2장 자료형과 변수

1. 변수
2. 기본 자료형
3. 문자열
4. 사용자 정의 자료형
5. 자료형의 이름을 다시 정의하기
6. 이름 공간

학습목표

- 기본 자료형의 종류에 대하여 설명할 수 있다.
- 문자열을 사용할 수 있다.
- 사용자 자료형을 만들 수 있다.
- 자료형의 이름을 다시 정의할 수 있다.
- 이름공간을 이해하고, 사용할 수 있다.

변수

- 상수(constant)
  - 한 번 정해지면 그 값을 변경할 수 없다.

- 변수(variable)
  - 값을 저장할 수 있는 그릇
  - 메모리에 할당되어 있는 기억 공간을 가리키는 식별자
  - 메모리의 주소가 존재한다.

  - 변수는 반드시 먼저 선언한 후에 사용
  - 변수 선언에 의해 변수의 형태가 정해진다.
  - 컴파일러는 변수의 메모리 공간을 할당하고 어떠한 연산을 실행할 수 있는지를 알게 된다.

변수

- 변수명을 정하는 규칙
  - 영문자(A ~ Z, a ~ z), 숫자(0 ~ 9), 밑줄 문자(_ , underscore)만을 사용하여 만들 수 있다. 공백 문자, 특수 문자를 사용할 수 없다.
  - 첫 문자는 숫자를 사용할 수 없다.
  - 밑줄 문자('_')로 시작하는 이름은 컴파일러와 리소스가 사용하기로 예약되어 있다.
  - 영문자는 대문자와 소문자를 구분한다. 즉, ABC와 abc는 다르게 인식한다.
  - C++에서 사용하는 예약어는 사용할 수 없다.
  - 그 역할을 쉽게 알 수 있는 이름을 사용하는 것이 좋다.

변수

- C++ 예약어 : 식별자로 사용할 수 없다

구분	예약어
표준 C 키워드	auto, break, case, char, const, continue, default, do, double, else, enum, extern, float, for, goto, if, int, long, register, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile, while
표준 C++에 추가된 키워드	alignas, alignof, asm, bool, catch, char16_t, char32_t, class, const_cast, constexpr, decltype, delete, dynamic_cast, explicit, export, false, friend, inline, mutable, namespace, new, noexcept, nullptr, operator, private, protected, public, reinterpret_cast, static_assert, static_cast, template, this, thread_local, throw, true, try, typeid, typename, using, virtual, wchar_t, while
C++ 연산자 표현	and, and_eq, bitand, bitor, compl, not, not_e, or, or_eq, xor, xor_eq

변수

- 변수의 기억 클래스
  - 기억 클래스: 변수가 어디에 저장되는지를 결정
  - 변수의 수명이 정해진다.
  - auto(자동) 변수, register 변수, static(정적) 변수, extern 변수

- auto변수
  - 지역변수는 auto 변수에 속한다.
  - auto 변수는 스택(stack) 영역에 저장되며 해당 블록에서만 유효
  - auto 키워드는 보통 생략된다.
  - auto 변수는 초기화를 해주어야 한다.
- register 변수
  - CPU의 레지스터에 변수를 저장
  - 초기화가 필요하다.

변수

- static 변수
  - 해당 블록 혹은 클래스 내에서만 유효한 변수
  - 전역변수처럼 자유 메모리 영역에 저장되므로 블록을 벗어나더라도 값이 사라지지 않고 그 값을 유지한다.
  - 한 번의 초기화만 가능하고 초기화를 하지 않으면 자동으로 숫자형은 0으로, 문자형은 NULL로 초기화가 된다.
  - 변수 선언 시에 static 지정어를 붙여야 한다.
- extern 변수
  - 프로젝트에 묶여 있는 다른 파일의 변수를 공유하여 사용하기 위하여 외부 변수로 선언
  - 변수 선언시에 extern을 변수 앞에 기술하여야 한다.

기본 자료형

- C++ 기본 자료형

구 분		크 기	설 명
논리형	bool	1 바이트	true와 false의 두 가지 값을 가진다.
문자형	char	1 바이트	(부호 있음) char (부호 없음) unsigned char
정수형	int	2 or 4 바이트	기계나 컴파일러에 따라 다름 (부호 있음) int (부호 없음) unsigned int
	short	2 바이트	(부호 있음) short (부호 없음) unsigned short
	long	4 바이트	(부호 있음) long (부호 없음) unsigned long
	long long	8 바이트	(부호 있음) long long (부호 없음) unsigned long long
실수형	float	4 바이트	단정도 부동소수점 수
	double	8 바이트	배정도 부동소수점 수
	long double	10~16 바이트	Visual C++에서는 double과 동일

기본 자료형

- bool 형
  - true : 참
    - 0이 아니면 참으로 취급
    - 참이면 1을 출력
  - false : 거짓
    - 0이면 거짓으로 취급
    - 거짓이면 0을 출력

- char (문자)형
  - 자료형의 길이 : 1 바이트
  - ' ' 안에 넣어 표현
  - ASCII 코드로 저장
  - 예
    - char ch1 = 'A', ch2 = '0', ch3 = '₩n';

기본 자료형

- ex2_1.cpp

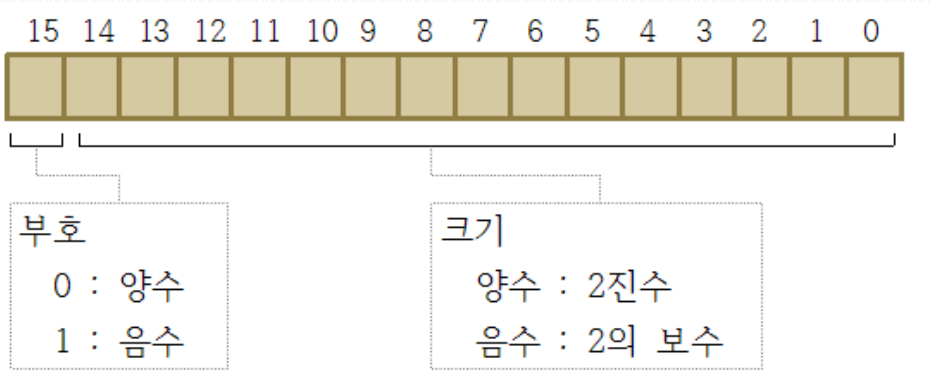
```
#include <iostream>
using namespace std;

void main()
{
	bool  a = true, b = true, c = false;
	cout << "a : " << a << ", b : " << b << ", c : " << c << endl;
	bool  d = 5, e = -3, f = 0;
	cout << "d : " << d << ", e : " << e << ", f : " << f << endl;

	cout << "bool형의 크기: " << sizeof(a) << " 바이트" << endl;
}
```

기본 자료형

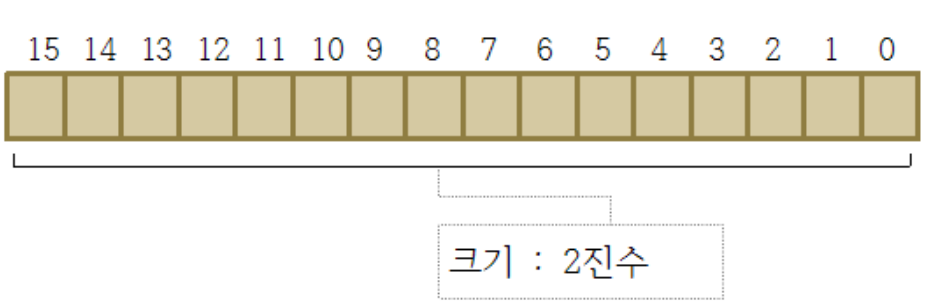
- 정수형 (1)
  - 부호 있는 정수 (예: int)
    - 부호
      - MSB
      - 0이면 양수, 1이면 음수
    - 크기
      - MSB를 제외한 나머지 비트
      - 양수: 2진수로 표기
      - 음수: 2의 보수로 표기
    - (예) : 0을 short int 로 나타내기
      - 00000000 00000000
    - (예) : -1을 short int 로 나타내기
      - 11111111 11111111
  - 부호 없는 정수의 비트 구성 (예: unsigned int)
    - 모든 비트: 크기를 나타냄

기본 자료형

- 정수형 (2)
  - 부호 있는 정수의 비트 구성 (예: short int)



  - 부호 없는 정수의 비트 구성 (예: short int)



기본 자료형

- 논리형, 문자형, 정수형 변수

자료형	크기	범위
bool	1 바이트	true, false
char	1 바이트	$-2^7 \sim 2^7-1$ (–128 ~ 127)
unsigned char	1 바이트	$0 \sim 2^8-1$ (0 ~ 255)
short	2 바이트	$-2^{15} \sim 2^{15}-1$ (–32,768 ~ 32,767)
unsigned short	2 바이트	$0 \sim 2^{16}-1$ (0 ~ 65535)
long	4 바이트	$-2^{31} \sim 2^{31}-1$ (–2,147,483,648 ~ 2,147,483,647)
unsigned long	4 바이트	$0 \sim 2^{32}-1$ (0 ~ 4294967295)
long long	8 바이트	$-2^{63} \sim 2^{63}-1$
unsigned long long	8 바이트	$0 \sim 2^{64}-1$

기본 자료형

- 실수형(부동소수점수)

자료형	크기	정밀도	범위
float	4 바이트	단정도	±3.4E+/-38 (7 digits)
double	8 바이트	배정도	±1.7E+/-308 (15 digits)
long double	10~16 바이트	Visual C++에서는 double와 동일	

  - 부동 소수점 상수는 기본적으로 double 형으로 저장
  - float 형은 접미어로 f, F를 붙인다.
  - long double 형은 접미어로 l, L을 붙인다.

문자열

- 문자열(string)
  - 문자들의 배열로 구성
  - 이중따옴표(" ")로 둘러싸여 표현
  - 문자열 상수의 마지막에는 널(NULL) 문자('₩0')가 자동으로 포함
  - char *s = "Hello!";
  - char s[7] = "Hello!";
  - char s[7] = {'H', 'e', 'l', 'l', 'o', '!', '₩0'};

- 문자열 변수의 기억공간 구조 (char s[7] = " Hello! ";)

s[0]	s[1]	s[2]	s[3]	s[4]	s[5]	s[6]
'H'	'e'	'l'	'l'	'o'	'!'	'\0'

문자열

- ex2_2.cpp

```
#include <iostream>
using namespace std;

void main()
{
        char str1[6] = {'k','o','r','e','a','₩0'};          // 문자열
        char str2[6] = {"korea"};                           // 문자열
        char str3[5] = {'k','o','r','e','a'};               // 문자의 집합
//      char str4[5] = {"korea"};                           // 오류

        cout << "str1 : " << str1 << endl;
        cout << "str2 : " << str2 << endl;
        cout << "str3 : " << str3 << endl;

        cout << "str3 : " ;
        for (int i=0; i<5; i++)
                cout << str3[i];
        cout << endl;
}
```

```
str1 : korea
str2 : korea
str3 : korea徼徼徼徼徼?orea
str3 : korea
계속하려면 아무 키나 누르십시오 . . .
```

문자열

- 문자열 처리 함수
  - C 언어 : #include <string.h>
  - C++ 언어 : #include <cstring>

```
char *strcpy(char *dest, const char *src);
            // 문자열을 src에서 dest로 복사한다.
            // 마지막의 널 문자도 복사된다.
```

```
unsigned int strlen(const char *s);
            // 문자열 s의 길이를 계산한다.
            // 마지막에 있는 널 문자('₩n')는 계산되지 않는다.
```

```
int strcmp(const char *s1, const char *s2);
            // 문자열을 비교한다.
            // 리턴 값 : < 0    s1이 s2보다 작다.
                       = 0    s1과 s2가 같다.
                       > 0    s1이 s2보다 크다
```

```
char *strcat(char *dest, const char *src);
            // 문자열을 연결한다.
            // src 문자열의 내용을 dest의 끝부분에 추가한다.
            // 마지막의 널 문자도 추가된다.
```

문자열

- ex2_3.cpp (1)

```
#include <iostream>
#include <cstring>
using namespace std;

void main()
{
	char flower[20];
	char *ptr;

	strcpy_s(flower, 20, "freesia");		// strcpy(flower, "freesia");
	cout << "1. flower : " << flower << endl;
	ptr = flower;
	cout << "2. ptr : " << ptr << endl << endl;

	strcpy_s(ptr, strlen("rose")+1, "rose");	// strcpy(ptr, "rose");
	cout << "3. ptr : " << ptr << endl << endl;
```

문자열

- ex2_3.cpp (2)

```
        strcat_s(flower, 20, "mary");           // strcat(flower, "mary");
        cout << "4. flower : " << flower << endl;
        ptr = flower;
        strcat_s(ptr, 20, " perfume.");         // strcat(ptr, " perfume.");
        cout << "5. ptr : " << ptr << endl;
        cout << "ptr 문자열의 길이는 " << strlen(ptr) << endl << endl;
}
```

```
1. flower : freesia
2. ptr : freesia

3. ptr : rose

4. flower : rosemary
5. ptr : rosemary perfume.
ptr 문자열의 길이는 17

계속하려면 아무 키나 누르십시오 . . .
```

문자열

- string 클래스의 문자열
  - #include <string>

- ex2_4.cpp

```
#include <iostream>
#include <string>
using namespace std;

void main()
{
	string ss1 = "Good-";
	string ss2 = "afternoon.";
	string ss3;

	cout << "ss1 : " << ss1 << endl;
	cout << "ss2 : " << ss2 << endl;
	ss3 = ss1;
	cout << "ss3 = ss1 : " << ss3 << endl;
	ss3 += ss2;
	cout << "ss3 += ss2 : " << ss3 << endl;
}
```

```
ss1 : Good-
ss2 : afternoon.
ss3 = ss1 : Good-
ss3 += ss2 : Good-afternoon.
계속하려면 아무 키나 누르십시오 . . .
```

사용자 정의 자료형

- 구조체 (structure)
  - 기본 자료형을 여러 개 묶어서 하나의 의미 있는 자료의 단위가 되도록 사용자가 만든 자료형

```
struct 구조체명 {
      자료형  변수명;          // 멤버 데이터
      자료형  변수명;          // 멤버 데이터
          ....
};  // 마지막에 반드시 ';'을 붙여야 한다.
```

```
struct   Student {              // 구조체의 선언
         int      id;
         char     name[10];
         float    score;
};

struct  Student  stu1;          // C 에서 구조체 변수 선언
Student  str2;                  // C++ 에서 구조체 변수 선언
```

사용자 정의 자료형

- ex2_5.cpp

```
#include <iostream>
using namespace std;

struct Student {
        int id;
        char name[10];
        float score;
};

void main()
{
        Student stu1;

        cout << "학번을 입력하시오: ₩t";
        cin >> stu1.id;
        cout << "이름을 입력하시오: ₩t";
        cin >> stu1.name;
        cout << "성적을 입력하시오: ₩t";
        cin >> stu1.score;
        cout << endl;

        cout << "학번: " << stu1.id << endl;
        cout << "이름: " << stu1.name << endl;
        cout << "성적: " << stu1.score << endl;
}
```

```
학번을 입력하시오:      20121234
이름을 입력하시오:      홍길동
성적을 입력하시오:      87.5

학번: 20121234
이름: 홍길동
성적: 87.5
계속하려면 아무 키나 누르십시오 . . .
```

사용자 정의 자료형

- 공용체
  - 동일한 메모리 공간에 여러 멤버들이 선언되어 동일한 기억 공간을 공유

```
union 공용체명  {
        자료형 변수명;                    // 멤버 데이터
        자료형 변수명;                    // 멤버 데이터
            ....
};  // 마지막에 반드시 ';'을 붙여야 한다.
```

```
union   Register {                  // 공용체의 선언
          char     ch;
          short    sh;
          long     lo;
};

Register   reg;                     // 공용체의 변수 선언
```

사용자 정의 자료형

- ex2_6.cpp (1)

```
#include <iostream>
using namespace std;

struct Student {
        int id;
        char name[16];
        float score;
};

union Register {
        char ch;
        short sh;
        long lo;
};
```

사용자 정의 자료형

- ex2_6.cpp (2)

```
void  main()
{
        Student  stu1 = {123456, "홍길동", 91.5};
        Register  reg1;
        reg1.lo = 0x12345678;

        cout << "size of stu1 : " << sizeof(stu1) << endl;
        cout << "size of reg1 : " << sizeof(reg1) << endl < endl;

        printf("reg1.ch = %8x ₩n", reg1.ch);
        printf("reg1.sh = %8x ₩n", reg1.sh);
        printf("reg1.lo = %8x ₩n", reg1.lo);
}
```

```
size of stu1 : 24
size of reg1 : 4

reg1.ch =       78
reg1.sh =     5678
reg1.lo = 12345678
계속하려면 아무 키나 누르십시오 . . .
```

사용자 정의 자료형

- 열거형
  - 변수가 가질 수 있는 값들을 미리 나열해 두어, 나열한 값들 외에는 변수에 대입할 수 없도록 한정해 둔 자료형

```
enum   열거형명 {
          값1, 값2, ... ;              // 멤버 데이터
};                                     // 마지막에 반드시 ';'을 붙여야 한다.
```

```
enum   days { MON, TUE, WED, THU, FRI, SAT, SUN };                    // 요일
enum   days { MON=1, TUE, WED, THU, FRI, SAT, SUN };

enum   colors  { black, brown, red, orange, yellow, green, blue, violet, gray, white };
                          // color code
```

자료형의 이름을 다시 정의하기

- typedef
  - typedef를 사용하여 자료형의 이름을 다시 정의할 수 있다.

```
typedef    old_type    new_type;          // 마지막에 반드시 ';'을 붙여야 한다.
```

  - [예]

```
typedef    int                 INT32;
typedef    short               INT16;
typedef    unsigned int        UINT32;
typedef    unsigned short      UINT16;
typedef    unsigned char       BYTE;

typedef    struct     complex {
               double real;
               double imag;
}  COMPLEX;
```

자료형의 이름을 다시 정의하기

- ex2_7.cpp

```
#include <iostream>
using namespace std;

typedef unsigned char BYTE;
typedef struct complex {
        double real;
        double image;
} COMPLEX;

void main()
{
        COMPLEX comp1 = {10.5, 8.2};
        cout << "comp1의 실수부: " << comp1.real << endl;
        cout << "comp1의 허수부: " << comp1.image << endl;

        BYTE  a = 'A', b = '0' ;
        cout << "a = " << a << ", b = " << b << endl;
}
```

```
comp1의 실수부: 10.5
comp1의 허수부: 8.2
a = A, b = 0
계속하려면 아무 키나 누르십시오 . . .
```

이름 공간

- namespace
  - 프로그램의 규모가 커짐에 따라, 시스템 개발을 할 때 모듈별로 여러 사람이 나누어서 동시에 진행하게 된다.
  - 변수, 함수, 사용자 정의 자료형, 클래스 그리고 typedef 등의 식별자를 구분하는 이름이 중복되고 이름들 사이에 충돌이 일어날 수 있다.
  - 동일한 이름의 변수가 있더라도 서로 다른 이름 공간에 있으면 각각의 namespace 안에서는 유일하게 구분할 수 있게 된다.

```
namespace 네임스페이스명 {
    변수
    함수
    구조체
    클래스
    …
}
```

이름 공간

- namespace 사용 예

```
namespace worker {
    char name[10];
    int overtime;
    void Show( … ) { … };
}
```

```
namespace student {
    char name[10];
    float score;
    void Show( … ) { … };
}
```

- 어떤 namespace 안의 식별자를 사용하려면 영역 지정 연산자 (scope resolution operator) ::를 이용하여 해당 namespace에 정의된 식별자를 사용할 수 있다.
  - (예) namespace worker인 경우의 변수 name을 사용하려면
    - worker::name으로 접근
  - (예) namespace student인 경우의 변수 name을 사용하려면
    - student::name으로 접근

이름 공간

- ex2_8.cpp (1)

```
#include <iostream>
#include <cstring>

namespace worker {
	char name[10];
	int overtime;
	void Show(char na[], int ot) {
		std::cout << "name: " << na << ", overtime: " << ot << std::endl;
	}
}
namespace student {
	char name[10];
	float score;
	void Show(char na[], float sc) {
		std::cout << "name: " << na << ", score: " << sc << std::endl;
	}
}
```

이름 공간

- ex2_8.cpp (2)

```
void main()
{
	strcpy_s(worker::name, 10, "Hong");
	worker::overtime = 15;
	worker::Show(worker::name, worker::overtime);
}
```

이름 공간

- using 문
  - using worker::name;
    - 특정 식별자(name)의 경우만 이름 공간을 생략하고 식별자를 사용할 수 있다.
  - using namespace worker;
    - 이름 공간에 정의된 모든 식별자의 이름 공간을 생략하고 식별자를 사용할 수 있다.

- using namespace std;
  - 표준 C++ 라이브러리는 namespace std를 사용한다.
  - iostream 헤더 파일에 있는 cout 객체, endl 조작자 등을 디폴트로 사용할 수 있다.

이름 공간

- ex2_9.cpp (1)

```
#include <iostream>
#include <cstring>

namespace worker {
	char name[10];
	int overtime;
	void Show(char na[], int ot) {
		std::cout << "name: " << na << ", overtime: " << ot << std::endl;
	}
}
namespace student {
	char name[10];
	float score;
	void Show(char na[], float sc) {
		std::cout << "name: " << na << ", score: " << sc << std::endl;
	}
}
```

이름 공간

- ex2_9.cpp (2)

```
void main()
{
        using student::name;
        using student::score;

        strcpy_s(name, 10, "Kim");
        score = 91.1f;
        student::Show(name, score);
}
```

```
name: Kim, score: 91.1
계속하려면 아무 키나 누르십시오 . . .
```

이름 공간

- ex2_10.cpp (1)

```
#include <iostream>
#include <cstring>
using namespace std;

namespace worker {
	char name[10];
	int overtime;
	void Show(char na[], int ot) {
		cout << "name: " << na << ", overtime: " << ot << endl;
	}
}
namespace student {
	char name[10];
	float score;
	void Show(char na[], float sc) {
		cout << "name: " << na << ", score: " << sc << endl;
	}
}
```

이름 공간

- ex2_10.cpp (2)

```
void main()
{
        using namespace student;

        strcpy_s(name, 10, "Kim");
        score = 91.1f;
        Show(name, score);
}
```

```
name: Kim, score: 91.1
계속하려면 아무 키나 누르십시오 . . .
```

Question & Answer

Any Question?

Please.