# 12장 storage class, linkage, memory management

### Keywords; auto, extern, static, register, const, volatile

### Functions: rand(), time(), malloc(), calloc(), free()

### 변수 scope

### C의 강점 – enable you to control a program’s fine points

#### 어떤 functions이 어떤 variables를 사용하고 얼마나 오래동안 그 variable를 사용하는지를 결정

## 12.1 storage class

### *storage classes*

#### memory에 데이터를 저장한다

##### C는 *object* 용어를 chunk of memory를 표현하기 위해 사용하다

int entity = 3;

##### 정수 변수 선언은 entity라는 *identifier를 만드는 것이다*

##### identifier는 name이다

##### identifier는 메모리에 저장된 object을 designate하는 것이다

#### identifier entity는 memory에 저장된 object을 designate하는 것이다

### int \* pt = &entity;

### int ranks[10];

##### pt는 identifier이다

##### pt는 주소를 갖고 있는 object을 designate하는 것이다.

#### expression \*pt는 identifier가 아니다

##### name이 아니기 때문이다

##### \*pt는 lvalue라는 object을 designate한다

##### entity는 lvalue인 identifier이다

##### \*pt는 lvalue인 expression이다

##### \*(ranks + 2 \* entity)는 lvalue이다

### modifiable lvalue

#### object의 값을 변경할 수 있는 lvalue

const char \*pc = “Behold”;

##### pc는 not modifiable lvalue

##### string literal도 modifiable lvalue가 아니다

### storage duration

#### object이 얼마나 오래동안 메모리에 stay하는가?

##### object을 접근하는 identifier를 scope와 linkage를 사용하여 기술한다

##### scope, linkage, storage duration의 조합으로 각 변수를 기술

## 12.1.1 Scope

### scope: identifier를 접근할 수 있는 program의 한 region

### block scope, function scope, function prototype scope, file scope

#### block scope

#### double blocky(double cleo) {

#### double patrick = 0.0;

#### ...

#### return patrick; }

###### for (int i = 0; i < 10; i++)

###### printf("A C99 feature: i = %d", i);

### function scope

#### goto label

### function prototype scope

### int mighty(int mouse, double large);

### file scope

#### function 바깥에서 정의된 변수

###### #include <stdio.h>

###### int units = 0;

###### /\* a variable with file scope \*/

###### void critic(void);

###### int main(void) { ... }

## 12.1.2 Linkage

### external linkage

#### file scope 변수

#### multifile program – multi translation units

### internal linkage

#### file scope 변수

#### single translation unit – 하나의 source codes + header files

### no linkage

#### block scope, function scope, function prototype scope인 변수들

#### private to block, function, ptototype

###### int giants = 5; // file scope, external linkage

###### static int dodgers = 3; // file scope, internal linkage

###### int main() { ... } ...

##### giants는 같은 program의 일부인 다른 files에서 사용

##### dodgers는 주어진 file에 private, file에 있는 다른 function에서 사용될 수 있다

## 12.1.3 Storage Duration

### scope와 lonkage는 identifier의 visibility을 기술

### storage duration은 identifier에 의해 접근된 object의 persistence를 기술

### 4가지 종류의 storage duration

#### static storage duration

##### program 실행동안 내내 존재

##### file scope 변수들이 여기에 해당

##### static 선언은 storage duration과 관계 없고 linkage type에 관한 것임-주의 필요

##### static 선언 변수: internal linkage

##### internal linkage 또는 external linkage로 선언된 변수들은 static storage duration

#### thread storage duration 안해도 됨

##### thread가 종료할 때까지 존재

##### concurrent programming

#### automatic storage duration

##### block내의 local variable이 여기에 해당

###### void more(int number)

###### {

###### int index;

###### static int ct = 0;

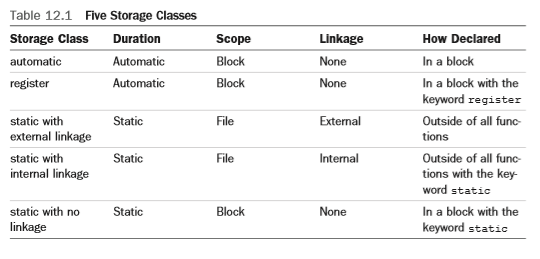
###### ...

###### return 0;

###### }

##### ct는 static memory에 저장, program이 종료할 때까지 존재

#### allocated storage duration



## 12.1.4 static variable with block scope

### static variable – variable stays put in memory(값은 상관없음)

#### file scope 변수도 static storage duration

#### block scope 변수이나 static duration: static 변수 선언

##### block scope, no linkage, static storage duration

##### storage-class specifier: static

//Listing 12.3

/\* loc\_stat.c -- using a local static variable \*/

#include <stdio.h>

void trystat(void);

int main(void)

{

int count;

for (count = 1; count <= 3; count++)

{

printf("Here comes iteration %d:\n", count);

trystat();

}

getchar();

return 0;

}

void trystat(void)

{

int fade = 1;

static int stay = 1;//static 변수

printf("fade = %d and stay = %d\n", fade++, stay++);

}

##### debugger를 사용하여 static int stay = 1;이 실행시에 skip되는 것을 확인

## 12.1.5 static variable with external linkage

### file scope, external linkage, static storage duration

#### external storage class라고 부름

#### external variable이라 부름: function 바깥에 변수 선언

### 다른 file에서 선언된 global variable을 사용하려면 extern 사용

//Listing 12.4

/\* global.c -- uses an external variable \*/

#include <stdio.h>

int units = 0; /\* an external variable \*/

void critic(void);

int main(void)

{

extern int units; /\* an optional redeclaration \*/

printf("How many pounds to a firkin of butter?\n");

scanf("%d", &units);//external 변수 사용

while (units != 56)

critic();

printf("You must have looked it up!\n");

getchar();

return 0;

}

void critic(void)

{

/\* optional redeclaration omitted \*/

printf("No luck, my friend. Try again.\n");

scanf("%d", &units);//external 변수사용

}

## 12.1.8 static variable with internal linkage

### static storage duration, file scope, internal linkage

#### function 바깥에서 static 변수 선언

##### external static 변수라고 부름

##### static variable with internal linkage라고 부름

//Listing 12.5 multiple file의 compile 사용 실습

// parta.c --- various storage classes

// compile with partb.c

#include <stdio.h>

void report\_count();

void accumulate(int k);

int count = 0; // file scope, external linkage, variable definition이라 한다

int main(void)

{

int value; // automatic variable

register int i; // register variable

printf("Enter a positive integer (0 to quit): ");

while (scanf("%d", &value) == 1 && value > 0)

{

++count; // use file scope variable

for (i = value; i >= 0; i--)

accumulate(i);

printf("Enter a positive integer (0 to quit): ");

}

report\_count();

getchar();

return 0;

}

void report\_count()

{

printf("Loop executed %d times\n", count);

}

//Listing 12.6

// partb.c -- rest of the program

// compile with parta.c

#include <stdio.h>

//file scope, external linkage, static storage duration

extern int count; // reference declaration, external linkage, static storage duration, defining declaration이라 한다

// file scope variable

static int total = 0; // static definition, internal linkage, static storage duration

void accumulate(int k); // prototype

void accumulate(int k) // k has block scope, no linkage

{

static int subtotal = 0; // static storage duration, no linkage, block scope

if (k <= 0)

{

printf("loop cycle: %d\n", count);

printf("subtotal: %d; total: %d\n", subtotal, total);

subtotal = 0;

}

else

{

subtotal += k;

total += k;

}

}

## 12.1.9 storage classes, functions

### external function: 다른 file에 있는 function에 의해 접근

### static function: file내에서 사용

static double beta(int, int);

## 12.2 Random-number function and static variable

### static 변수로서 internal linkage인 경우- random-number function

//Listing 12.7

/\* rand0.c –– produces random numbers \*/

/\* uses ANSI C portable algorithm \*/

#include <stdio.h>

static unsigned long int next = 1; /\* the seed \*/

//static with internal linkage

unsigned int rand0(void)

{

/\* magic formula to generate pseudorandom number \*/

printf("rand0():: next = %ld\n", next);

next = next \* 1103515245 + 12345;

return (unsigned int)(next / 65536) % 32768;

}

//Listing 12.8: multiple files의 compile 실습

/\* r\_drive0.c -- test the rand0() function \*/

/\* compile with rand0.c \*/

#include <stdio.h>

extern unsigned int rand0(void);

int main(void)

{

int count;

for (count = 0; count < 5; count++)

printf("%d\n", rand0());

getchar();

return 0;

}

//Listing 12.9

#include <stdio.h>

/\* s\_and\_r.c -- file for rand1() and srand1() \*/

/\* uses ANSI C portable algorithm \*/

static unsigned long int next = 1; /\* the seed \*/

int rand1(void)

{

/\* magic formula to generate pseudorandom number \*/

printf("rand1()::next = %ld\n", next);

next = next \* 1103515245 + 12345;

return (unsigned int)(next / 65536) % 32768;

}

void srand1(unsigned int seed)

{

next = seed;

}

//Listing 12.10

/\* r\_drive1.c -- test rand1() and srand1() \*/

/\* compile with s\_and\_r.c \*/

#include <stdio.h>

#include <stdlib.h>

extern void srand1(unsigned int x);

extern int rand1(void);

int main(void)

{

int count;

unsigned seed;

printf("Please enter your choice for seed.\n");

while (scanf("%u", &seed) == 1)

{

srand1(seed); /\* reset seed \*/

for (count = 0; count < 5; count++)

printf("%d\n", rand1());

printf("Please enter next seed (q to quit):\n");

}

printf("Done\n");

getchar();

return 0;

}

### automated reseeding

#### time()

###### srand1((unsigned int) time(0));

### Roll’Em

#### dice-rolling

//Listing 12.11

#include "diceroll.h"

//diceroll.h

/\*

extern int roll\_count;

int roll\_n\_dice(int dice, int sides);

\*/

//Listing 12.11 diceroll.h

extern int roll\_count;

int roll\_n\_dice(int dice, int sides);

//Listing 12.12

/\* diceroll.c -- dice role simulation \*/

/\* compile with mandydice.c \*/

#include <stdio.h>

#include <stdlib.h> /\* for library rand() \*/

int roll\_count = 0; /\* external linkage \*/

static int rollem(int sides) /\* private to this file \*/

{

int roll;

roll = rand() % sides + 1;

++roll\_count; /\* count function calls \*/

return roll;

}

int roll\_n\_dice(int dice, int sides)

{

int d;

int total = 0;

if (sides < 2)

{

printf("Need at least 2 sides.\n");

return -2;

}

if (dice < 1)

{

printf("Need at least 1 die.\n");

return -1;

}

for (d = 0; d < dice; d++)

total += rollem(sides);

return total;

}

//Listing 12.13

/\* manydice.c -- multiple dice rolls \*/

/\* compile with diceroll.c \*/

#include <stdio.h>

#include <stdlib.h> /\* for library srand() \*/

#include <time.h> /\* for time() \*/

#include "diceroll.h" /\* for roll\_n\_dice() \*/

/\* and for roll\_count \*/

int main(void)

{

int dice, roll;

int sides;

int status;

srand((unsigned int)time(0)); /\* randomize seed \*/

printf("Enter the number of sides per die, 0 to stop.\n");

while (scanf("%d", &sides) == 1 && sides > 0)

{

printf("How many dice?\n");

if ((status = scanf("%d", &dice)) != 1)

{

if (status == EOF)

break; /\* exit loop \*/

else

{

printf("You should have entered an integer.");

printf(" Let's begin again.\n");

while (getchar() != '\n')

continue; /\* dispose of bad input \*/

printf("How many sides? Enter 0 to stop.\n");

continue; /\* new loop cycle \*/

}

}

roll = roll\_n\_dice(dice, sides);

printf("You have rolled a %d using %d %d-sided dice.\n",

roll, dice, sides);

printf("How many sides? Enter 0 to stop.\n");

}

printf("The rollem() function was called %d times.\n",

roll\_count); /\* use extern variable \*/

printf("GOOD FORTUNE TO YOU!\n");

return 0;

}

## 12.3 Allocated memory

### malloc(): return the address of the first byte of that block

#### assign that address to a pointer variable

#### pointer-to-char

#### generic pointer: pointer-to-void

###### double \*ptd;

###### ptd = (double\*) malloc(30 \* sizeof (double));

##### ptd[0]. ptd[1] 사용

## 12.3.1 Free()

//Listing 12.14

/\* dyn\_arr.c -- dynamically allocated array \*/

#include <stdio.h>

#include <stdlib.h> /\* for malloc(), free() \*/

int main(void)

{

double\* ptd;

int max;

int number;

int i = 0;

puts("What is the maximum number of type double entries?");

if (scanf("%d", &max) != 1)

{

puts("Number not correctly entered -- bye.");

exit(EXIT\_FAILURE);

}

ptd = (double\*)malloc(max \* sizeof(double));//typecast

if (ptd == NULL)

{

puts("Memory allocation failed. Goodbye.");

exit(EXIT\_FAILURE);

}

/\* ptd now points to an array of max elements \*/

puts("Enter the values (q to quit):");

while (i < max && scanf("%lf", &ptd[i]) == 1)

++i;

printf("Here are your %d entries:\n", number = i);

for (i = 0; i < number; i++)

{

printf("%7.2f ", ptd[i]);

if (i % 7 == 6)

putchar('\n');

}

if (i % 7 != 0)

putchar('\n');

puts("Done.");

free(ptd);

system("pause");

return 0;

}

## 12.3.2 calloc()

### memory cell을 할당

###### long \*newmem;

###### newmem = (long \*) calloc(100, sizeof (long));

## 12.3.3 dynamic memory allocation

###### int \*pi;

###### pi = (int \*) malloc(n \* sizeof(int));

###### pi[2] = -5;

###### int (\*p2)[6];

###### p2 = (int (\*)[6]) malloc(n\* 6 \* sizeof(int));

###### p2[1][2] = 12;

//Listing 12.15

// where.c -- where's the memory?

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int static\_store = 30;

const char\* pcg = "String Literal";

int main()

{

int auto\_store = 40;

char auto\_string[] = "Auto char Array";

int\* pi;

char\* pcl;

pi = (int\*)malloc(sizeof(int));

\*pi = 35;

pcl = (char\*)malloc(strlen("Dynamic String") + 1);

strcpy(pcl, "Dynamic String");

printf("static\_store: %d at %p\n", static\_store, &static\_store);

printf(" auto\_store: %d at %p\n", auto\_store, &auto\_store);

printf(" \*pi: %d at %p\n", \*pi, pi);

printf(" %s at %p\n", pcg, pcg);

printf(" %s at %p\n", auto\_string, auto\_string);

printf(" %s at %p\n", pcl, pcl);

printf(" %s at %p\n", "Quoted String", "Quoted String");

free(pi);

free(pcl);

getchar();

return 0;

}

## 12.4 ANSI C Type Qualifiers

### C90: constancy와 volatility

### C99: restrict

## 12.4.1 const type qualifier

###### const int nochange;

###### nochange = 12;

###### const int nochange = 12;

###### const float \*pf;

###### float const \* pfc;

###### float \* const pt;

###### const float \* const ptr;

## 12.4.2 volatile

### 변수가 program이 아닌 다른 agencies에 의해 변경될 수 있다

## 12.4.3 restrict

### optimize하도록 허용

int ar[10];

int \* restrict restar = (int \*) malloc(10 \* sizeof(int));

int \* par = ar;

for (n = 0; n < 10; n++)

{

par[n] += 5;

restar[n] += 5;

ar[n] \*= 2;

par[n] += 3;

restar[n] += 3;

}