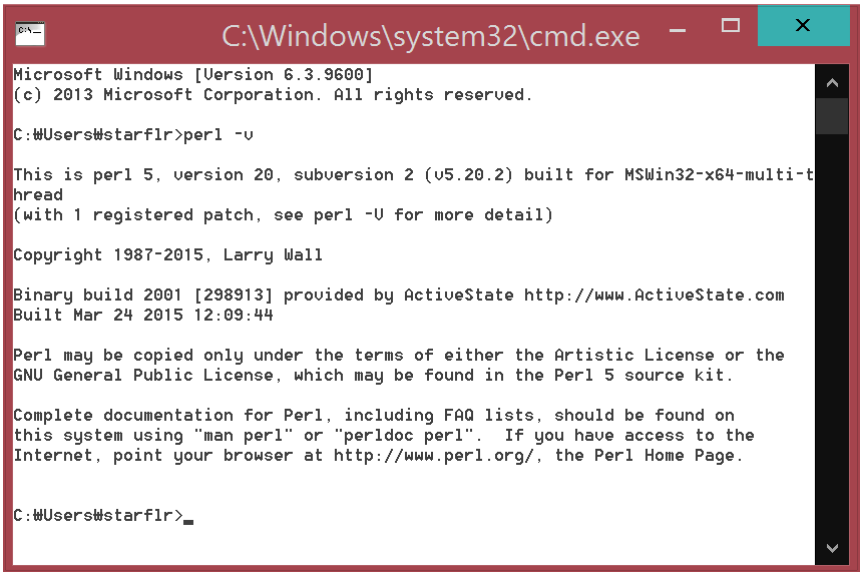


그림 10.7. 윈도우즈에서 Perl 인터프리터를 실행한 화면.

10.3 Hello, Perl!

어떤 언어든 제일 먼저 프로그래밍 연습하는 것이 바로 이 Hello 시리즈이다. 지금 Perl에 대해서 공부를 하고 있으므로, Hello, Perl!을 출력하는 프로그램을 작성해보도록 한다.

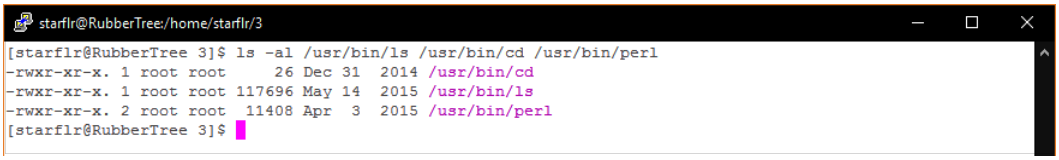
6장에서 소개한 vi 에티터를 사용해서 아래 화면과 같이 프로그램을 입력한다(그림 10.8).

그림 10.8. Hello, Perl 프로그램 코드.

Perl 프로그램 작성 시 가장 첫줄은 #!/usr/bin/perl -w을 넣는다. 가장 앞에 #은 주석이라는 의미이나, !과 함께 사용하면 UNIX/Linux에서 해당 프로그램 구동시 /usr/bin/perl 프로그램을 인터프리터로서 쓰라는 것을 의미한다. -w는 경고 (warning)을 출력하는 옵션인데, 초기에 프로그래밍을 할 때 유용하게 사용할 수 있다. 특히, 변수(variable)의 오탈자로 인해서 오동작 하는 경우를 쉽게 찾아낼 수 있다.

Box 10.2. /usr/bin

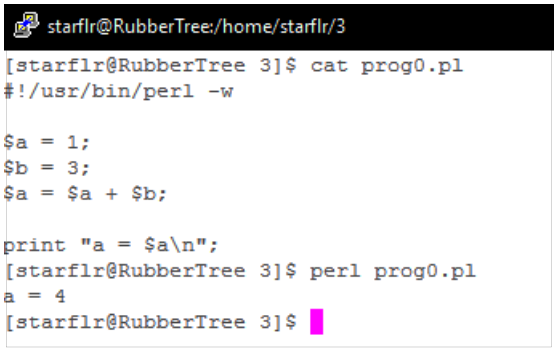
Linux의 대부분 명령어는 모두 프로그램으로 되어 있다. 6장에서 소개한 모든 명령어는 다 프로그램인데, 이들은 전부 /usr/bin 폴더에 위치해 있다. Perl 프로그램을 할 때 상단에 #!/usr/bin/perl -w로 명시하는 것도 perl 프로그램이 /usr/bin에 존재하기 때문이다. ls 명령어로 해당 파일들이 존재하는지 확인해보도록 하자.



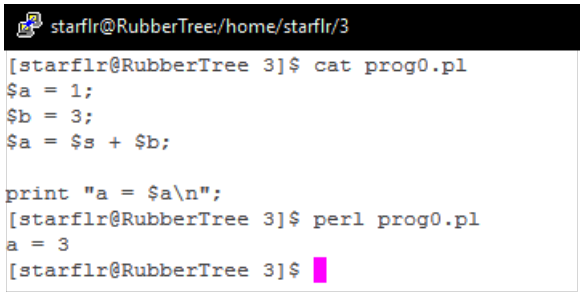
다음 줄에는 use strict;을 명시하였는데, 이는 변수를 정의하지 않고 사용하는 경우는 에러를 내고 프로그램을 중지하도록 하는 기능을 한다. 이 기능이 있는 이유는, Perl에서는 기본적으로 변수를 선언하지 않고 사용하는 것이 가능하다. 프로그래밍 중간에 오탈자로 인해서 변수 이름이 달라졌는데, 프로그램 작성자가 이를 눈치 채지 못하고 넘어가게 되면 프로그램이 원하는 대로 동작하지 않게 된다(Box 10.3 참조). 따라서, 이를 방지하기 위하여 변수를 반드시 선언하고 사용하는 기능을 use strict에서 제공을 해준다.

Box 10.3. Perl에서 오탈자로 인해서 원하는 결과를 얻지 못하는 경우

아직 perl 프로그램은 1개만 보았지만, 선행학습겸 변수 사용에 대해서 간략하게 소개를 한다.



$a와 $b 변수를 사용해서 합산을 하는 프로그램의 예제이다. 프로그램 구동 결과 4가 나오는 것을 확인할 수 있다(1+3=4). 여기서 프로그램을 짤 때 오탈자가 생겨서 $a가 $s가 된다면(아래 참조), 결과가 0+3=3 으로 계산이 된다.



이때, use strict;을 사용하고 $a와 $b를 정의해서(변수 정의는 my를 사용한다.) 가면 다음와 같은 에러가 출력된다.

```
starflr@RubberTree:/home/starflr/3
[starflr@RubberTree 3]$ cat prog0.pl
use strict;

my $a = 1;
my $b = 3;
$a = $s + $b;

print "a = $a\n";
[starflr@RubberTree 3]$ perl prog0.pl
Global symbol "$s" requires explicit package name at prog0.pl line 5.
Execution of prog0.pl aborted due to compilation errors.
[starflr@RubberTree 3]$
```

위에서 보는 것처럼 $s 변수가 선언되지 않았는데, 5번째 줄에서 사용되어서 에러를 출력하고 프로그램을 종료하게 된다. 이렇게 되면 중간에 프로그램 작성자가 문제가 있었음을 확인하고 수정해서 원하는 프로그램을 작성할 수 있게 된다.

마지막으로 화면에 출력하는 줄인 print “Hello, Perl!\n”;을 입력한다. print는 화면에 문자열을 출력하는 함수이며, 가장 마지막의 \n은 개행문자로 줄 바꿈을 의미한다.

만들어진 프로그램(prog1.pl)을 Linux 서버 상에서 실행해보도록 한다. 프로그램 실행방법은 프롬프트에서 perl prog1.pl을 입력하면 된다. 현재 위치에 prog1.pl 이 있어야만 Perl 인터프리터가 이를 읽어서 프로그램을 구동하게 된다(그림 10.9).

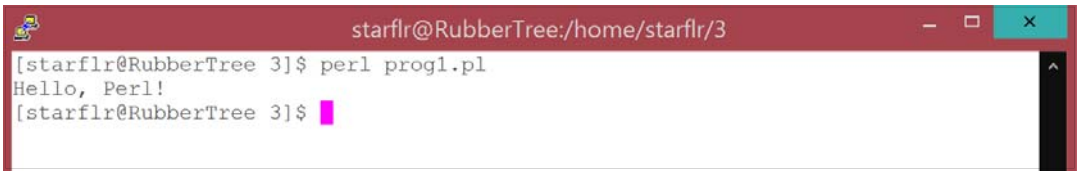


그림 10.9. prog1.pl 프로그램을 구동한 화면. 원하는 대로 Hello, Perl!이 출력되었다.

보는 바와 같이 Hello, Perl 프로그램이 완성되었다.

10.4 숫자 합산 프로그램

다음으로 좀 더 복잡한 프로그램을 작성해보도록 한다. 1부터 100까지의 합을 구하는 프로그램을 작성하는데, 프로그램은 1부터 100까지 숫자를 증가시키면서 그 합을 저장하고, 100에 도달하면 해당 값을 출력하도록 프로그래밍한다.

이 프로그램을 위해서 Perl의 변수에 대해서 소개를 한다. 프로그램에서 변수(variable)란, 변하는 값을 저장하는 공간을 의미한다. 실제로는 메모리 상에서 특정 공간을 할당받고, 여기에 값을 저장하게 된다. Perl에서 변수를 사용하기 위해서는 my 명령어를 통해서 변수를 정의하게 된다(그림 10.10).

그림 10.10. Perl 변수 (Variable)의 예제 프로그램.

위의 프로그램을 보면 변수 a는 $a로 표기되고 my에 의해서 정의되었다. 그리고 초깃값을 1로 가진다. $b는 초깃값으로 2의 값을 가지게 된다. $c는 0의 값을 가진다. 이렇게 정의된 변수는 프로그램 안에서 다양한 값을 가질 수 있는데, 위의 예제에서는 변수 c에 a와 b의 값을 합산한 값($a + $b)를 할당하도록 하였다. 이 과정을 거치면 변수 c에는 1+2=3의 값이 저장되게 된다.

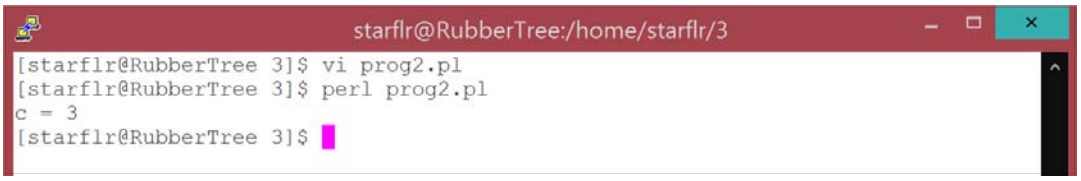


그림 10.11. 예제 프로그램 (prog2.pl) 실행 결과.

프로그램을 실행하면 예상대로 c = 3이 출력된다 (그림 10.11).

다음으로 Perl의 루프(loop)문에 대해서 간략하게 알아보도록 한다. Perl에서는 C와 동일한 여러 가지 루프문을 제공하는데, 여기에서는 for 문에 대해서 알아보도록 한다.

그림 10.12. for loop 예제 프로그램.

위의 예제 프로그램을 보면 for 문 안에서 변수 i가 정의되고 0의 값을 가진 후 (my $i = 0), i값이 10보다 작을 때까지($i<10) 반복을 한다. 한번 반복할 때마다 i의 값은 1씩 증가($i++)한다. for 문에 의한 반복은 이와 같이, 초기 값(0), 종료 조건 ($i<10), 한번 반복할 때마다 실행하는 명령어 ($i++) 로 구성되어 있고, { 와 } 로 감싸져 있는 줄(print “Hello, Perl!\n”;)을 반복하게 된다(그림 10.12). 이 프로그램을 구동하게 되면 i값이 0부터 9까지 (10일 때는 $i<10 조건에 맞지 않아서 루프가 종료된다.) 변하면서 루프안의 명령어를 실행한다.

그림 10.13. for 문 Perl 프로그램 실행 결과.

프로그램 구동 결과 위의 화면처럼 10줄의 Hello, Perl! 이 실행됨을 알 수 있다.

마지막으로 1부터 100까지의 합을 구하는 프로그램을 for문을 활용해서 구현해보도록 한다. 바로 위의 예제에서 10번 반복하는 프로그램 코드를 100번 반복하도록 수정한 후에, 첫 예제에서 사용한 명령($c = $a + $b;)을 활용해서 숫자를 계속 합산하도록 코드를 작성한다. 단, i값이 1부터 100의 값을 가지도록 조정을 해야 한다(그림 10.14).

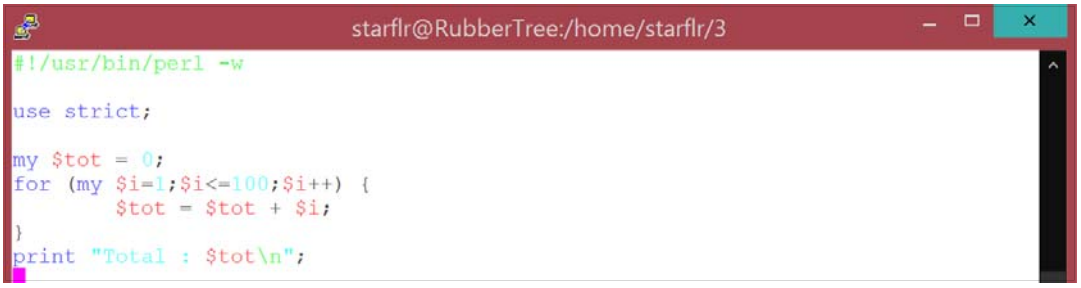


그림 10.14. 1부터 100까지의 합을 구하는 프로그램.

위의 프로그램을 보면 for 문을 활용하고 i 변수가 1부터 100까지의 값을 가지면서 1씩 증가하도록 설정하였다. 그리고 tot변수를 정의하여, i 의 값을 tot에 계속 합산($tot = $tot + $i)하도록 하였다. 이렇게 하면 프로그램이 100번의 반복을 통해서 1부터 100까지의 합을 계산할 수 있게 된다. 실행 결과는 다음과 같다(그림 10.15).

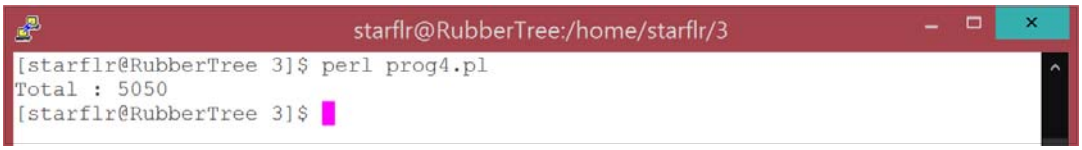


그림 10.15. 1부터 100까지 합을 계산하는 프로그램 실행 결과.

10.5 파일 입출력 프로그램

이제 Perl을 이용해서 파일을 열고 닫는 프로그램을 작성해보도록 한다. 실제로 생물정보학에서 사용하는 많은 데이터들은 텍스트 파일로 되어 있다. 따라서 Perl로 파일을 열고 내용을 읽어내는 것은 매우 유용하게 사용될 수 있다.

Perl에서 파일을 여는 함수는 open이며, 닫는 함수는 close이다. 파일을 열게 되면 파일 핸들(file handle)이 생성이 되는데, 이를 이용해서 파일 내용을 읽거나 쓸 수 있게 된다. 먼저 파일을 열고 내용을 읽는 예제 프로그램을 확인해본다(그림 10.16).

그림 10.16. 파일을 여는 프로그램 예제.

위의 예제에서 보면 open 명령어 다음 괄호를 열고 2개의 인자가 사용되었다. 하나는 대문자로 **DATA** 이고, 다른 하나는 **파일 이름**이다. open 명령어의 첫 번째 인자가 파일 핸들이고, 이는 처음에 소개된 STDIN, STDOUT, STDERR와 같은 종류의 것이 된다. 단 DATA는 프로그래머에 의해서 생성되고 소멸될 수 있다. 실제 prog4.pl 파일이 존재하면 에러 없이 open명령어가 실행될 것이다.

다음에 있는 while 명령은 for문 과 비슷한 루프명령어 인데, while 다음의 조건이 충족하면 { } 안의 명령을 실행하게 된다. 본 프로그램에서의 while 문의 조건은 my $line = <DATA>인데, $line변수를 정의하고 DATA라는 파일 핸들에서 한 줄을 읽어오는 명령에 해당된다. 이때 명령어 전체는 파일에서 한 줄을 정상적으로 읽어오면 참(true)을 반환하고, 그렇지 않고 에러가 발생하거나 파일의 끝(End of File)에 도달하게 되면 거짓(false)을 반환하게 된다. 이에 파일의 끝까지 모두 읽고 난 다음에는 my $line=<DATA>가 자연스럽게 거짓(false)이 되어서 while 루프를 종료하게 된다.

결국 이 while 명령어는 파일을 열고 난 다음 처음부터 끝까지 한 줄씩 데이터를 가져오는 역할을 담당하게 된다.

while 문 안에서는 line 변수의 값을 출력하도록 되어 있으므로, 파일을 연 후, 처음부터 한 줄씩 읽어서 화면에 출력하는 과정을 거치게 된다. 프로그램 실행 결과는 아래와 같다(그림 10.17).

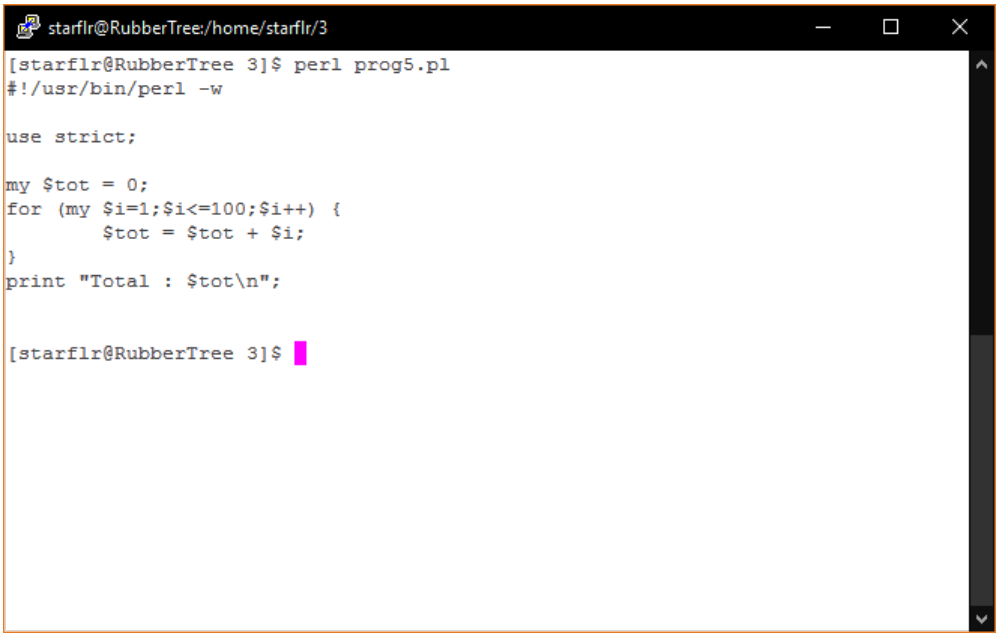


그림 10.17 프로그램 실행 결과

prog4.pl 파일의 모든 내용이 출력되었다. 기존에 배운 cat 명령으로 내용이 같은지 확인해본다(그림 10.18).

그림 10.18. cat 프로그램으로 prog4.pl 파일 내용을 확인한 결과.

내용이 동일함을 확인할 수 있다.

이 프로그램을 기반으로 해서 특정 파일의 줄 수를 세는 프로그램을 작성해볼 수 있다. 위의 프로그램에서, 파일 내용을 출력하는 것 대신 루프가 한 번씩 돌때마다 숫자를 1씩 증가시키는 코드를 넣으면 특정 파일의 줄 수를 확인할 수 있게 된다. 프로그램은 아래와 같다(그림 10.19).

그림 10.19. 파일의 줄 수를 세는 프로그램.

이전 프로그램과 다른 점은 cnt 변수가 정의되고, 루프를 돌때마다 1씩 증가시키는 코드 ($cnt++) 가 들어가 있다. 이는 루프를 몇 번 도는지를 셀 수 있고, 이는 곧 파일이 줄 수와 동일하게 된다. 프로그램 실행 결과, prog4.pl 파일은 총 11줄을 가지고 있음을 확인할 수 있다(그림 10.20).

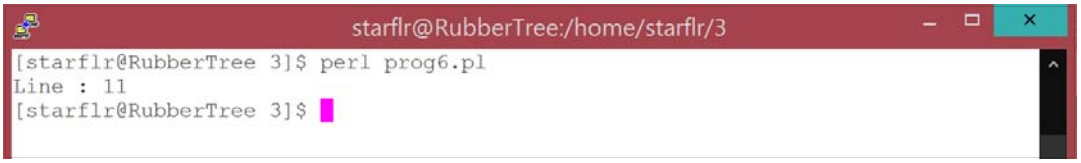


그림 10.20. prog4.pl 파일의 줄수를 세는 프로그램 구동 결과.

10.6 배열

배열(array)에 대한 내용을 살펴보도록 한다. 배열이라 함은 변수의 집합체를 의미한다. 프로그램 작성 시 26개의 변수가 필요한 경우, $a부터 $z까지 26개의 변수를 정의할 수도 있지만, 좀 더 쉽게 @a (배열 a)를 선언하고 이 안에 0번부터 25번 방을 만들어서 데이터를 저장하는 것도 가능하다.

예를 들어서, 1, 2, 3, 5, 8의 수열을 만들어보는 프로그램을 짜보도록 한다. 1 과 2는 초기 값으로, 3은 1+2로 계산이 되고 5는 2+3, 즉 1, 2, 3에서 가장 마지막의 두 개의 수를 합산하고, 8은 3 + 5 로, 앞과 동일한 방법으로 계산하게 되면 원하는 수열을 얻을 수 있다. 이를 직관적으로 프로그램을 작성하게 되면 아래

그림 10.21과 같이 5개의 변수를 사용해서 구현할 수 있다.

starflr@RubberTree:/home/starflr/3

```
[starflr@RubberTree 3]$ cat prog7.pl
#!/usr/bin/perl -w

use strict;

my $a = 1;
my $b = 2;
my $c = $a + $b;
my $d = $b + $c;
my $e = $c + $d;

print "a = $a\n";
print "b = $b\n";
print "c = $c\n";
print "d = $d\n";
print "e = $e\n";
[starflr@RubberTree 3]$
```

그림 10.21 1,2,3,5,8 수열을 생성하는 프로그램

이 프로그램의 실행결과는 아래와 같다(그림 10.22).

starflr@RubberTree:/home/starflr/3

```
[starflr@RubberTree 3]$ perl prog7.pl
a = 1
b = 2
c = 3
d = 5
e = 8
[starflr@RubberTree 3]$
```

그림 10.22 프로그램 실행 결과.

이 프로그램을 배열을 이용해서 프로그램을 작성할 수 있다(그림 10.23).

starflr@RubberTree:/home/starflr/3

```
#!/usr/bin/perl -w

use strict;

my @a;
$a[0] = 1;
$a[1] = 2;

for (my $i=2;$i<5;$i++) {
        $a[$i] = $a[$i-1] + $a[$i-2];
}

for (my $i=0;$i<@a;$i++) {
        print $i." => ".$a[$i]."\n";
}
```

그림 10.23 배열을 이용한 프로그램.

위의 코드에서 배열 변수 @a의 선언은 my로 진행한다. 이후에 배열에 데이터를 할당하기 위해서 $a[0], $a[1]과 같이 위치 정보 (0, 1)을 표시하고 값을 할당하면 된다. 그 다음 코드는 3번째 값은 1번째와 2번째 변수의 합으로 표현되고, 4번째, 5번째의 경우도 각각 자신의 앞과 두 번째 앞에 있는 값의 합으로 정의가 된다. 따라서 for 문을 이용해서 $i의 값을 2부터 4까지 순환하면서 $a[$i] = $a[$i-1] + $a[$i-2]; 명령을 수행하여 원하는 값을 배열 @a의 $i번째 방에 할당을 한다.

마지막으로 계산이 끝나고 출력을 할 때도 for 문을 이용해서 결과를 출력한다. 마지막 for 문의 두 번째 조건에 해당하는 부분이 $i<@a 로 되어 있는데, $i는 변수이고 @a는 배열이다. 그런데 이 코드가 에러가 나지 않고 동작하는 것은 배열 변수가 일반 변수 (Scalar)와 비교나 변수에 할당되게 되는 경우에는 해당 배열이 가지고 있는 요소(element) 수를 반환하는 기능을 자동으로 수행한다. 따라서 $i<@a는 이 프로그램에서 $i<5 와 동일한 효과를 지닌다.

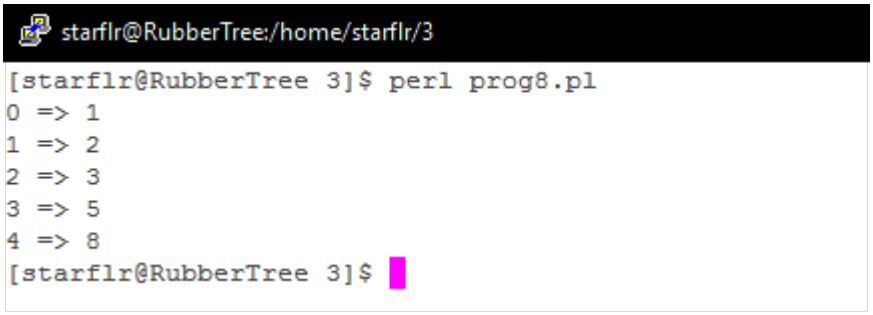


그림 10.24 프로그램 구동 결과

이와 같이 배열을 사용하게 될 경우, 프로그램이 좀 더 체계적으로 구성이 될 수 있다.

10.7 외부에서 인자 전달받기

외부에서 인자를 전달받는 것은, ls 나 grep 명령어가 뒤에 인자를 입력해서 원하는 기능을 수행하게 하는 것과 동일하다. Linux에서는 프로그램 구동 시에 인자를 받을 수가 있는데, perl에서는 @ARGV 라고 하는 특수 변수에 해당 내용이 저장이 되게 된다. 따라서, 원하는 외부 인자를 접근하기 위해서는 @ARGV 배열 변수를 사용하면 된다.

예를 들어서 사용자가 prog9.pl을 구동시킬 때 1000이라는 값을 같이 입력하면, 1000값을 출력하는 프로그램을 작성해보도록 한다(그림 10.25).

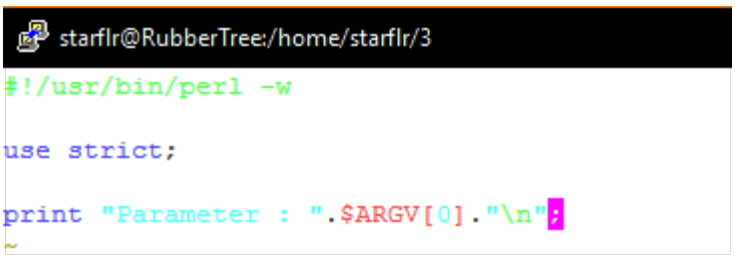


그림 10.25 첫 번째 외부 인자 출력하기

프로그램 구동은 아래와 같이 진행한다. 1000값을 prog9.pl 다음에 같이 입력을 하면, 그림 10.26과 같은 결과를 얻게 된다.

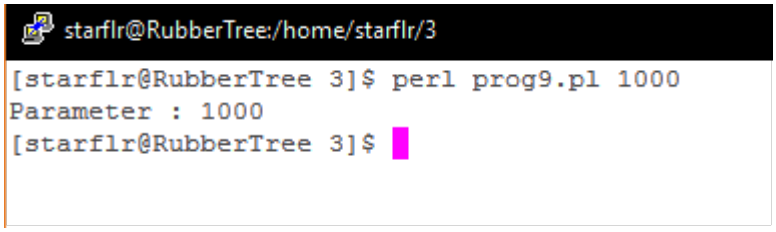


그림 10.26 prog9.pl의 구동 결과

ARGV 변수를 이용해서 1부터 100까지 합을 계산하는 프로그램을 개선해서, 1부터 입력받은 값까지의 합을 계산하는 프로그램을 구성해보도록 한다.

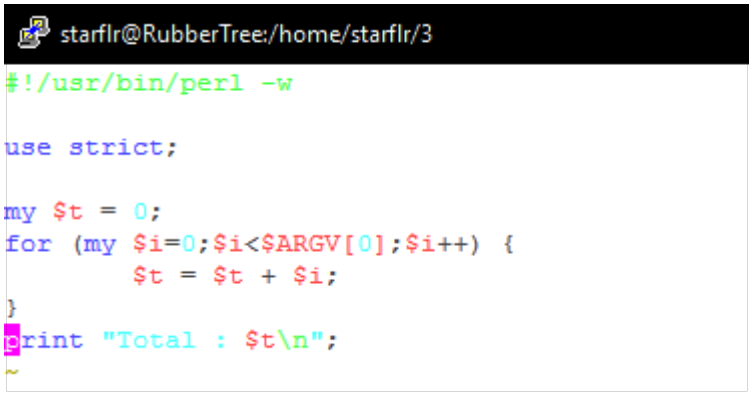


그림 10.27 인자를 받아서, 1부터 인자까지의 합을 구하는 프로그램

외부 인자를 받아서 구동하는 프로그램이기 때문에 다양한 결과를 프로그램을 고치지 않고 얻을 수 있다(그림 10.28).

그림 10.28 프로그램 실행결과

외부 환경인자를 받아들일 수 있게 되면, 보다 효율적인 프로그램 및 상황에 맞는 결과를 얻어낼 수 있다.