초보자도 할 수 있는
C# Programming

Fast Campus
Online

한창민 강사.

초보
탈출 I

Fast campus

9강
일반화 프로그래밍

Fast Campus
Online

한창민 강사.

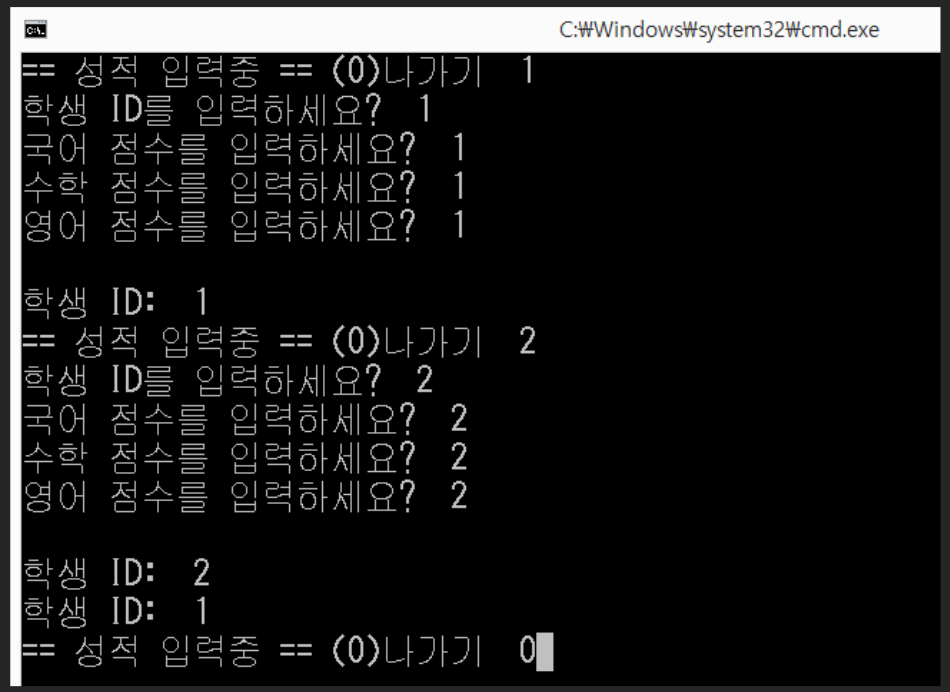
Fast campus

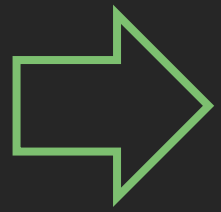
- Check
  - 101_Check(093_Check를 **입력 제한 없이** 처리)
    - **성적 프로그램을 3명까지만 저장하고 정보 검색이 가능한 프로그램**
    - **배열을 사용하여 데이터 저장**
    - **저장했던 자료에서 참고하고 싶은 학생번호로 정보 보여주기**
    - ContainsKey()함수 참고: Hashtable에 저장된 키값이 있는지 체크
    - 결과창 다음 장 참고

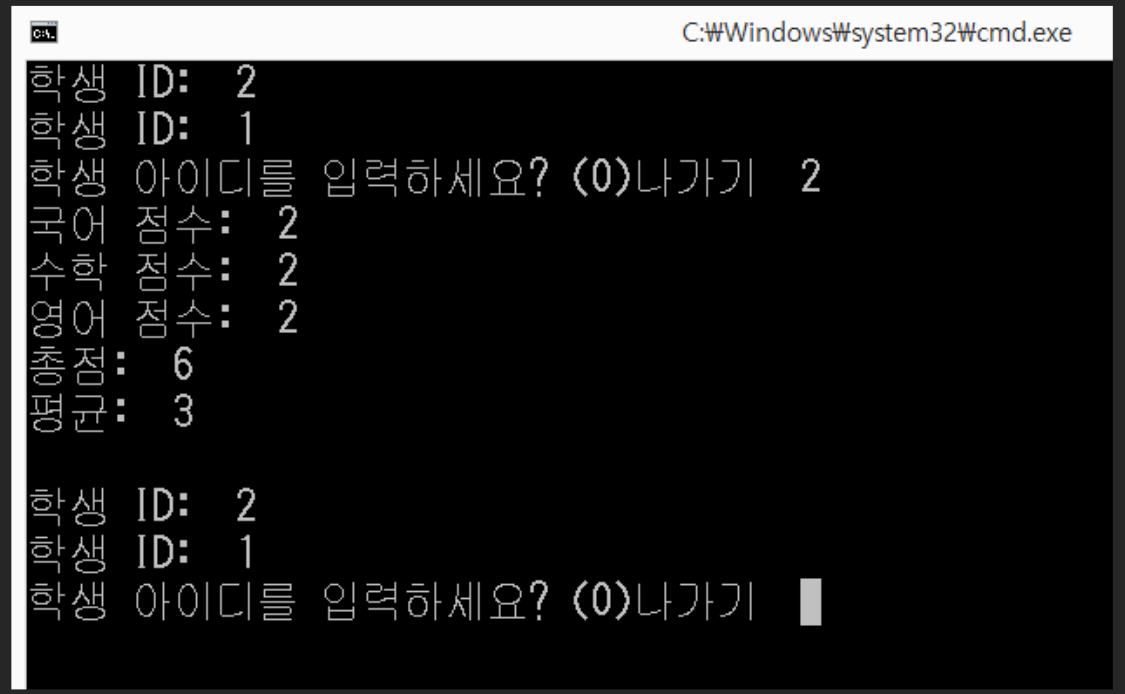
```
static int CheckID(int id, Hashtable hashTable) {
    if(hashTable.ContainsKey(id)) {
        return id;
    }
```

Fast campus

- Check
  - 101_Check(093_Check를 **입력 제한 없이** 처리)







- Check
  - 102_Check(094_Check를 **입력 제한 없이** 처리)
    - **입력 받은 두 수의 결과를 저장하는 프로그램**
    - **제한 없이 계속 입력 받을 수 있음**
    - **저장했던 자료를 모두 보여주기**

Fast campus

- 일반화(Generic)
  - 클래스, 함수 일반화 가능
  - <T> 키워드
  - 장점:
    - Boxing, UnBoxing을 줄일 수 있음
    - 불필요한 오버로딩을 줄일 수 있음

```
static void GenericPrint<T>(T data) {
    Console.WriteLine("data: {0}", data);
}

static void GenericPrint<T>(T[] arrData) {
    for(int i = 0; i < arrData.Length; i++) {
        Console.WriteLine("arrData: {0}", arrData[i]);
    }
}
```


➡ 소스코드 (_103_GenericFunc)

Fast campus

- 일반화(Generic)
  - 클래스 일반화

```
class GenericAA<T>
{
    private T num;
    public T GetNum() {
        return num;
    }

    public void SetNum(T data) {
        num = data;
    }
}
```

```
GenericAA<int> AA = new GenericAA<int>();
```

```
GenericAA<float> BB = new GenericAA<float>();
```


➡ 소스코드 (_104_GenericClass)

Fast campus

- dynamic 키워드
  - object, var
  - 런타임에 변수 형식 변경 가능
  - 일반화 함수에서 변수타입 대응가능

```
static T AddArray<T>(T[] arrDatas) {
    dynamic temp = 0;
```

- default 키워드
  - value type: 0 초기화
  - reference type: null 초기화

```
T temp = default(T);
for(int i = 0; i < arrDatas.Length; i++) {
    temp += (dynamic)arrDatas[i];
}
```


소스코드 (_105_dynamic)

Fast campus

- where 키워드(한정자)
  - 매개변수 제한 기능

```
class AA<T> where T : struct //값형식으로 제한
{
    private T sData;
```

```
class BB<T> where T : class //참조 형식으로 제한
{
    private T sRefData;
```

```
interface II{
    void IIPrint();
}

class CC<T> where T : II //interface로 제한
{
    public T _interface;
}
```

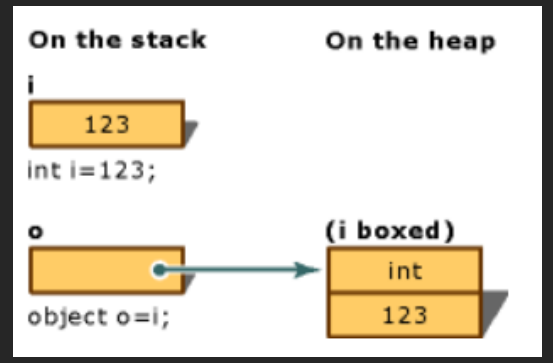
➡ 소스코드 (_106_where)

초보
탈출 I

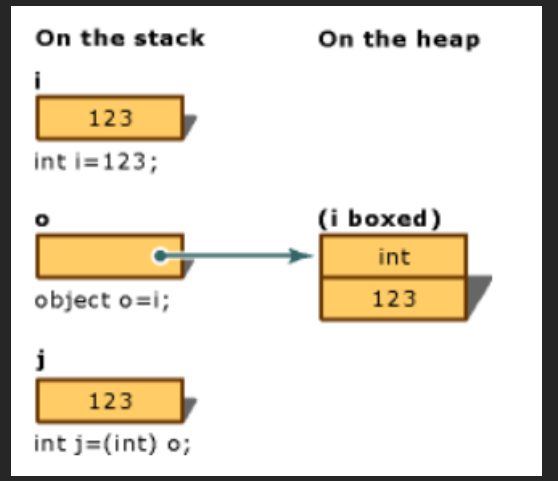
제약 조건	설명
`where T : struct`	형식 인수는 null을 허용하지 않는 값 형식이어야 합니다. Null 허용 값 형식에 대한 자세한 내용은 Null 허용 값 형식을 참조하세요. 모든 값 형식에 액세스할 수 있는 매개 변수가 없는 생성자가 있으므로, `struct` 제약 조건은 `new()` 제약 조건을 나타내고 `new()` 제약 조건과 결합할 수 없습니다. 또한 `struct` 제약 조건을 `unmanaged` 제약 조건과 결합할 수 없습니다.
`where T : class`	형식 인수는 참조 형식이어야 합니다. 이 제약 조건은 모든 클래스, 인터페이스, 대리자 또는 배열 형식에도 적용됩니다.
`where T : notnull`	형식 인수는 nullable이 아닌 형식이어야 합니다. 인수는 C# 8.0 이상의 nullable이 아닌 참조 형식이거나 nullable이 아닌 값 형식일 수 있습니다. 이 제약 조건은 모든 클래스, 인터페이스, 대리자 또는 배열 형식에도 적용됩니다.
`where T : unmanaged`	형식 인수는 nullable이 아닌 비관리형 형식이어야 합니다. `unmanaged` 제약 조건은 `struct` 제약 조건을 나타내며 `struct` 또는 `new()` 제약 조건과 결합할 수 없습니다.
`where T : new()`	형식 인수에 매개 변수가 없는 public 생성자가 있어야 합니다. 다른 제약 조건과 함께 사용할 경우 `new()` 제약 조건을 마지막에 지정해야 합니다. `new()` 제약 조건은 `struct` 또는 `unmanaged` 제약 조건과 결합할 수 없습니다.
`where T : <기본 클래스 이름>`	형식 인수가 지정된 기본 클래스이거나 지정된 기본 클래스에서 파생되어야 합니다.
`where T : <인터페이스 이름>`	형식 인수가 지정된 인터페이스이거나 지정된 인터페이스를 구현해야 합니다. 여러 인터페이스 제약 조건을 지정할 수 있습니다. 제약 인터페이스가 제네릭일 수도 있습니다.
`where T : U`	T에 대해 제공되는 형식 인수는 U에 대해 제공되는 인수이거나 이 인수에서 파생되어야 합니다.

Fast campus

- 일반화 컬렉션(Collections Generic)
  - 컬렉션의 박싱 과 언박싱의 단점을 해결
  - List<T>
  - Queue<T>
  - Stack<T>
  - Dictionary<T>




```
using System.Collections.Generic;
```



➡ 소스코드 (_107_List, _108_Queue)

➡ 소스코드 (_109_Stack, _110_Dictionary)