



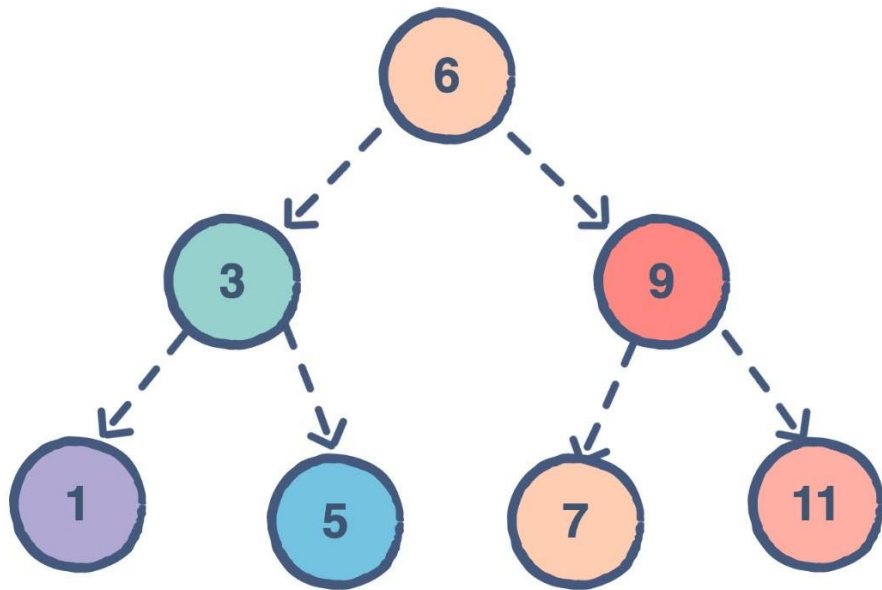
Huffman coding (HW1)

MediaLab.
Dowan Kwon



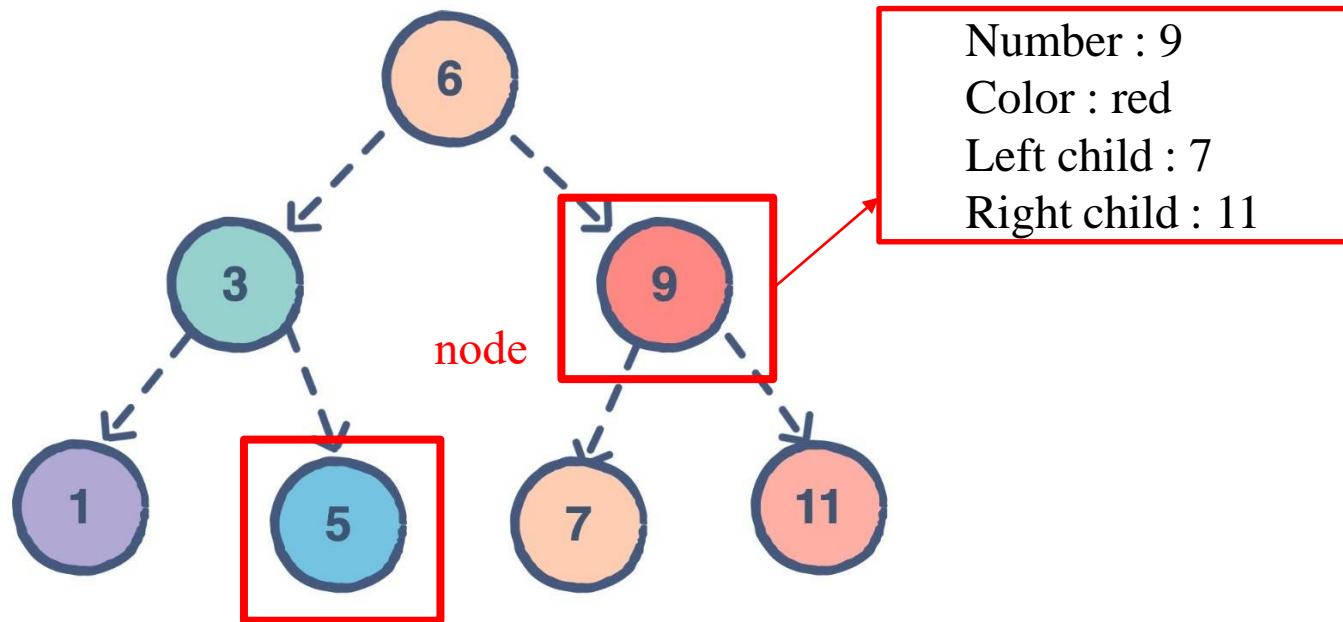
- ❖ Binary Search Tree
- ❖ Huffman coding (code)
- ❖ Assignment
- ❖ Appendix
 - Visual Studio Installation Guidelines

❖ Binary Search Tree



An example of a binary search tree

❖ Binary Search Tree

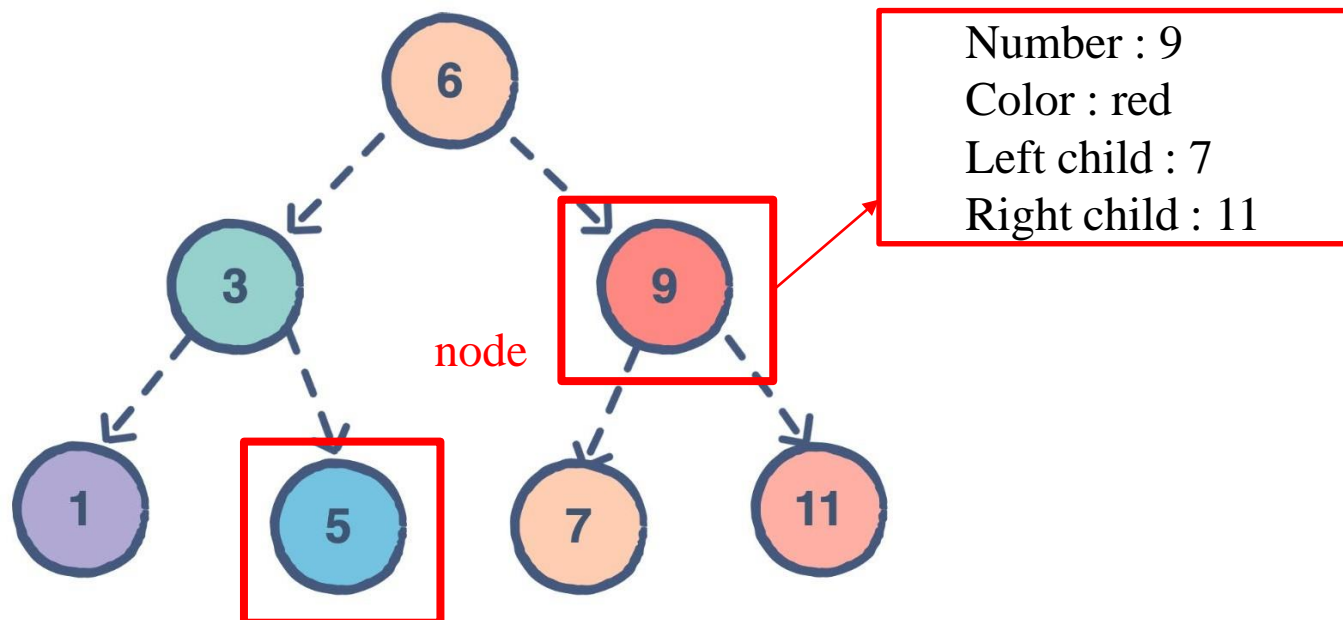


An example of a binary search tree

Number : 5
Color : blue
Left child : NULL
Right child : NULL

Parent node knows the addresses
of its child nodes.

❖ Binary Search Tree



An example of a binary search tree

Number : 5
Color : blue
Left child : NULL
Right child : NULL

Parent node knows the addresses of its child nodes.

Search Methods

DFS (Depth First Search)



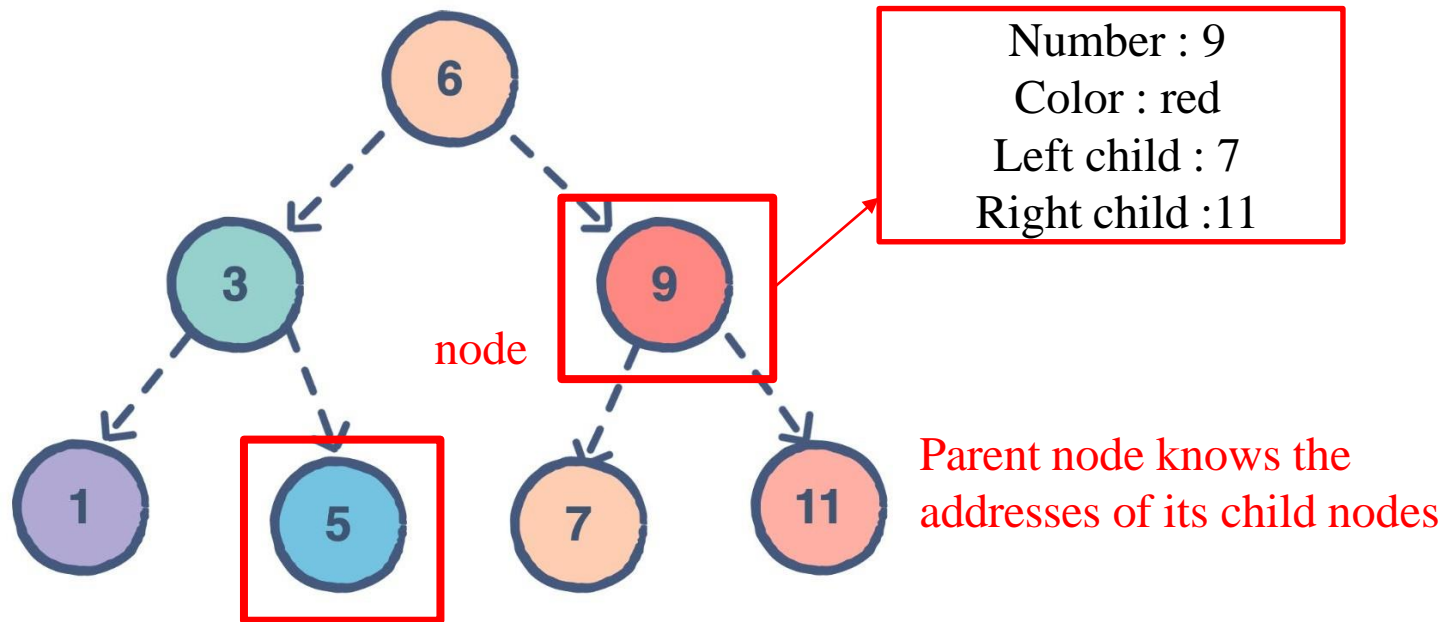
BFS (Breadth First Search)



Question (DFS)

- Is there a 5 inside the tree?
 - If it is there, what color is it?
1. Start from root node (6)
 2. 6 is not 5
 3. Go to left child (3)
 4. 3 is not 5
 5. Go to left child (1)
 6. 1 is not 5
 7. Go to left child (NULL)
 8. Go back to parent (3)
 9. Go to right child (5)
 10. 5 is inside the tree, and it is blue

❖ Binary Search Tree



An example of a binary search tree

Number : 5
Color : blue
Left child : NULL
Right child : NULL

Main.cpp

```
#include "huffman.h"

int main()
{
    getHuffmanCode();

    return 0;
}
```

노드를 나타내는 구조체

```
struct node
{
    string characters;
    unsigned int frequency;
    string code;
    node * leftChild;
    node * rightChild;
};
```

vector<node> nodeArray;

A	B	C	...
7	2	3	

huffman.h

```
void getHuffmanCode()
{
    int size;
    unsigned int tempInt;
    char alphabet;

    //데이터에 사용되는 문자의 종류 개수 입력
    cout << endl;
    cout << "Huffman Tree : ";
    cin >> size;

    cout << endl << endl;
    cout << "문자 빈도" << endl;
    cout << "-----" << endl;

    //각 문자별 빈도 수 노드 생성
    for (int i = 0; i < size; i++)
    {
        cout << "H";
        node tempNode;
        cin >> alphabet;
        cin >> tempInt;

        tempNode.characters = alphabet;
        tempNode.frequency = tempInt;
        tempNode.leftChild = NULL;
        tempNode.rightChild = NULL;
        nodeArray.push_back(tempNode);
    }

    //Huffman Tree 생성
    node root = getHuffmanTree();

    cout << endl << endl;
    cout << "문자" << "Code" << endl;
    cout << "-----" << endl;

    //Huffman Coding Table 생성
    depthFirstSearch(&root, "");
}
```

❖ 다음 데이터 Huffman coding을 통해 압축해보자.

AAAAAAABBCCCCDEEEFFFFFFFG

① 데이터에서 사용되는 각 문자에 대한 출현 빈도수를 구한다.

문자	A	B	C	D	E	F	G
출현빈도	7	2	3	1	4	6	1

노드를 나타내는 구조체

```
struct node
{
    string characters;
    unsigned int frequency;
    string code;
    node * leftChild;
    node * rightChild;
};
```

Characters
Frequency
Code

Root

L

R

```
int size;
unsigned int templnt;
char alphabet;

//데이터에 사용되는 문자의 종류 갯수 입력
cout << endl;
cout << "##" << "Huffman Tree : ";
cin >> size;
```

1

```
//각 문자별 빈도 수 노드 생성
for (int i = 0; i < size; i++)
{
    cout << "##";
    node tempNode;
    cin >> alphabet;
    cin >> templnt;

    tempNode.characters = alphabet;
    tempNode.frequency = templnt;
    tempNode.leftChild = NULL;
    tempNode.rightChild = NULL;
    nodeArray.push_back(tempNode);
}
```

2

vector<node> nodeArray;

A	B	C	D	E	F	G
7	2	3	1	4	6	1



❖ 다음 데이터 Huffman coding을 통해 압축해보자.

AAAAAAABBCCCCDEEEEEEFFFFFFFG

① 데이터에서 사용되는 각 문자에 대한 출현 빈도수를 구한다.

문자	A	B	C	D	E	F	G
출현빈도	7	2	3	1	4	6	1

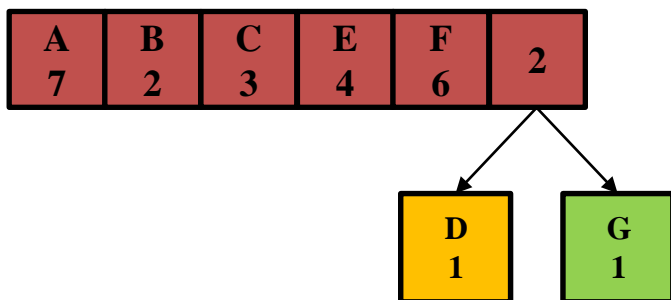
② 빈도가 가장 적은 2개의 문자의 합을 Root에 적는다.

③ 새로운 빈도수의 합을 기준으로 다시 ②과정 반복

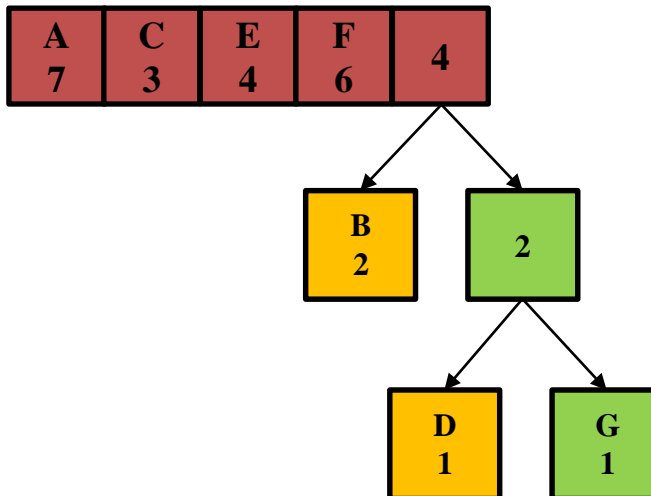
Step 1

A	B	C	D	E	F	G
7	2	3	1	4	6	1

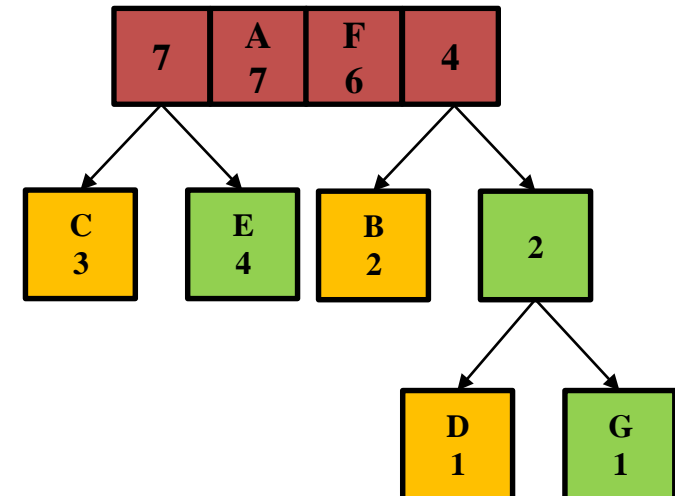
Step 2



Step 3



Step 4





❖ 다음 데이터 Huffman coding을 통해 압축해보자.

AAAAAAABBCCCCDEEEEEEFFFFFG

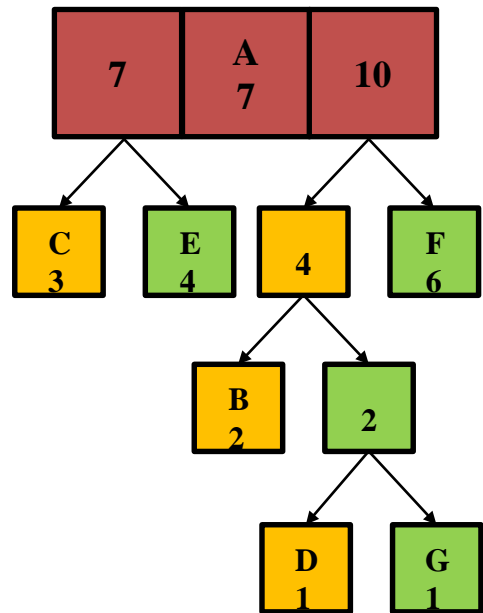
① 데이터에서 사용되는 각 문자에 대한 출현 빈도수를 구한다.

문자	A	B	C	D	E	F	G
출현빈도	7	2	3	1	4	6	1

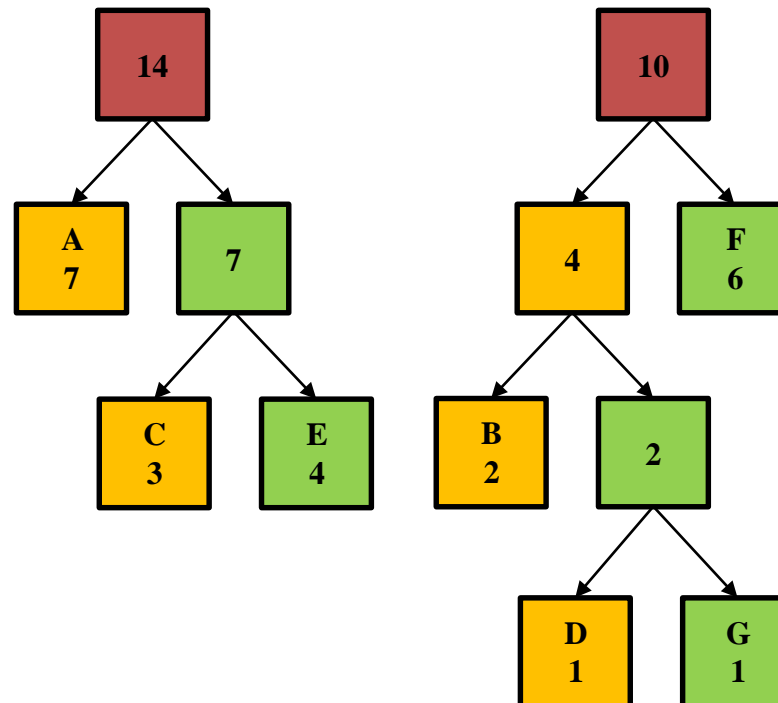
② 빈도가 가장 적은 2개의 문자의 합을 Root에 적는다.

③ 새로운 빈도수의 합을 기준으로 다시 ②과정 반복

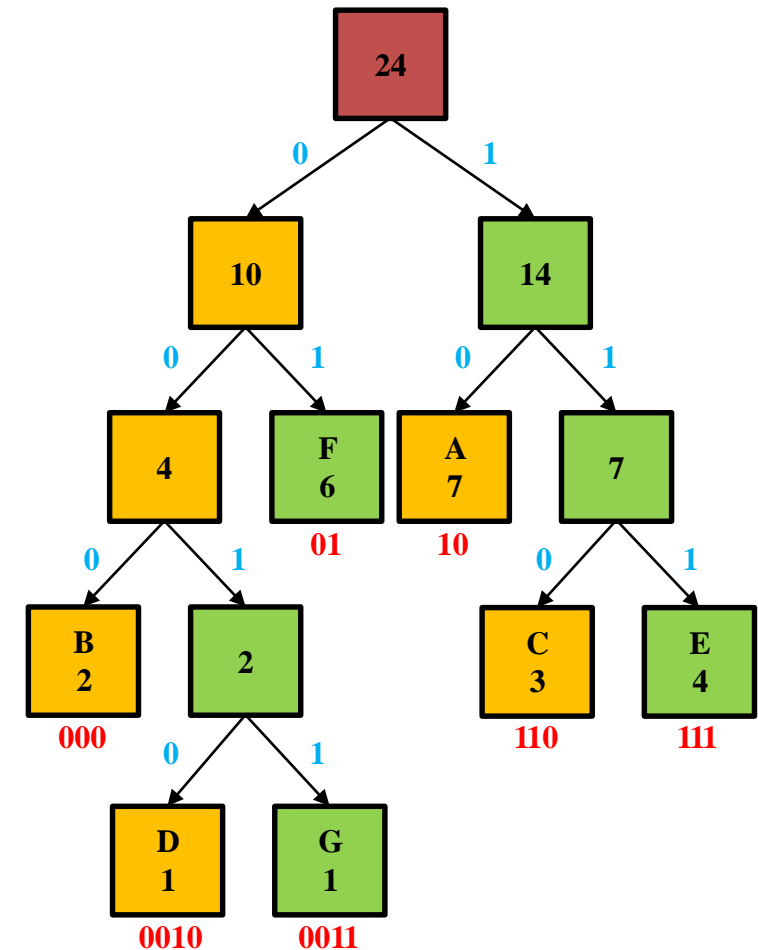
Step 5



Step 6



Final Tree



❖ 다음 데이터 Huffman coding을 통해 압축해보자.

AAAAAABBCCCCDEEEEEFFFFFG

① 데이터에서 사용되는 각 문자에 대한 출현 빈도수를 구한다.

문자	A	B	C	D	E	F	G
출현빈도	7	2	3	1	4	6	1

② 빈도가 가장 적은 2개의 문자의 합을 Root에 적는다.

③ 새로운 빈도수의 합을 기준으로 다시 ②과정 반복

```
struct node
{
    string characters;
    unsigned int frequency;
    string code;
    node * leftChild;
    node * rightChild;
};
```

Characters
Frequency
Code

Root

L

R

A	B	C	D	E	F	G
7	2	3	1	4	6	1

```
node getHuffmanTree()
{
    while (!nodeArray.empty())
    {
        node * tempNode = new node; - Parent
        node * tempNode1 = new node; - Children
        node * tempNode2 = new node;
        *tempNode1 = extractMin();
        *tempNode2 = extractMin();

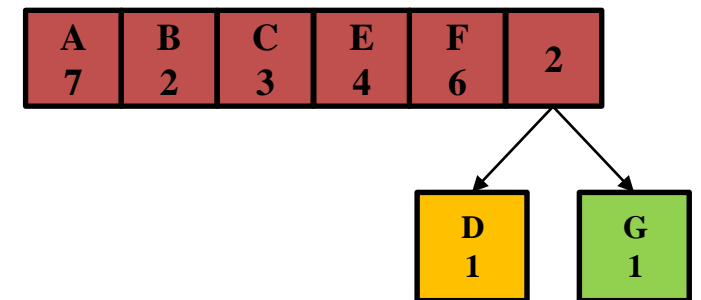
        tempNode->leftChild = tempNode1;
        tempNode->rightChild = tempNode2;
        tempNode->frequency = tempNode1->frequency + tempNode2->frequency;
        nodeArray.push_back(*tempNode);

        //Root Node만 남았으므로 Huffman Tree 완성
        if (nodeArray.size() == 1) break;
    }
    return nodeArray[0];
}
```

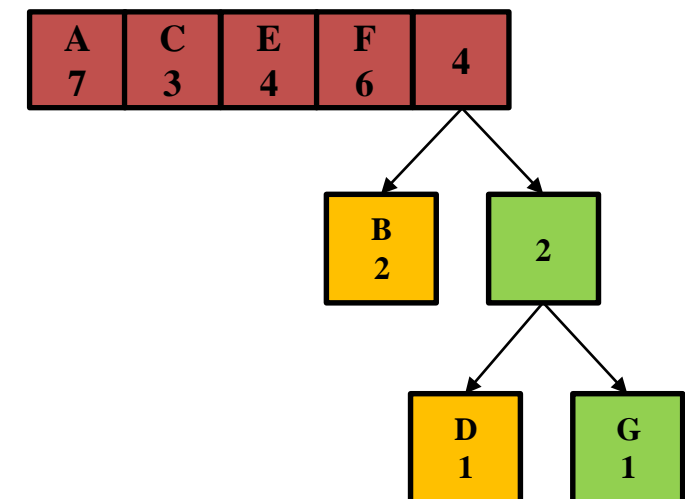
Step 1

A	B	C	D	E	F	G
7	2	3	1	4	6	1

Step 2



Step 3



❖ 다음 데이터 Huffman coding을 통해 압축해보자.

AAAAAABBCCCCDEEEEEEFFFFFFFG

① 데이터에서 사용되는 각 문자에 대한 출현 빈도수를 구한다.

문자	A	B	C	D	E	F	G
출현빈도	7	2	3	1	4	6	1

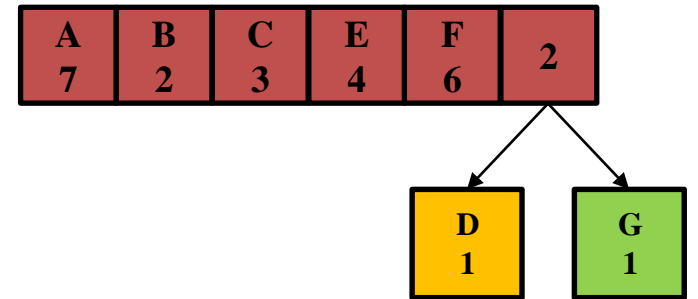
② 빈도가 가장 적은 2개의 문자의 합을 Root에 적는다.

③ 새로운 빈도수의 합을 기준으로 다시 ②과정 반복

Step 1

A	B	C	D	E	F	G
7	2	3	1	4	6	1

Step 2



```

struct node
{
    string characters;
    unsigned int frequency;
    string code;
    node * leftChild;
    node * rightChild;
};
    
```

A	B	C	D	E	F	G
7	2	3	1	4	6	1

```

node getHuffmanTree()
{
    while (!nodeArray.empty())
    {
        node * tempNode = new node;
        node * tempNode1 = new node;
        node * tempNode2 = new node;
        *tempNode1 = extractMin();
        *tempNode2 = extractMin();

        tempNode->leftChild = tempNode1;
        tempNode->rightChild = tempNode2;
        tempNode->frequency = tempNode1->frequency + tempNode2->frequency;
        nodeArray.push_back(*tempNode);

        //Root Node만 남았으므로 Huffman Tree 완성
        if (nodeArray.size() == 1) break;
    }
    return nodeArray[0];
}
    
```

- nodeArray에서 빈도수가 가장 적은
노드를 반환함.
- nodeArray에서 해당 노드 제거.

```

node extractMin()
{
    unsigned int min = UINT_MAX;
    vector<node>::iterator iter, position;
    for (iter = nodeArray.begin(); iter != nodeArray.end(); iter++)
    {
        if (min > (*iter).frequency)
        {
            position = iter;
            min = (*iter).frequency;
        }
    }

    node tempNode = (*position);
    nodeArray.erase(position);

    return tempNode;
}
    
```

- nodeArray 탐색하
여 빈도수가 최소인
노드 골라냄.

Characters
Frequency
Code

Root

L

R

❖ 다음 데이터 Huffman coding을 통해 압축해보자.

AAAAAAABBCCCCDEEEEEEFFFFFFFG

① 데이터에서 사용되는 각 문자에 대한 출현 빈도수를 구한다.

문자	A	B	C	D	E	F	G
출현빈도	7	2	3	1	4	6	1

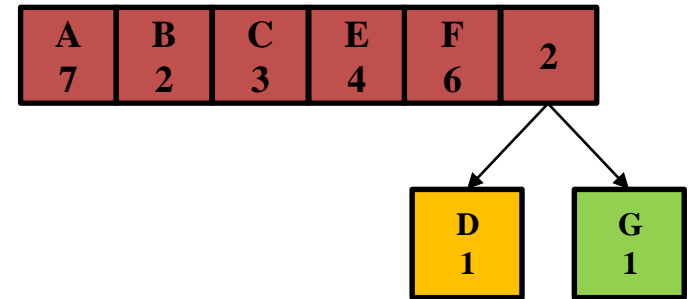
② 빈도가 가장 적은 2개의 문자의 합을 Root에 적는다.

③ 새로운 빈도수의 합을 기준으로 다시 ②과정 반복

Step 1

A	B	C	D	E	F	G
7	2	3	1	4	6	1

Step 2



```

struct node
{
    string characters;
    unsigned int frequency;
    string code;
    node * leftChild;
    node * rightChild;
};
  
```

Characters
Frequency
Code

Root

L

R

```

node getHuffmanTree()
{
    while (!nodeArray.empty())
    {
        node * tempNode = new node;
        node * tempNode1 = new node;
        node * tempNode2 = new node;
        *tempNode1 = extractMin();
        *tempNode2 = extractMin();

        tempNode->leftChild = tempNode1;
        tempNode->rightChild = tempNode2;
        tempNode->frequency = tempNode1->frequency + tempNode2->frequency;
        nodeArray.push_back(*tempNode);

        //Root Node만 남았으므로 Huffman Tree 완성
        if (nodeArray.size() == 1) break;
    }
    return nodeArray[0];
}
  
```

- nodeArray에서 빈도수가 가장 적은
노드를 반환함.
- nodeArray에서 해당 노드 제거.

```

node extractMin()
{
    unsigned int min = UINT_MAX;
    vector<node>::iterator iter, position;
    for (iter = nodeArray.begin(); iter != nodeArray.end(); iter++)
    {
        if (min > (*iter).frequency)
        {
            position = iter;
            min = (*iter).frequency;
        }
    }

    node tempNode = (*position);
    nodeArray.erase(position);

    return tempNode;
}
  
```

- 해당 노드 nodeArray
에서 제거

❖ 다음 데이터 Huffman coding을 통해 압축해보자.

AAAAAAABBCCCCDEEEEEEFFFFFFFG

① 데이터에서 사용되는 각 문자에 대한 출현 빈도수를 구한다.

문자	A	B	C	D	E	F	G
출현빈도	7	2	3	1	4	6	1

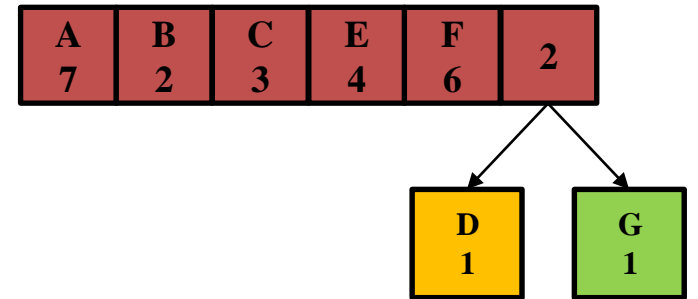
② 빈도가 가장 적은 2개의 문자의 합을 Root에 적는다.

③ 새로운 빈도수의 합을 기준으로 다시 ②과정 반복

Step 1

A	B	C	D	E	F	G
7	2	3	1	4	6	1

Step 2



```

struct node
{
    string characters;
    unsigned int frequency;
    string code;
    node * leftChild;
    node * rightChild;
};
  
```

A	B	C	D	E	F	G
7	2	3	1	4	6	1

```

node getHuffmanTree()
{
    while (!nodeArray.empty())
    {
        node * tempNode = new node;
        node * tempNode1 = new node;
        node * tempNode2 = new node;
        *tempNode1 = extractMin();
        *tempNode2 = extractMin();

        tempNode->leftChild = tempNode1;
        tempNode->rightChild = tempNode2;
        tempNode->frequency = tempNode1->frequency + tempNode2->frequency;
        nodeArray.push_back(*tempNode);

        //Root Node만 남았으므로 Huffman Tree 완성
        if (nodeArray.size() == 1) break;
    }
    return nodeArray[0];
}
  
```

- nodeArray에서 빈도수가 가장 적은
노드를 반환함.
- nodeArray에서 해당 노드 제거.

```

node extractMin()
{
    unsigned int min = UINT_MAX;
    vector<node>::iterator iter, position;
    for (iter = nodeArray.begin(); iter != nodeArray.end(); iter++)
    {
        if (min > (*iter).frequency)
        {
            position = iter;
            min = (*iter).frequency;
        }
    }

    node tempNode = (*position);
    nodeArray.erase(position);

    return tempNode;
}
  
```

- 빈도수 가장 적은 노드 반환

Characters
Frequency
Code

Root

L

R

❖ 다음 데이터 Huffman coding을 통해 압축해보자.

AAAAAAABBCCCCDEEEEEEFFFFFFFG

① 데이터에서 사용되는 각 문자에 대한 출현 빈도수를 구한다.

문자	A	B	C	D	E	F	G
출현빈도	7	2	3	1	4	6	1

② 빈도가 가장 적은 2개의 문자의 합을 Root에 적는다.

③ 새로운 빈도수의 합을 기준으로 다시 ②과정 반복

```
struct node
{
    string characters;
    unsigned int frequency;
    string code;
    node * leftChild;
    node * rightChild;
};
```

Characters
Frequency
Code

Root

L

R

A	B	C	D	E	F	G
7	2	3	1	4	6	1

```
node getHuffmanTree()
{
    while (!nodeArray.empty())
    {
        node * tempNode = new node;
        node * tempNode1 = new node;
        node * tempNode2 = new node;
        *tempNode1 = extractMin();
        *tempNode2 = extractMin();

        tempNode->leftChild = tempNode1;
        tempNode->rightChild = tempNode2;
        tempNode->frequency = tempNode1->frequency + tempNode2->frequency;
        nodeArray.push_back(*tempNode);

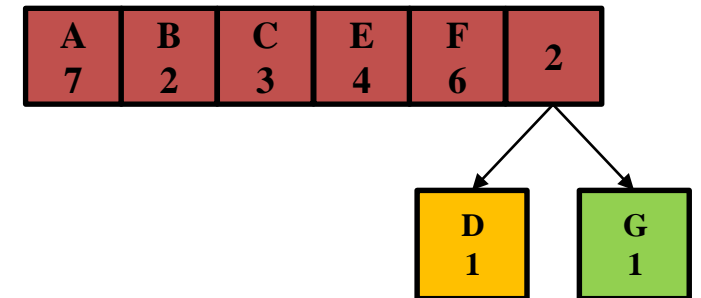
        //Root Node만 남았으므로 Huffman Tree 완성
        if (nodeArray.size() == 1) break;
    }
    return nodeArray[0];
}
```

- 부모 노드에 자식
노드의 주소값 배정

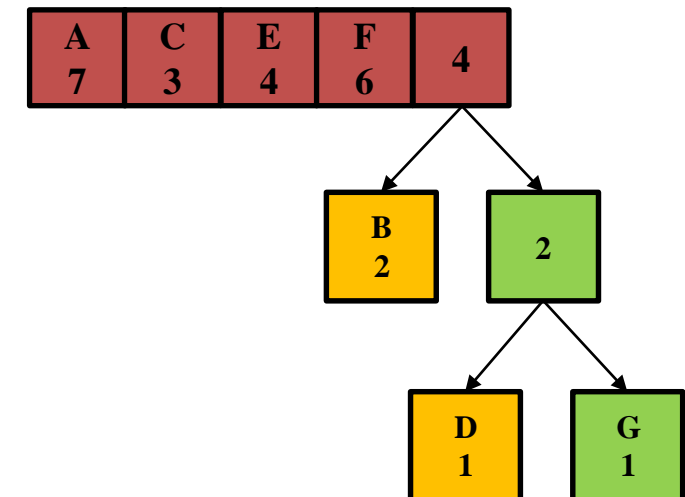
Step 1

A	B	C	D	E	F	G
7	2	3	1	4	6	1

Step 2



Step 3



❖ 다음 데이터 Huffman coding을 통해 압축해보자.

AAAAAAABBCCCCDEEEEEEFFFFFFFG

① 데이터에서 사용되는 각 문자에 대한 출현 빈도수를 구한다.

문자	A	B	C	D	E	F	G
출현빈도	7	2	3	1	4	6	1

② 빈도가 가장 적은 2개의 문자의 합을 Root에 적는다.

③ 새로운 빈도수의 합을 기준으로 다시 ②과정 반복

```
struct node
{
    string characters;
    unsigned int frequency;
    string code;
    node * leftChild;
    node * rightChild;
};
```

Characters
Frequency
Code

Root

L

R

A	B	C	D	E	F	G
7	2	3	1	4	6	1

```
node getHuffmanTree()
{
    while (!nodeArray.empty())
    {
        node * tempNode = new node;
        node * tempNode1 = new node;
        node * tempNode2 = new node;
        *tempNode1 = extractMin();
        *tempNode2 = extractMin();

        tempNode->leftChild = tempNode1;
        tempNode->rightChild = tempNode2;
        tempNode->frequency = tempNode1->frequency + tempNode2->frequency;
        nodeArray.push_back(*tempNode);

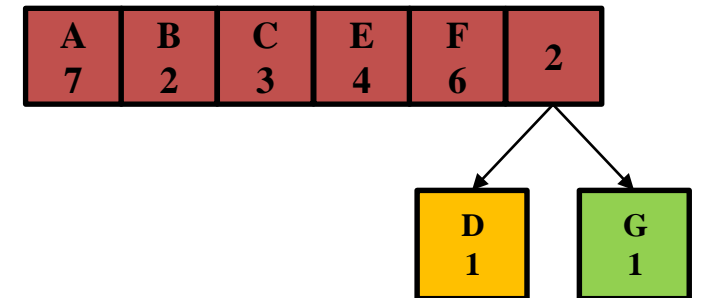
        //Root Node만 남았으므로 Huffman Tree 완성
        if (nodeArray.size() == 1) break;
    }
    return nodeArray[0];
}
```

- 빈도수 합 설정 후
nodeArray에 삽입

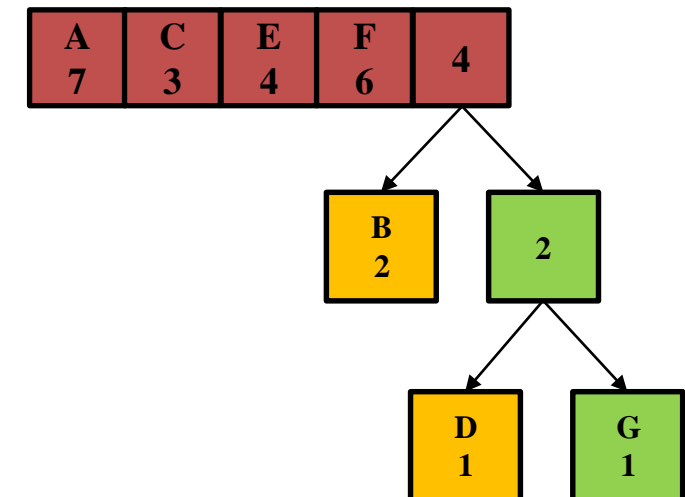
Step 1

A	B	C	D	E	F	G
7	2	3	1	4	6	1

Step 2



Step 3



❖ 다음 데이터 Huffman coding을 통해 압축해보자.

AAAAAAABBCCCCDEEEEEEFFFFFG

① 데이터에서 사용되는 각 문자에 대한 출현 빈도수를 구한다.

문자	A	B	C	D	E	F	G
출현빈도	7	2	3	1	4	6	1

② 빈도가 가장 적은 2개의 문자의 합을 Root에 적는다.

③ 새로운 빈도수의 합을 기준으로 다시 ②과정 반복

```
struct node
{
    string characters;
    unsigned int frequency;
    string code;
    node * leftChild;
    node * rightChild;
};
```

Characters
Frequency
Code

Root

L

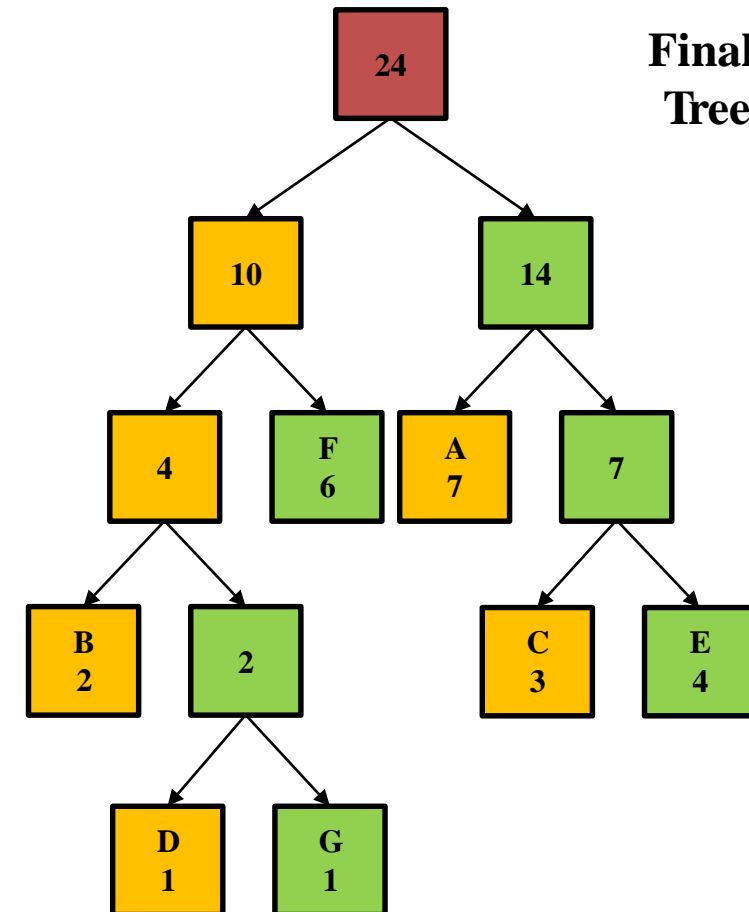
R

A	B	C	D	E	F	G
7	2	3	1	4	6	1

```
node getHuffmanTree()
{
    while (!nodeArray.empty())
    {
        node * tempNode = new node;
        node * tempNode1 = new node;
        node * tempNode2 = new node;
        *tempNode1 = extractMin();
        *tempNode2 = extractMin();

        tempNode->leftChild = tempNode1;
        tempNode->rightChild = tempNode2;
        tempNode->frequency = tempNode1->frequency + tempNode2->frequency;
        nodeArray.push_back(*tempNode);

        //Root Node만 남았으므로 Huffman Tree 완성
        if (nodeArray.size() == 1) break;
    }
    return nodeArray[0]; - Root Node 반환
}
```



❖ 다음 데이터 Huffman coding을 통해 압축해보자.

AAAAAABBCCCCDEEEEEEFFFFFFFG

④ 각 가지의 왼쪽에는 0, 오른쪽에는 1을 쓴다. (Depth First Search를 이용하였다.)

```
void depthFirstSearch(node* tempRoot, string s)
{
    node* root1 = tempRoot;
    root1->code = s;
    if (root1 == NULL)
    {
    }
    else if (root1->leftChild == NULL && root1->rightChild == NULL)
    {
        cout << "Wt" << root1->characters << "Wt" << root1->code << endl;
    }
    else
    {
        root1->leftChild->code = s.append("0");
        s.erase(s.end() - 1);
        root1->rightChild->code = s.append("1");
        s.erase(s.end() - 1);

        depthFirstSearch(root1->leftChild, s.append("0"));
        s.erase(s.end() - 1);
        depthFirstSearch(root1->rightChild, s.append("1"));
        s.erase(s.end() - 1);
    }
}
```

- 각 노드별 코드 부여하는 재귀함수

```
//Huffman Coding Table 생성
depthFirstSearch(&root, "");
```

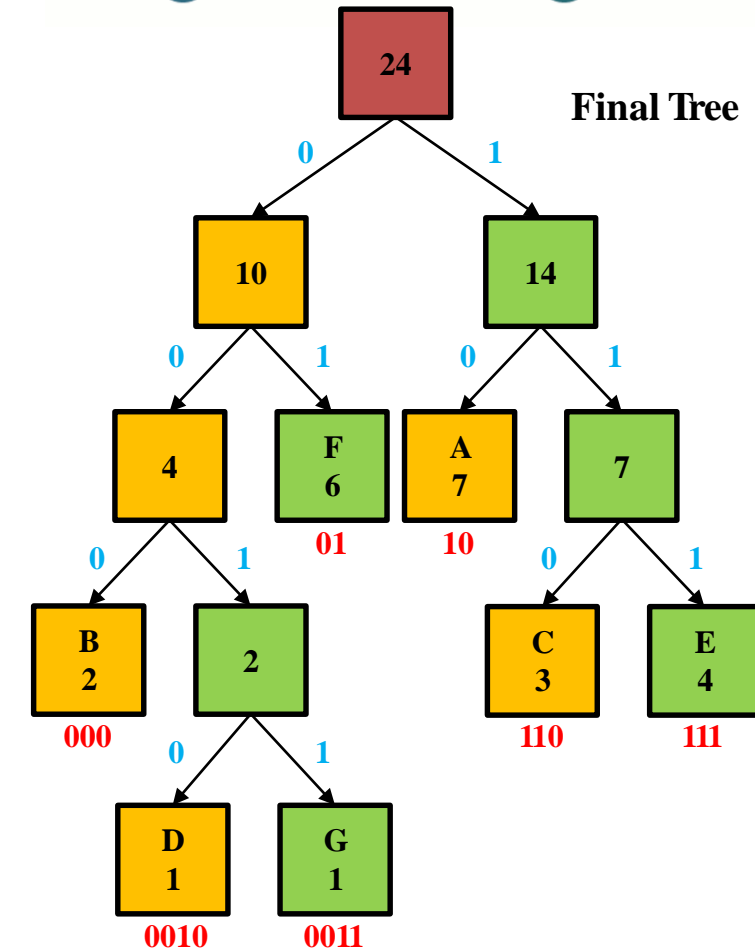
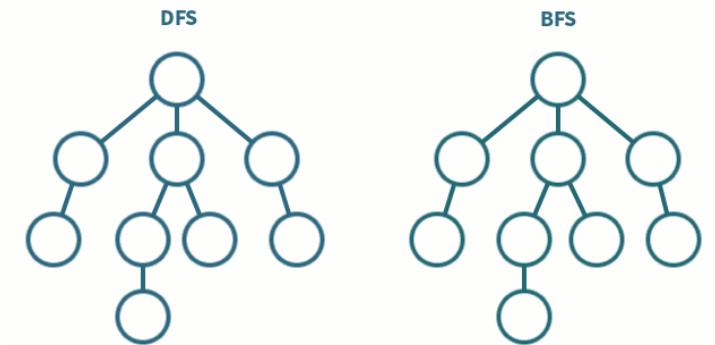
Microsoft Visual Studio 디버그 콘솔

Huffman Tree : 7

문자 빈도

A	7
B	2
C	3
D	1
E	4
F	6
G	1

문자	Code
B	000
D	0010
G	0011
F	01
A	10
C	110
E	111



❖ 다음 데이터 Huffman coding을 통해 압축해보자.

AAAAAAABBCCCEEEEEFFFFFG

④ 각 가지의 왼쪽에는 0, 오른쪽에는 1을 쓴다. (Depth First Search를 이용하였다.)

```
void depthFirstSearch(node* tempRoot, string s)
{
    node* root1 = tempRoot;

    root1->code = s; - 허프만 코드 (시작할 때는 비어있음)
    if (root1 == NULL)
    {
    }
    else if (root1->leftChild == NULL && root1->rightChild == NULL)
    {
        cout << "Wt" << root1->characters << "Wt" << root1->code << endl;
    }
    else
    {
        root1->leftChild->code = s.append("0");
        s.erase(s.end() - 1);
        root1->rightChild->code = s.append("1");
        s.erase(s.end() - 1);

        depthFirstSearch(root1->leftChild, s.append("0"));
        s.erase(s.end() - 1);
        depthFirstSearch(root1->rightChild, s.append("1"));
        s.erase(s.end() - 1);
    }
}
```

```
//Huffman Coding Table 생성
depthFirstSearch(&root, "");
```

Microsoft Visual Studio 디버그 콘솔

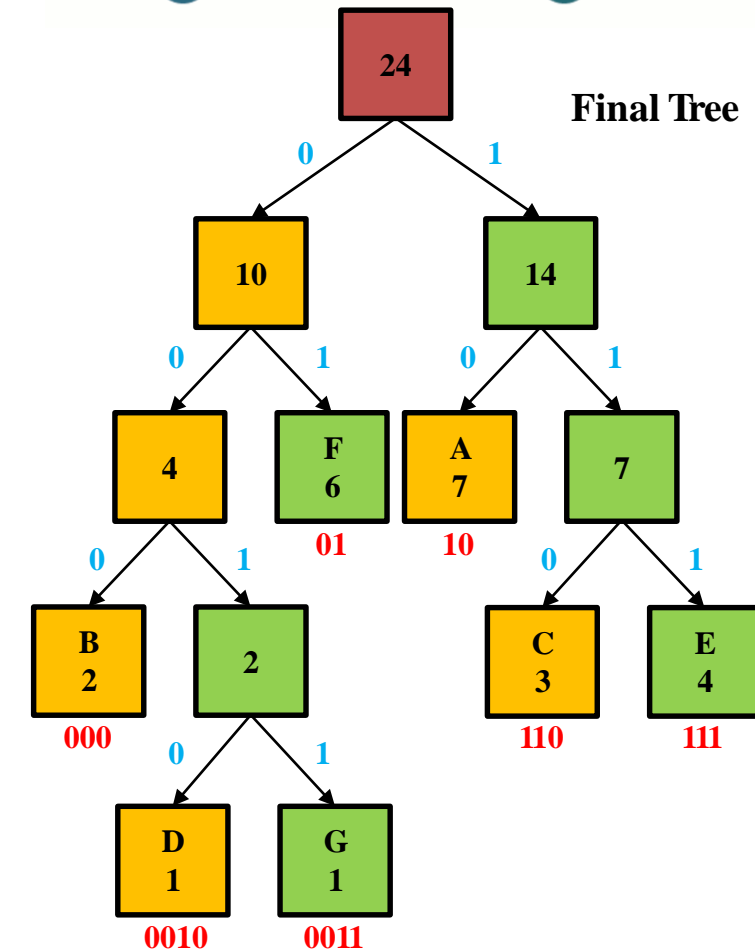
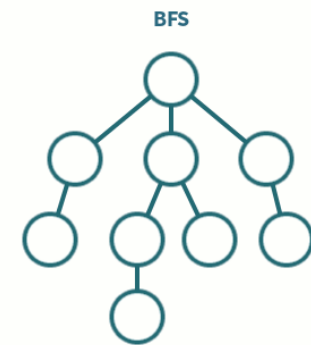
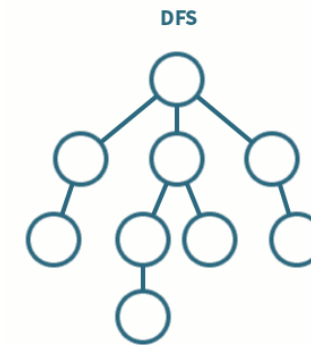
Huffman Tree : 7

문자 빈도

A 7
B 2
C 3
D 1
E 4
F 6
G 1

문자 Code

문자	Code
B	000
D	0010
G	0011
F	01
A	10
C	110
E	111





❖ 다음 데이터 Huffman coding을 통해 압축해보자.

AAAAAAABBCCCCDEEEEEEFFFFFFFG

④ 각 가지의 왼쪽에는 0, 오른쪽에는 1을 쓴다. (Depth First Search를 이용하였다.)

```
void depthFirstSearch(node* tempRoot, string s)
{
    node* root1 = tempRoot;

    root1->code = s;
    if (root1 == NULL)
    {
    }
    else if (root1->leftChild == NULL && root1->rightChild == NULL)
    {
        cout << "Wt" << root1->characters << "Wt" << root1->code << endl;
    }
    else
    {
        root1->leftChild->code = s.append("0");
        s.erase(s.end() - 1);
        root1->rightChild->code = s.append("1");
        s.erase(s.end() - 1);

        depthFirstSearch(root1->leftChild, s.append("0"));
        s.erase(s.end() - 1);
        depthFirstSearch(root1->rightChild, s.append("1"));
        s.erase(s.end() - 1);
    }
}
```

-s("")에 0 붙인 후
왼쪽 자식에게 코드 부여.
-s("0")의 마지막
인덱스 제거.

Microsoft Visual Studio 디버그 콘솔

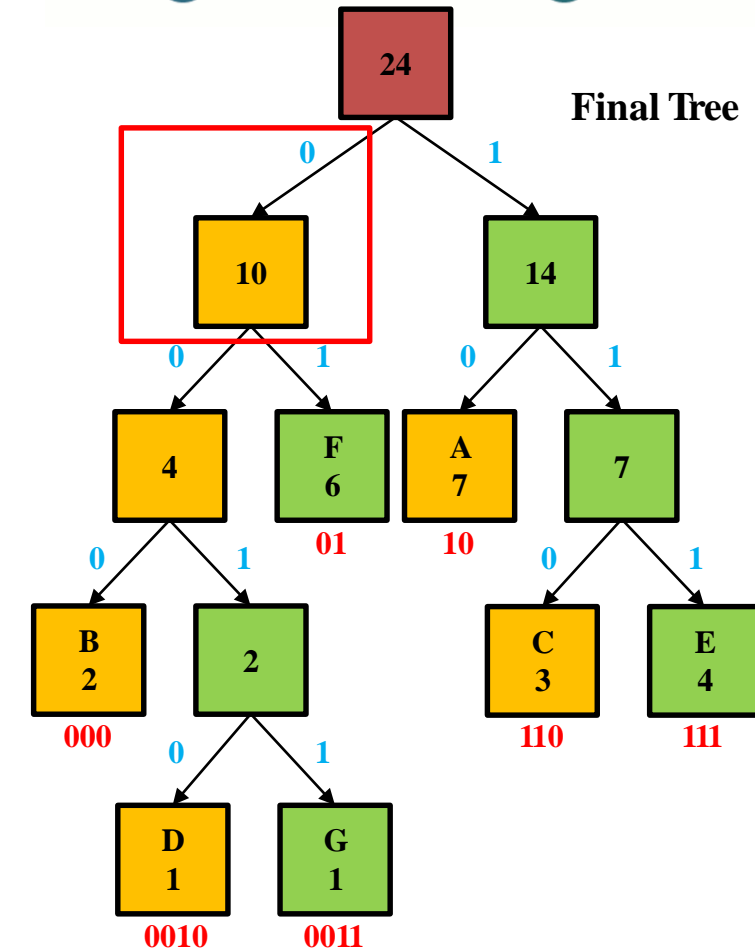
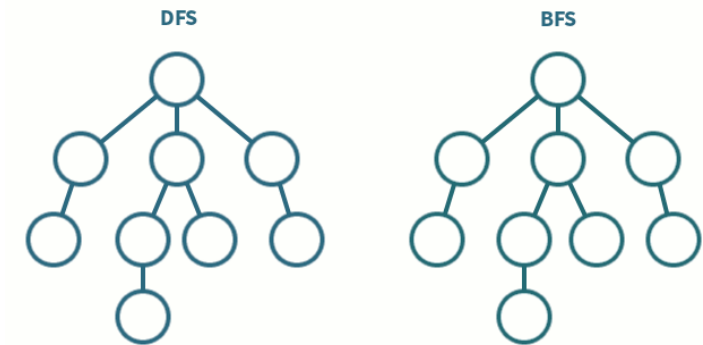
Huffman Tree : 7

문자 빈도

A	7
B	2
C	3
D	1
E	4
F	6
G	1

문자 Code

B	000
D	0010
G	0011
F	01
A	10
C	110
E	111





❖ 다음 데이터 Huffman coding을 통해 압축해보자.

AAAAAAABBCCCEEEEEFFFFFFG

④ 각 가지의 왼쪽에는 0, 오른쪽에는 1을 쓴다. (Depth First Search를 이용하였다.)

```
void depthFirstSearch(node* tempRoot, string s)
{
    node* root1 = tempRoot;

    root1->code = s;
    if (root1 == NULL)
    {
    }
    else if (root1->leftChild == NULL && root1->rightChild == NULL)
    {
        cout << "Wt" << root1->characters << "Wt" << root1->code << endl;
    }
    else
    {
        root1->leftChild->code = s.append("0");
        s.erase(s.end() - 1);
        root1->rightChild->code = s.append("1");
        s.erase(s.end() - 1);

        depthFirstSearch(root1->leftChild, s.append("0"));
        s.erase(s.end() - 1);
        depthFirstSearch(root1->rightChild, s.append("1"));
        s.erase(s.end() - 1);
    }
}
```

- s("")에 1 붙인 후
오른쪽 자식에게
코드 부여.
- s("1")의 마지막
인덱스 제거.

Microsoft Visual Studio 디버그 콘솔

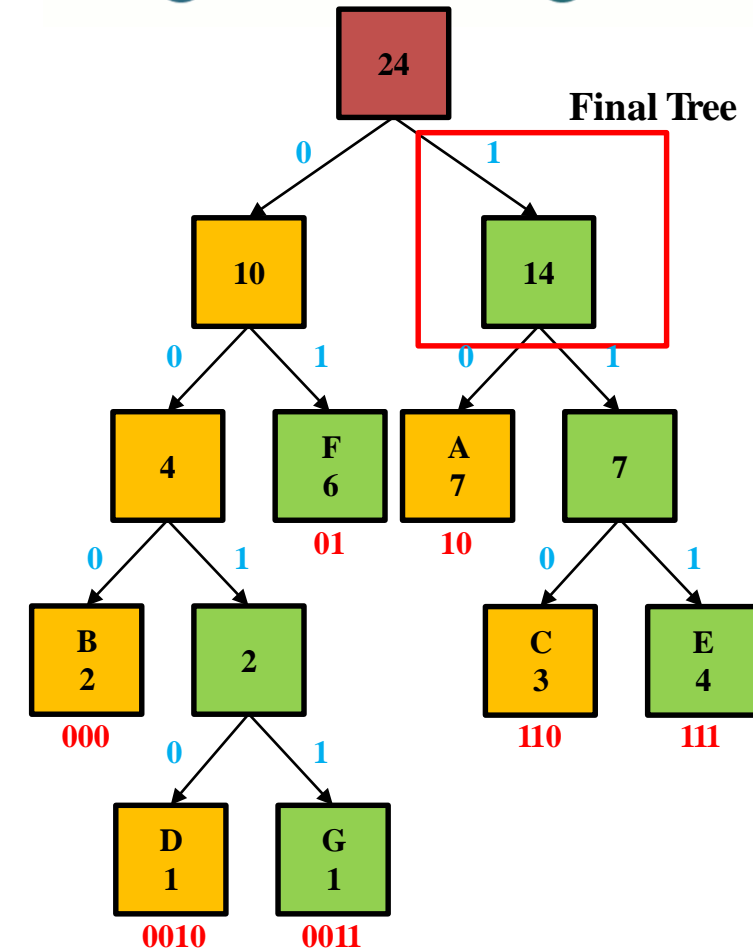
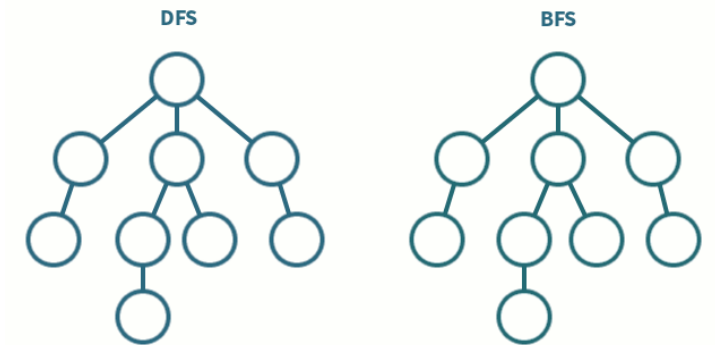
Huffman Tree : 7

문자 빈도

A	7
B	2
C	3
D	1
E	4
F	6
G	1

문자 Code

B	000
D	0010
G	0011
F	01
A	10
C	110
E	111



❖ 다음 데이터 Huffman coding을 통해 압축해보자.

AAAAAAABBCCCEEEEEFFFFF

④ 각 가지의 왼쪽에는 0, 오른쪽에는 1을 쓴다. (Depth First Search를 이용하였다.)

```
void depthFirstSearch(node* tempRoot, string s)
{
    node* root1 = tempRoot;

    root1->code = s;
    if (root1 == NULL)
    {
    }
    else if (root1->leftChild == NULL && root1->rightChild == NULL)
    {
        cout << "Wt" << root1->characters << "Wt" << root1->code << endl;
    }
    else
    {
        root1->leftChild->code = s.append("0");
        s.erase(s.end() - 1);
        root1->rightChild->code = s.append("1");
        s.erase(s.end() - 1);

        depthFirstSearch(root1->leftChild, s.append("0"));
        s.erase(s.end() - 1);
        depthFirstSearch(root1->rightChild, s.append("1"));
        s.erase(s.end() - 1);
    }
}
```

- 왼쪽 자식으로
depthFirstSearch
실행.
- 왼쪽 자식의 현재
코드는 "0"

Microsoft Visual Studio 디버그 콘솔

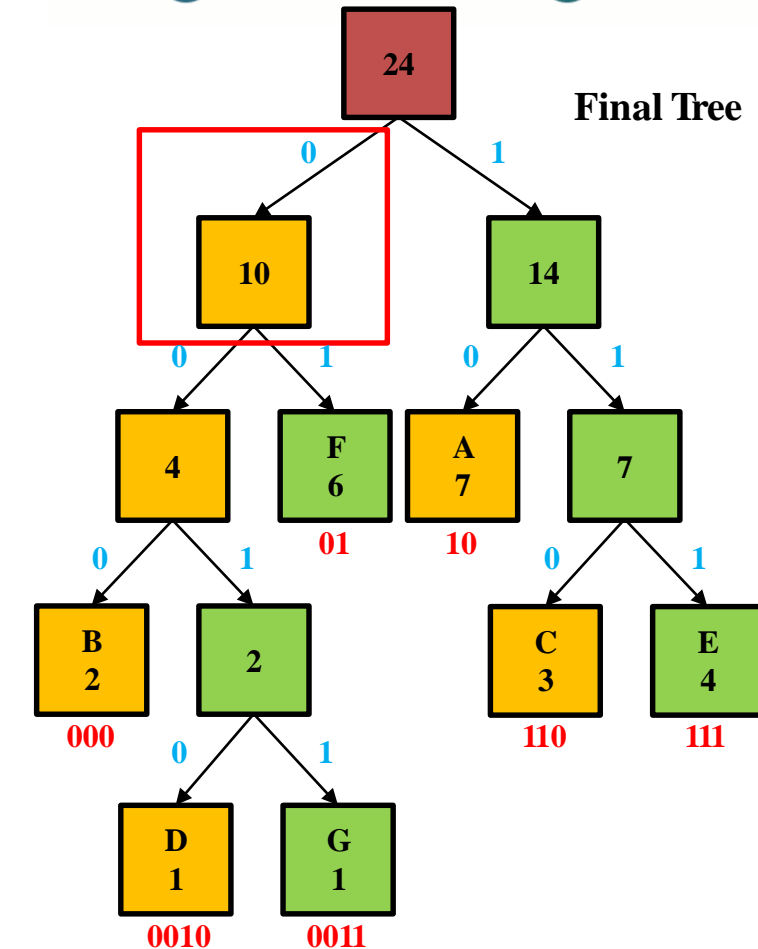
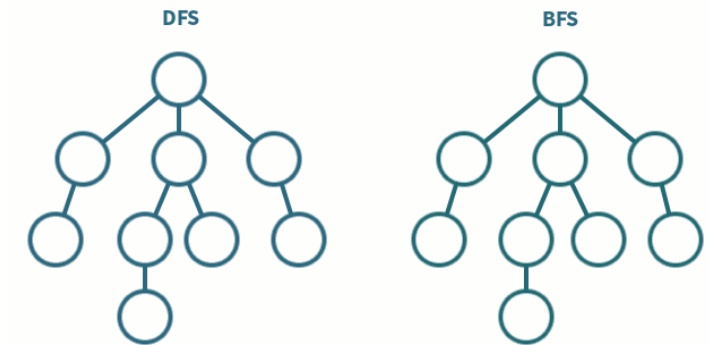
Huffman Tree : 7

문자 빈도

A	7
B	2
C	3
D	1
E	4
F	6
G	1

문자 Code

B	000
D	0010
G	0011
F	01
A	10
C	110
E	111



❖ 다음 데이터 Huffman coding을 통해 압축해보자.

AAAAAAABBCCCEEEEEFFFFFG

④ 각 가지의 왼쪽에는 0, 오른쪽에는 1을 쓴다. (Depth First Search를 이용하였다.)

```
void depthFirstSearch(node* tempRoot, string s)
{
    node* root1 = tempRoot;

    root1->code = s;
    if (root1 == NULL)
    {
    }
    else if (root1->leftChild == NULL && root1->rightChild == NULL)
    {
        cout << "Wt" << root1->characters << "Wt" << root1->code << endl;
    }
    else
    {
        root1->leftChild->code = s.append("0");
        s.erase(s.end() - 1);
        root1->rightChild->code = s.append("1");
        s.erase(s.end() - 1);

        depthFirstSearch(root1->leftChild, s.append("0"));
        s.erase(s.end() - 1);
        depthFirstSearch(root1->rightChild, s.append("1"));
        s.erase(s.end() - 1);
    }
}
```

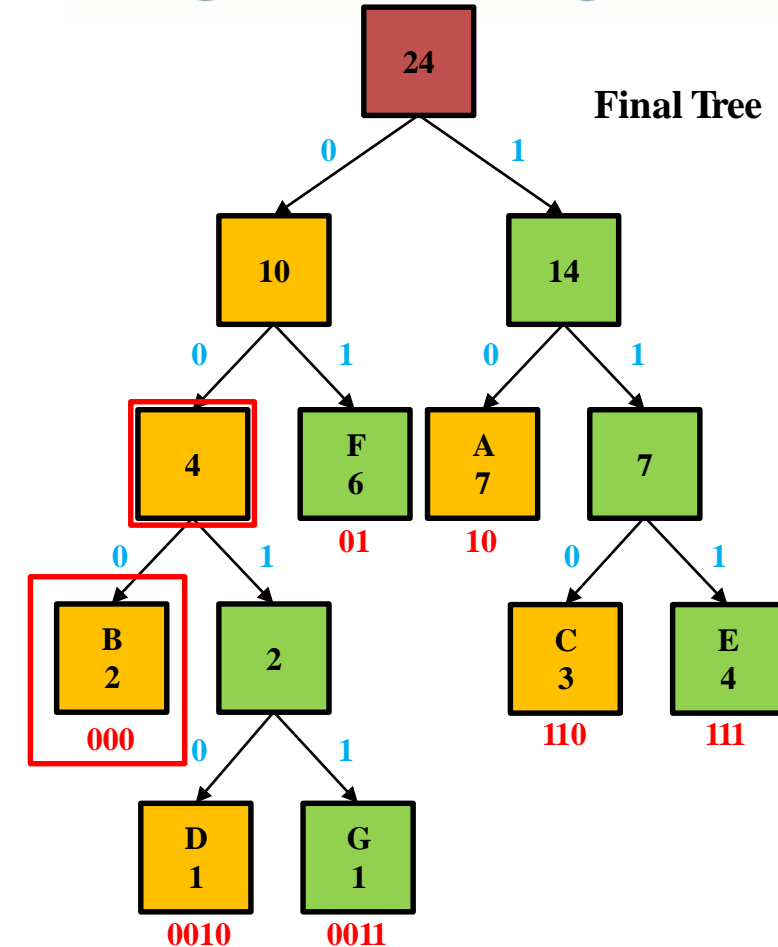
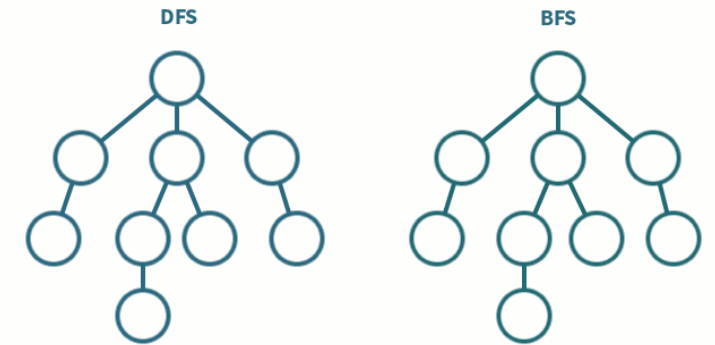
Microsoft Visual Studio 디버그 콘솔

Huffman Tree : 7

문자 빈도

A	7
B	2
C	3
D	1
E	4
F	6
G	1

문자	Code
B	000
D	0010
G	0011
F	01
A	10
C	110
E	111



❖ 다음 데이터 Huffman coding을 통해 압축해보자.

AAAAAAABBCCCEEEEEFFFFFG

④ 각 가지의 왼쪽에는 0, 오른쪽에는 1을 쓴다. (Depth First Search를 이용하였다.)

```
void depthFirstSearch(node* tempRoot, string s)
{
    node* root1 = tempRoot;

    root1->code = s;
    if (root1 == NULL)
    {
    }
    else if (root1->leftChild == NULL && root1->rightChild == NULL)
    {
        cout << "Wt" << root1->characters << "Wt" << root1->code << endl;
    }
    else
    {
        root1->leftChild->code = s.append("0");
        s.erase(s.end() - 1);
        root1->rightChild->code = s.append("1");
        s.erase(s.end() - 1);

        depthFirstSearch(root1->leftChild, s.append("0"));
        s.erase(s.end() - 1);
        depthFirstSearch(root1->rightChild, s.append("1"));
        s.erase(s.end() - 1);
    }
}
```

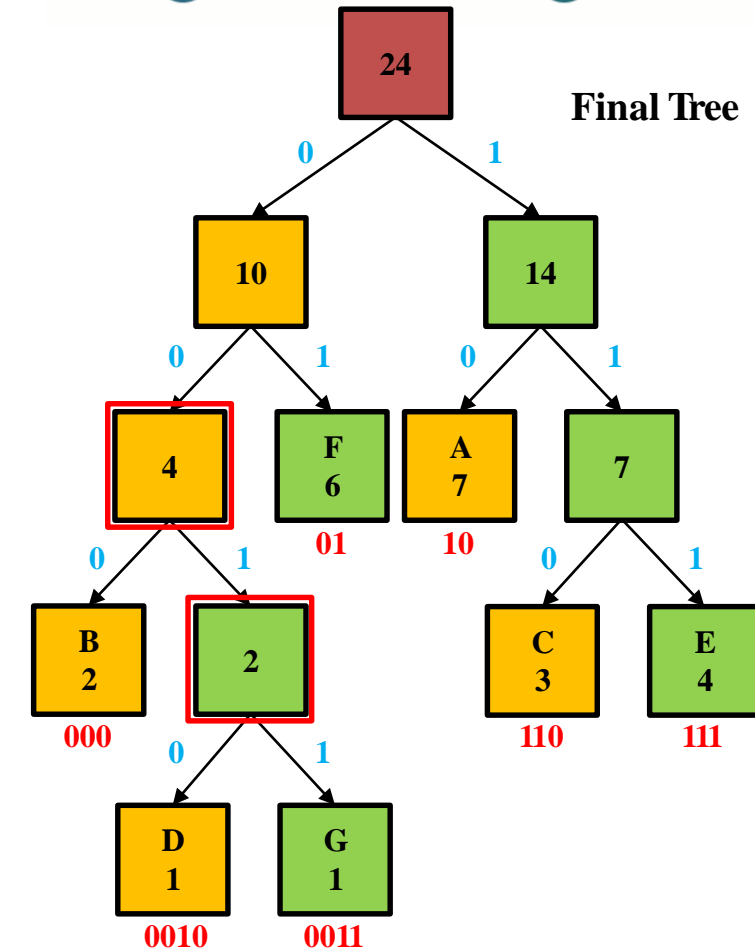
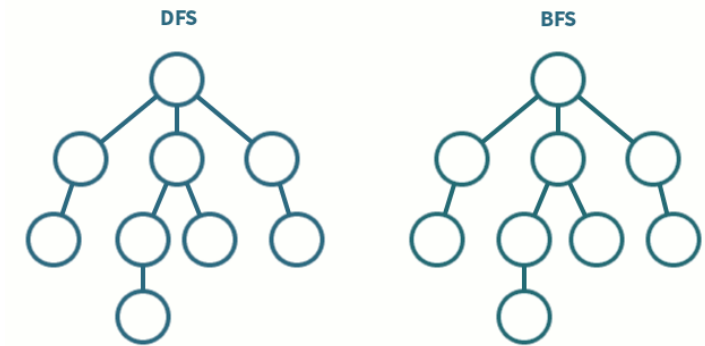
Microsoft Visual Studio 디버그 콘솔

Huffman Tree : 7

문자 빈도

A	7
B	2
C	3
D	1
E	4
F	6
G	1

문자	Code
B	000
D	0010
G	0011
F	01
A	10
C	110
E	111



❖ 다음 데이터 Huffman coding을 통해 압축해보자.

AAAAAAABBCCCCDEEEEEFFFFF

요약.

- ① 캐릭터 종류는 몇가지인가?
- ② 각 캐릭터별 빈도수 입력 받고 노드를 생성한다.
- ③ 빈도수에 따른 hierarchy로 binary search tree를 생성한다.
- ④ Root부터 시작하여 각 노드별 huffman code를 부여한다.

①

②

③

④

```
void getHuffmanCode()
{
    int size;
    unsigned int tempInt;
    char alphabet;

    //데이터에 사용되는 문자의 종류 개수 입력
    cout << endl;
    cout << "Huffman Tree : ";
    cin >> size;

    cout << endl << endl;
    cout << "문자 빈도" << endl;
    cout << "-----" << endl;

    //각 문자별 빈도 수 노드 생성
    for (int i = 0; i < size; i++)
    {
        cout << "H";
        node tempNode;
        cin >> alphabet;
        cin >> tempInt;

        tempNode.characters = alphabet;
        tempNode.frequency = tempInt;
        tempNode.leftChild = NULL;
        tempNode.rightChild = NULL;
        nodeArray.push_back(tempNode);
    }

    //Huffman Tree 생성
    node root = getHuffmanTree();

    cout << endl << endl;
    cout << "문자" << "H" << "Code" << endl;
    cout << "-----" << endl;

    //Huffman Coding Table 생성
    depthFirstSearch(&root, "");
}
```


❖ Assignment

- Realization of **Adaptive Huffman Coding** in the lecture note

Media Lab.
KyungHee University

– Adaptive Huffman coding for AADCCDD

"A"

"AA"

"AAD"

"AADC"

18

Media Lab.
KyungHee University

"AADCC" Step 1

"AADCC" Step 2

"AADCC" Step 3

"AADCCD"

"AADCCDD"

19

```
#include "adpative_huffman.h"

int main()
{
    get_adaptive_huffman_code();

    return 0;
}
```

- 인터페이스는 우측 그림과 유사하게 만들어주세요

input : A

data	count
A	1

data	code
NEW	0
A	1

input : B

data	count
A	3
B	1

data	code
NEW	00
B	01
A	1



Lab 1.

제출일		학번	
전공		이름	

① 문제	
② 주요 변수	
③ 알고리즘	
④ 결과	

- 파일명 : "실감미디어_1st_학번_이름.zip"
 - EX) 실감미디어_1st_2023000100_홍길동.zip
- 압축 파일 구성
 - 소스 파일, .hwp(.doc) 2개 파일 (Python 의 경우)
 - 소스 파일, .hwp(.doc), .exe 3개 파일 (C/C++ 의 경우)
- 파일 내용
 - 보고서 : 이름, 학번, 문제, 주요 변수, 알고리즘, 결과
 - (.py, .ipynb)파이썬 소스코드 or (.cpp, .c) C/C++ 소스 코드
 - .exe : 실행파일 (C/C++ 의 경우만 해당)
- 주의 사항
 - 제출이 늦을 경우 감점은 있지만, 학기 종료까지 제출 하시면 점수가 있습니다.

조 교 : 권도완

이메일 : kdwys97@khu.ac.kr

연구실 : Media Lab.3 (전자정보대학 567호)



Appendix



Visual Studio Community Install (Window)

❖ Visual Studio

VS 마이크로 소프트에서 개발한 통합 개발 환경(IDE)

윈도우, 리눅스, macOS에서 작동하며, 다양한 언어(C, C#, C++, Python, node.js 등)로 프로그래밍이 가능합니다.

다운로드 페이지 : <https://visualstudio.microsoft.com/ko/free-developer-offers/>



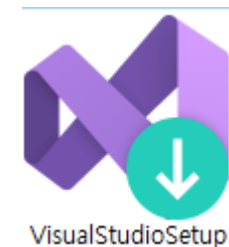
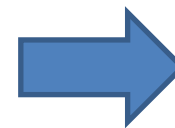
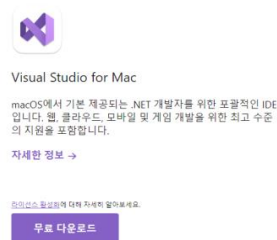
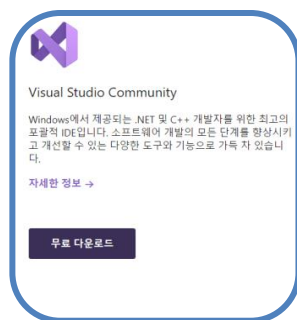
Visual Studio Community

❖ Visual Studio

Visual Studio Community : 개인 사용자용 무료 version

Visual Studio Professional : 유료 version

Visual Studio Enterprise : Professional로는 대규모 개발이 충분하지 않은 사용자를 위한 버전

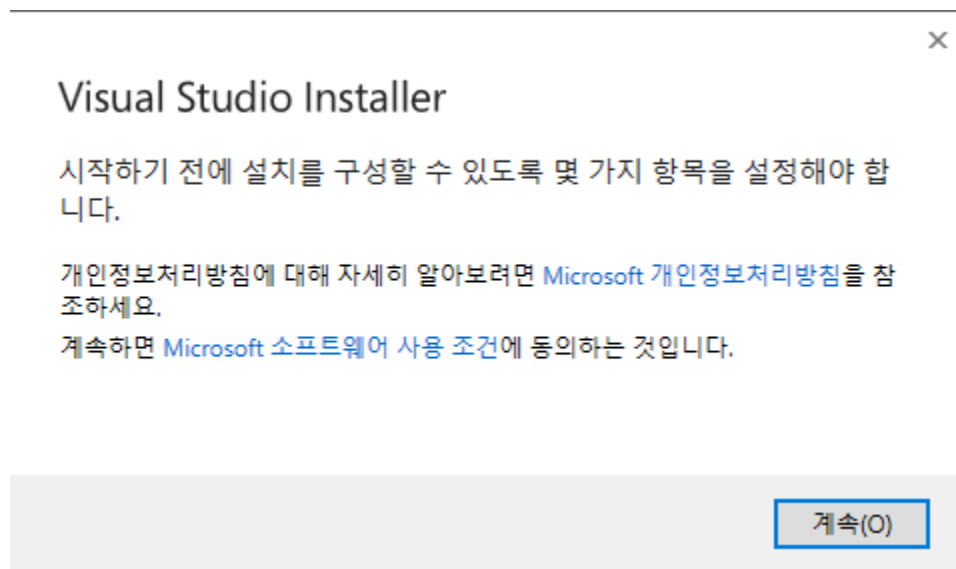


Microsoft 개발자 클라우드의
일부

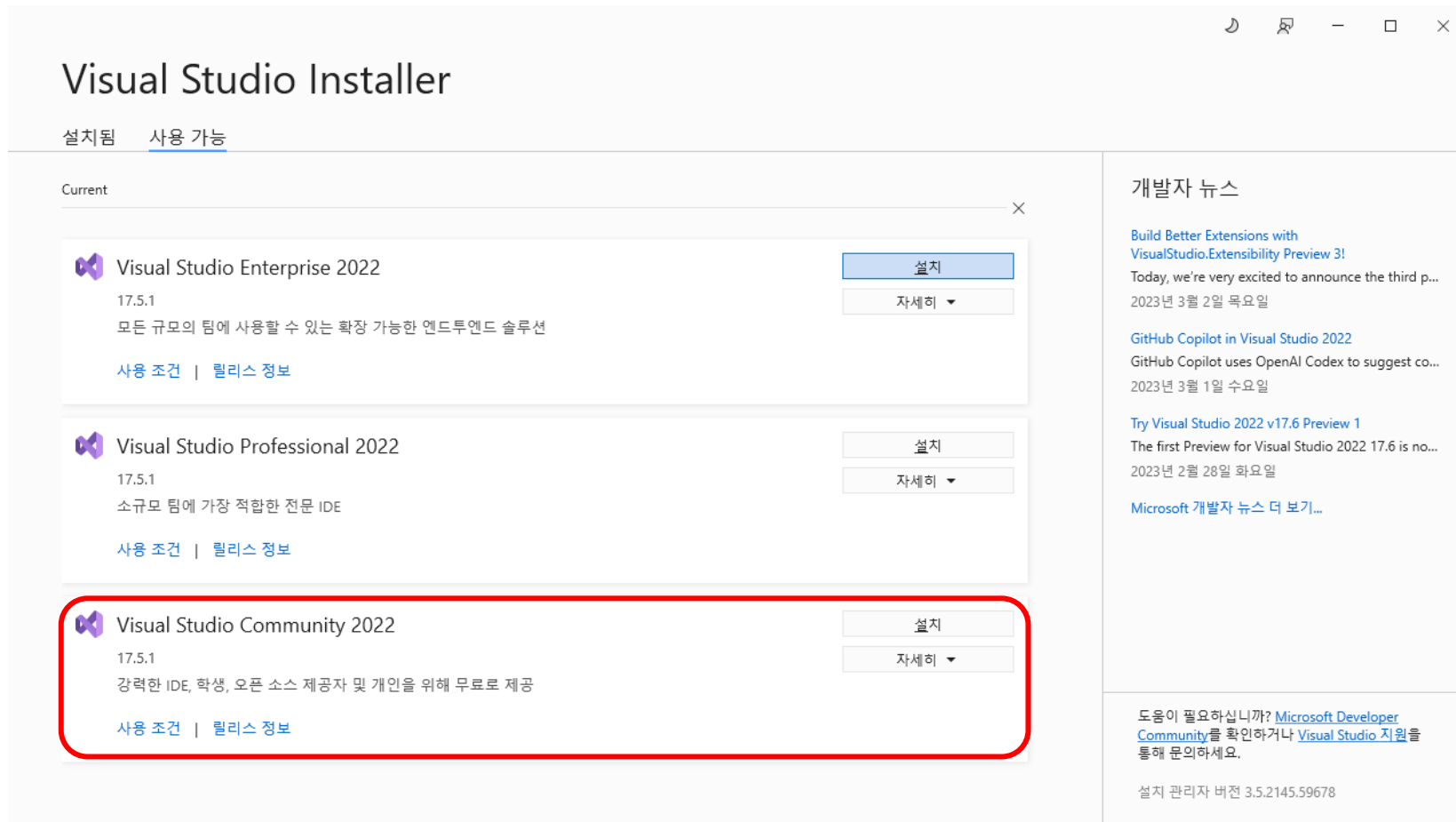
❖ Visual Studio(Window)

VisualStudioSetup.exe 파일 실행

해당 파일을 설치를 시작하면 아래와 같은 창 발생 : “계속” 클릭



❖ Visual Studio(Window)



❖ Visual Studio(Window)

설치 — Visual Studio Community 2022 — 17.5.1

워크로드 개별 구성 요소 언어 팩 설치 위치

1 설치할 항목을 선택하는 데 도움이 필요하신가요? [추가 정보](#)

Python 개발
Python에 대한 편집, 디버깅, 대화형 개발 및 소스 제어입니다.

Node.js 개발
비동기 이벤트 구동 JavaScript 런타임인 Node.js를 사용하여 확장 가능한 네트워크 애플리케이션을 빌드합니다.

데스크톱 및 모바일 (5)

.NET Multi-Platform App UI 개발
.NET MAUI와 함께 C#을 사용하여 단일 코드베이스에서 Android, iOS, Windows 및 Mac용 앱을 빌드합니다.

.NET 데스크톱 개발
.NET 및 .NET Framework와 함께 C#, Visual Basic 및 F#를 사용하여 WPF, Windows Forms 및 콘솔 애플리케이션을...

C++를 사용한 데스크톱 개발 ☒
MSVC, Clang, CMake 또는 MSBuild 등 선택한 도구를 사용하여 Windows용 최신 C++ 앱을 빌드합니다.

유니버설 Windows 플랫폼 개발 ☐
C#, VB 또는 C++(선택 사항)를 사용하여 유니버설 Windows 플랫폼용 애플리케이션을 만듭니다.

C++를 사용한 모바일 개발 ☐

위치
C:\Program Files\Microsoft Visual Studio\2022\Community [변경...](#)

계속하면 선택한 Visual Studio 버전에 대한 [라이선스](#)에 동의하게 됩니다. Microsoft는 Visual Studio와 함께 다른 소프트웨어를 다운로드할 수 있는 기능도 제공합니다. 이 소프트웨어는 [타사 고지 사항](#) 또는 해당 라이선스에 명시된 것처럼 별도로 라이선스가 허용됩니다. 계속하면 이러한 라이선스에도 동의하게 됩니다.

필요한 전체 공간 15.37GB

다운로드하는 동안 설치 **설치**

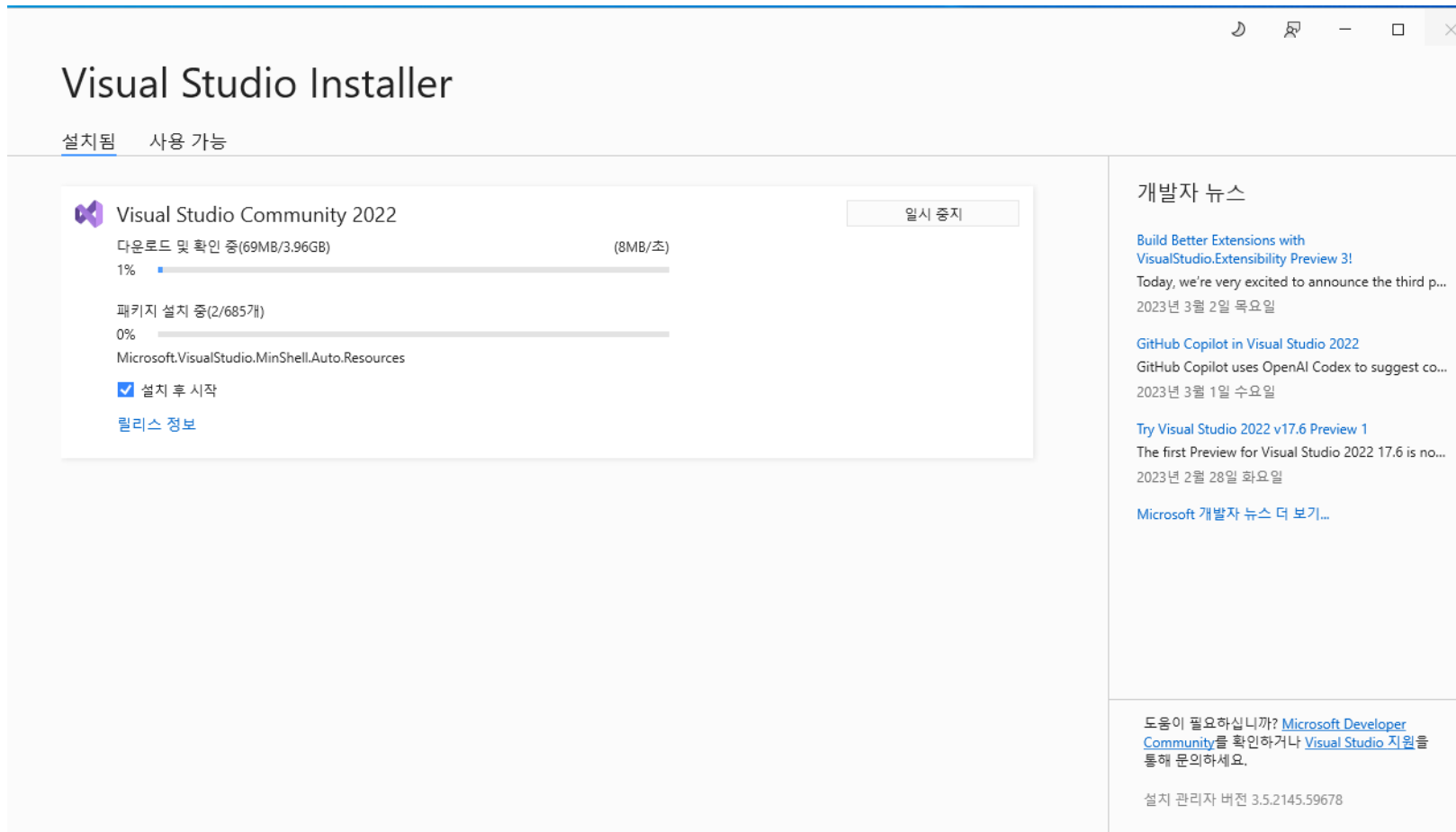
설치 세부 정보

C++를 사용한 데스크톱 개발

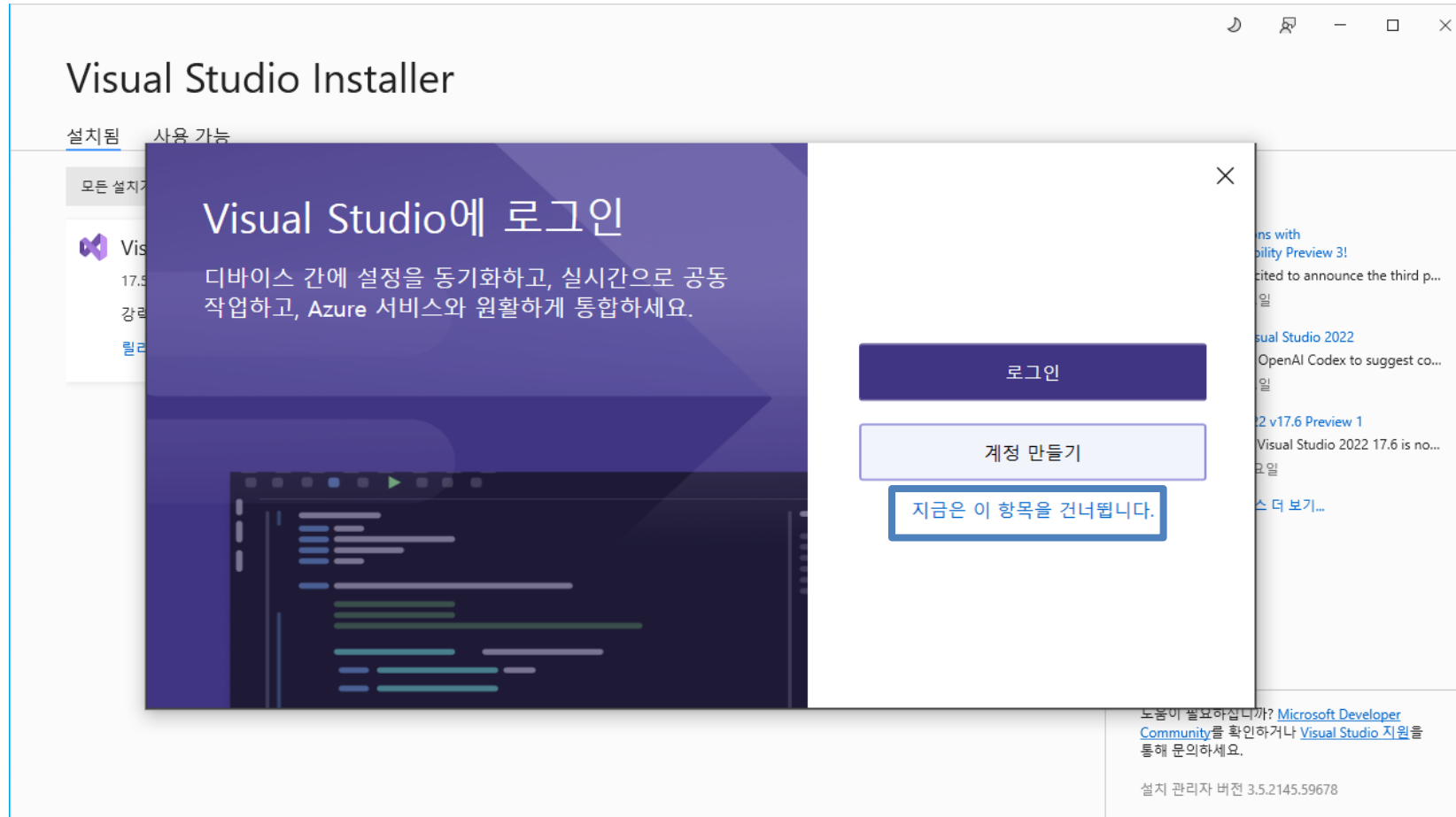
- 포함됨
 - ☒ C++ 핵심 데스크톱 기능
- 선택 사항
 - ☒ MSVC v143 - VS 2022 C++ x64/x86 빌드...
 - ☒ 최신 v143 빌드 도구용 C++ ATL(x86 및 x64)
 - ☒ Windows 11 SDK(10.0.22000.0)
 - ☒ 보안 문제 분석
 - ☒ Just-In-Time 디버거
 - ☒ C++ 프로파일링 도구
 - ☒ Windows용 C++ CMake 도구
 - ☒ Test Adapter for Boost.Test
 - ☒ Test Adapter for Google Test
 - ☒ Live Share
 - ☒ IntelliCode
 - ☒ C++ AddressSanitizer
 - ☒ 최신 v143 빌드 도구용 C++ MFC(x86 및 x64)
 - ☐ v143 빌드 도구용 C++ 모듈(x64/x86 - 실행...
 - ☐ Windows 11 SDK(10.0.22621.0)

지원되지 않는 구성 요소 제거(R)

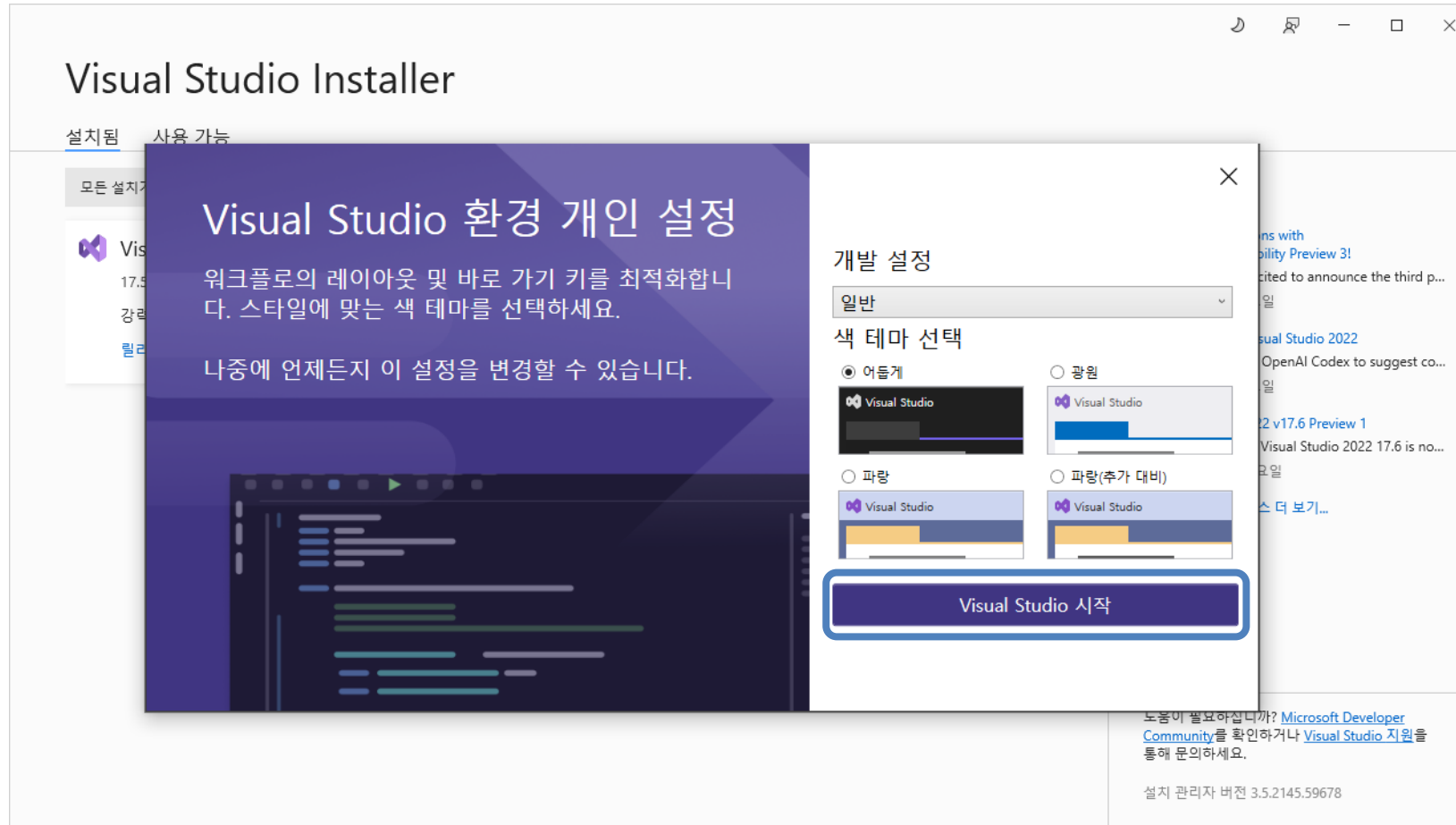
❖ Visual Studio(Window)



❖ Visual Studio(Window)



❖ Visual Studio(Window)



❖ Visual Studio(Window)



❖ Visual Studio(Window)



❖ Visual Studio(Window)

The image shows the 'New Project' (새 프로젝트 구성) dialog box in Visual Studio. It is titled '빈 프로젝트' (Empty Project) and shows 'C++' and 'Windows' as the selected templates. The dialog has three main sections, each highlighted with a blue box and a circled number:

- 1** The 'Project Name' (프로젝트 이름(I)) text box, which contains the text 'Example'.
- 2** The 'Solution Name' (솔루션 이름(M)) text box, which also contains the text 'Example'. Below this box is a checkbox labeled '솔루션 및 프로젝트를 같은 디렉터리에 배치(D)' (Place solution and project in the same directory).
- 3** The 'Make' (만들기(C)) button at the bottom right of the dialog.

Other visible elements include the 'Location' (위치(L)) dropdown menu showing 'C:\Users\media\source\repos', a 'Back' (뒤로(B)) button, and a status message at the bottom: '"C:\Users\media\source\repos\Example\Example\"'에 프로젝트이(가) 만들어집니다.



Visual Studio Community Install (Mac)

❖ Visual Studio(Mac)



Visual Studio Community

Windows에서 제공되는 .NET 및 C++ 개발자를 위한 최고의 포괄적 IDE입니다. 소프트웨어 개발의 모든 단계를 향상시키고 개선할 수 있는 다양한 도구와 기능으로 가득 차 있습니다.

[자세한 정보 →](#)

무료 다운로드



Visual Studio for Mac

macOS에서 기본 제공되는 .NET 개발자를 위한 포괄적인 IDE입니다. 웹, 클라우드, 모바일 및 게임 개발을 위한 최고 수준의 지원을 포함합니다.

[자세한 정보 →](#)

권장인스톨러를 사용하여 대해 자세히 알아보세요.

무료 다운로드



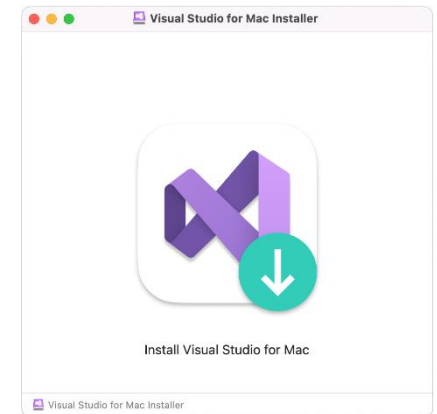
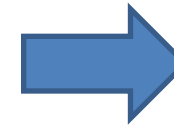
Visual Studio Code

Windows, macOS 및 Linux에서 실행되는 독립 실행형 소스 코드 편집기입니다. JavaScript 및 웹 개발자를 위한 최고의 선택이며, 거의 모든 프로그래밍 언어를 지원할 수 있는 확장 기능을 제공합니다.

[자세한 정보 →](#)

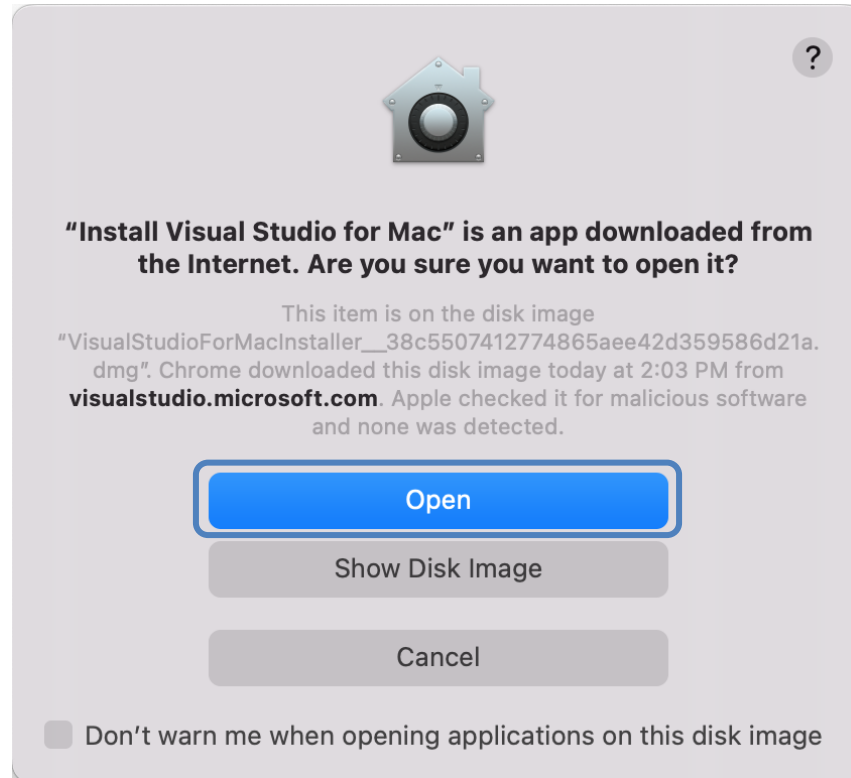
Visual Studio Code를 사용하면 자막
권장인스톨러를 사용하여 다운로드할 수 있습니다.

무료 다운로드

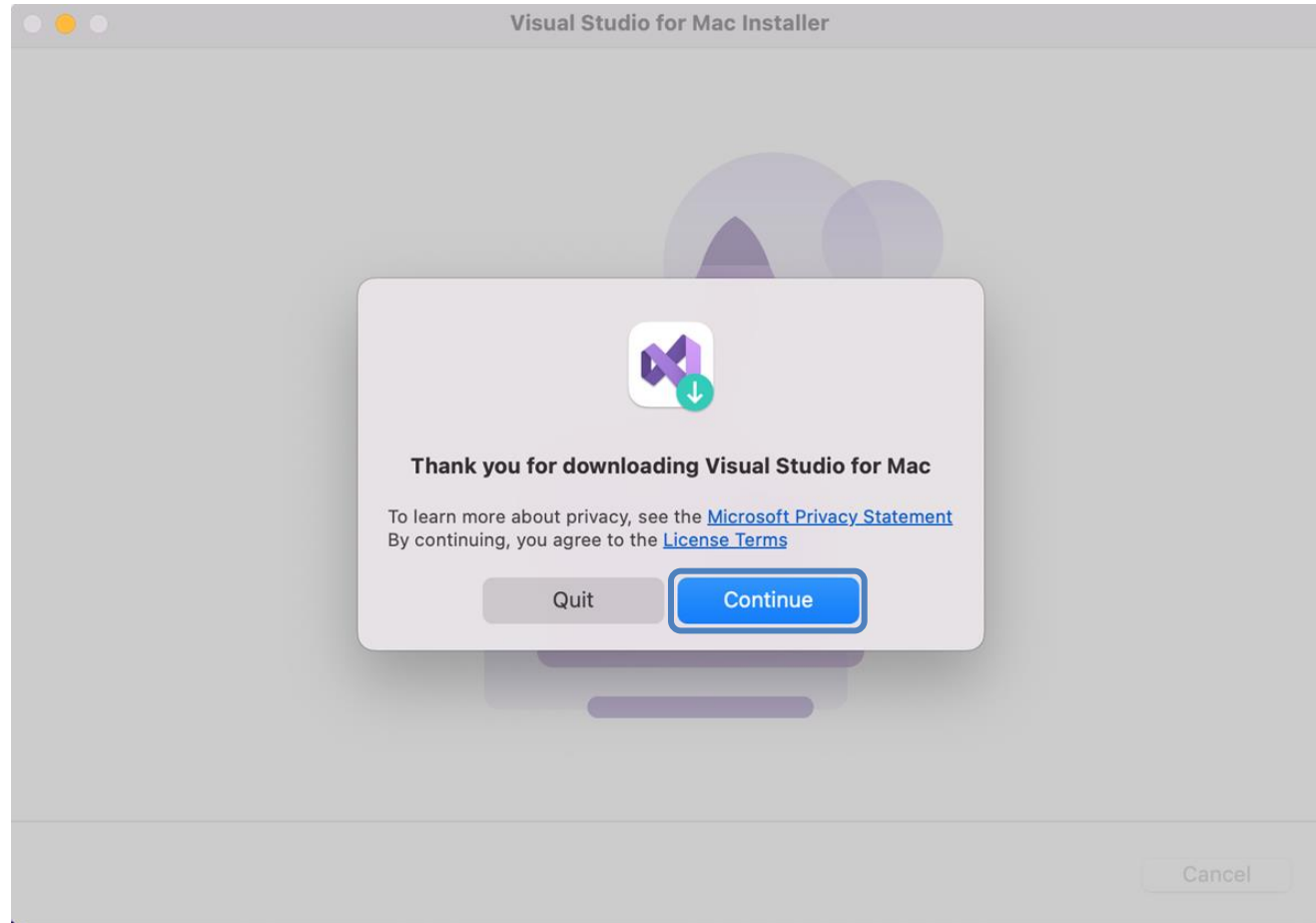


Microsoft 개발자 클라우드의
일부

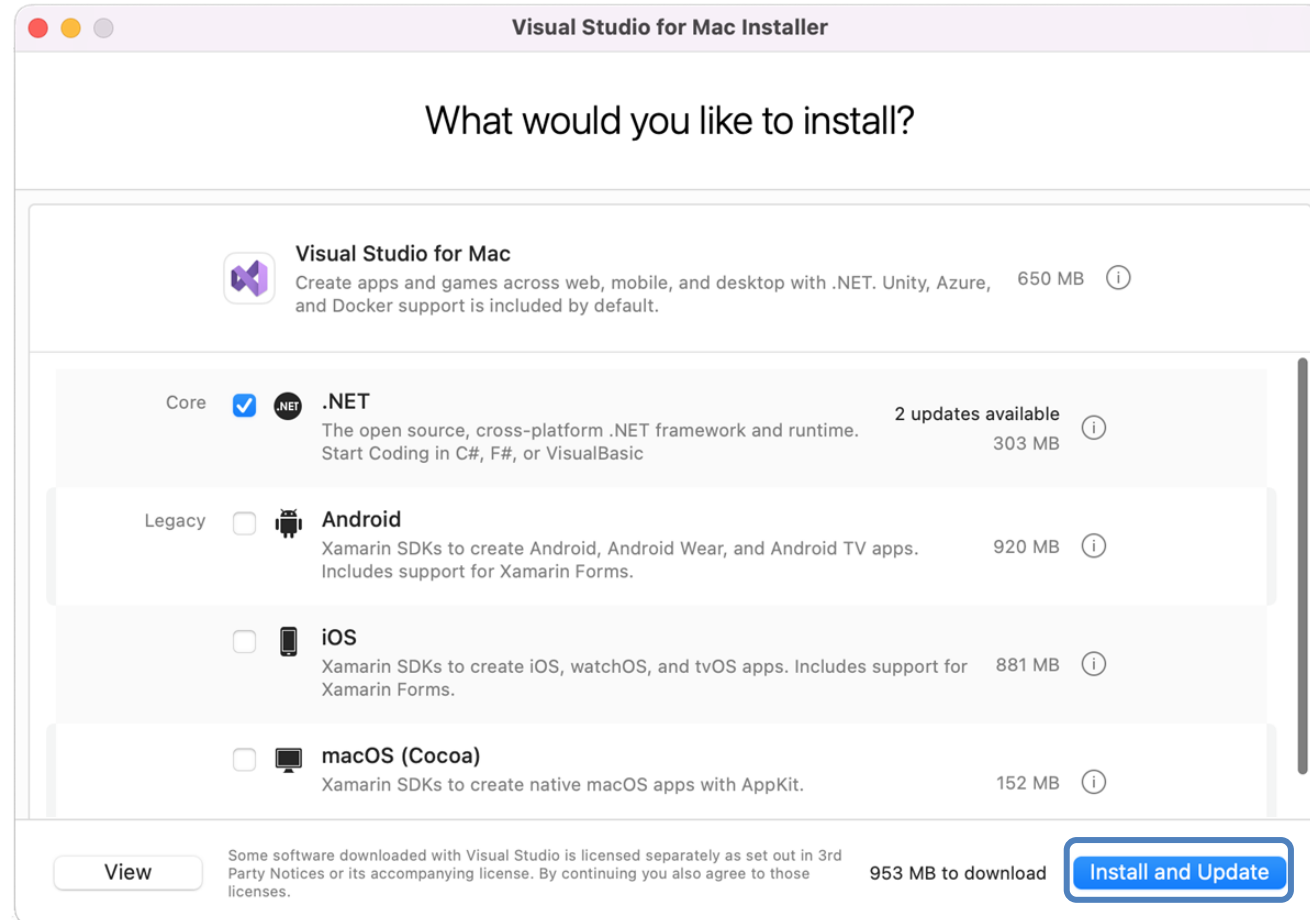
❖ Visual Studio(Mac)



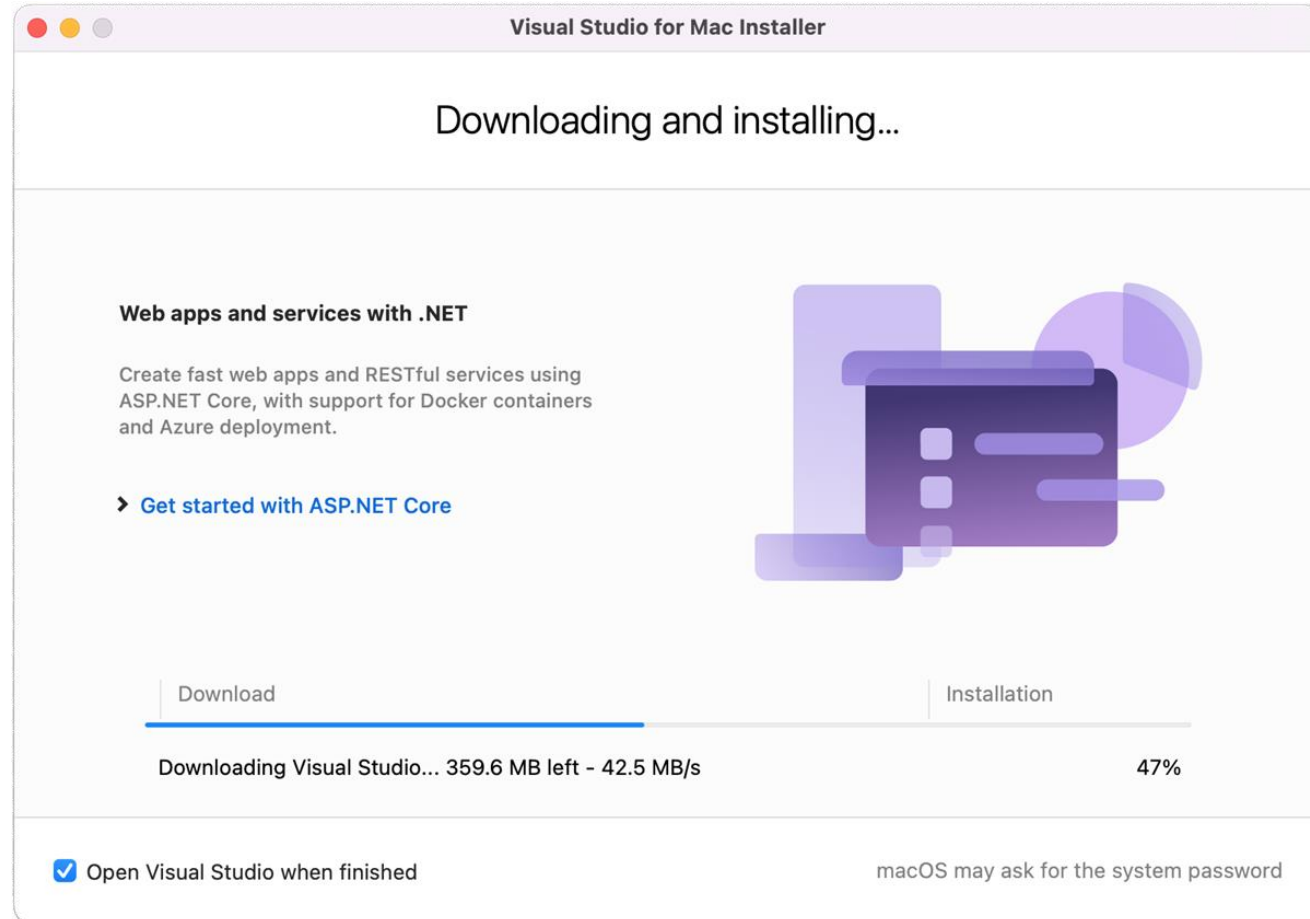
❖ Visual Studio(Mac)



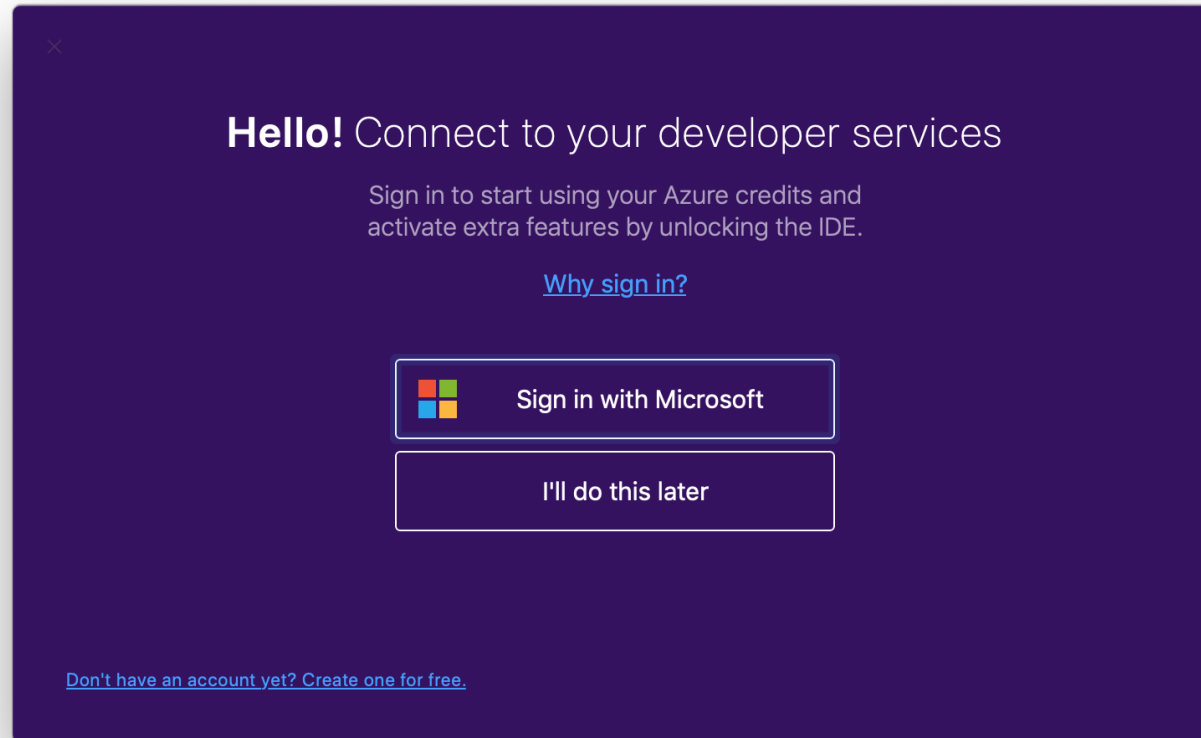
❖ Visual Studio(Mac)



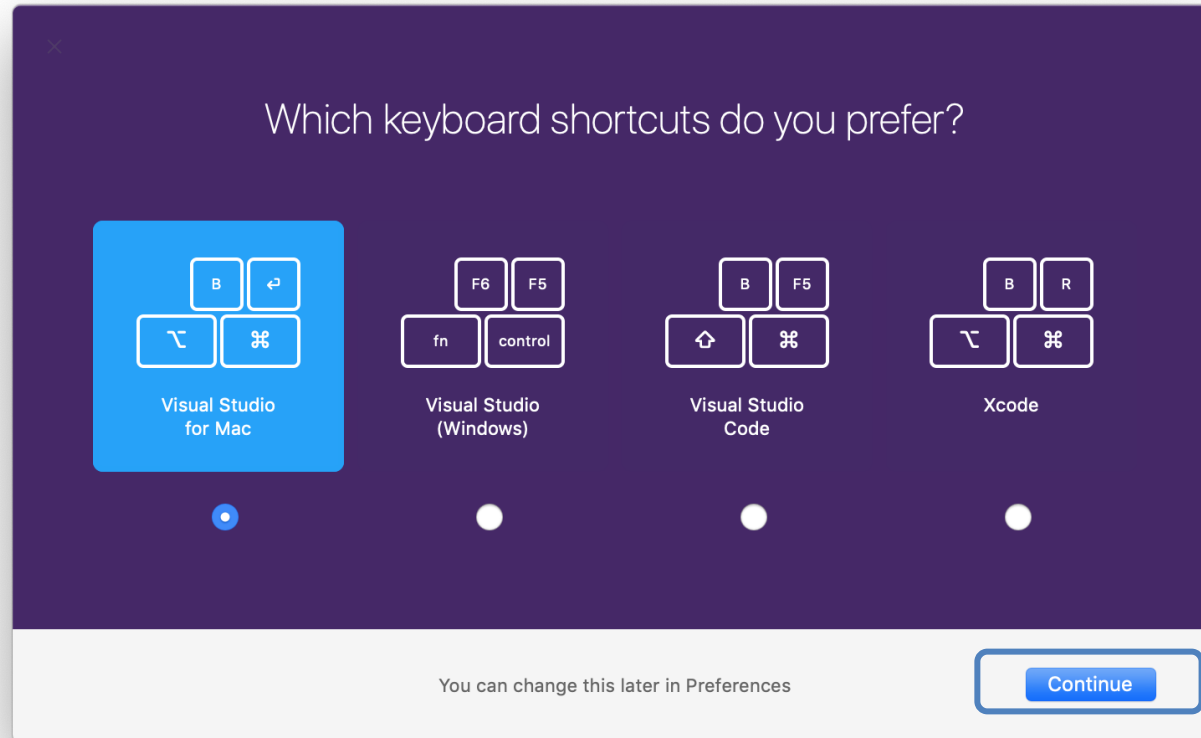
❖ Visual Studio(Mac)



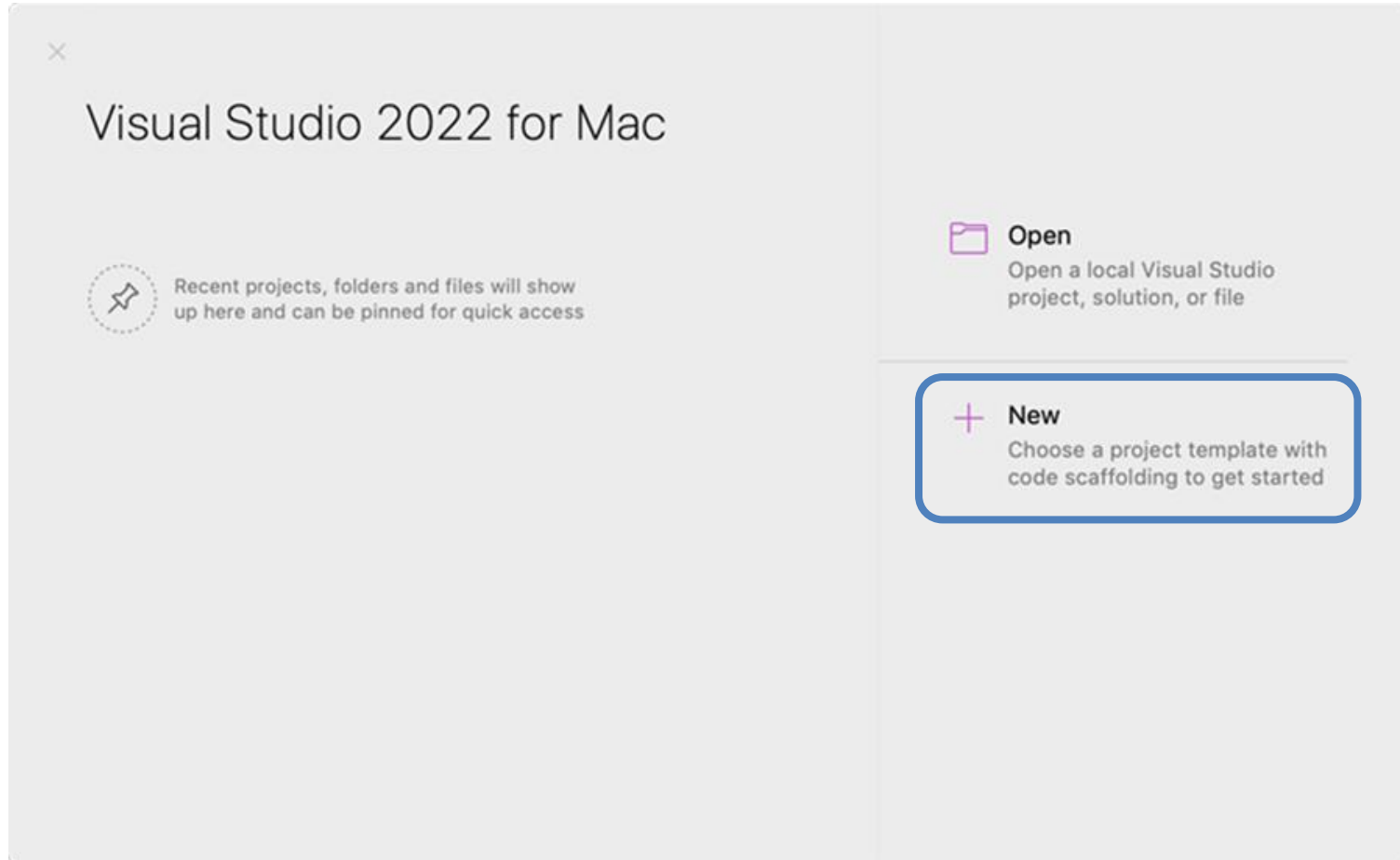
❖ Visual Studio(Mac)



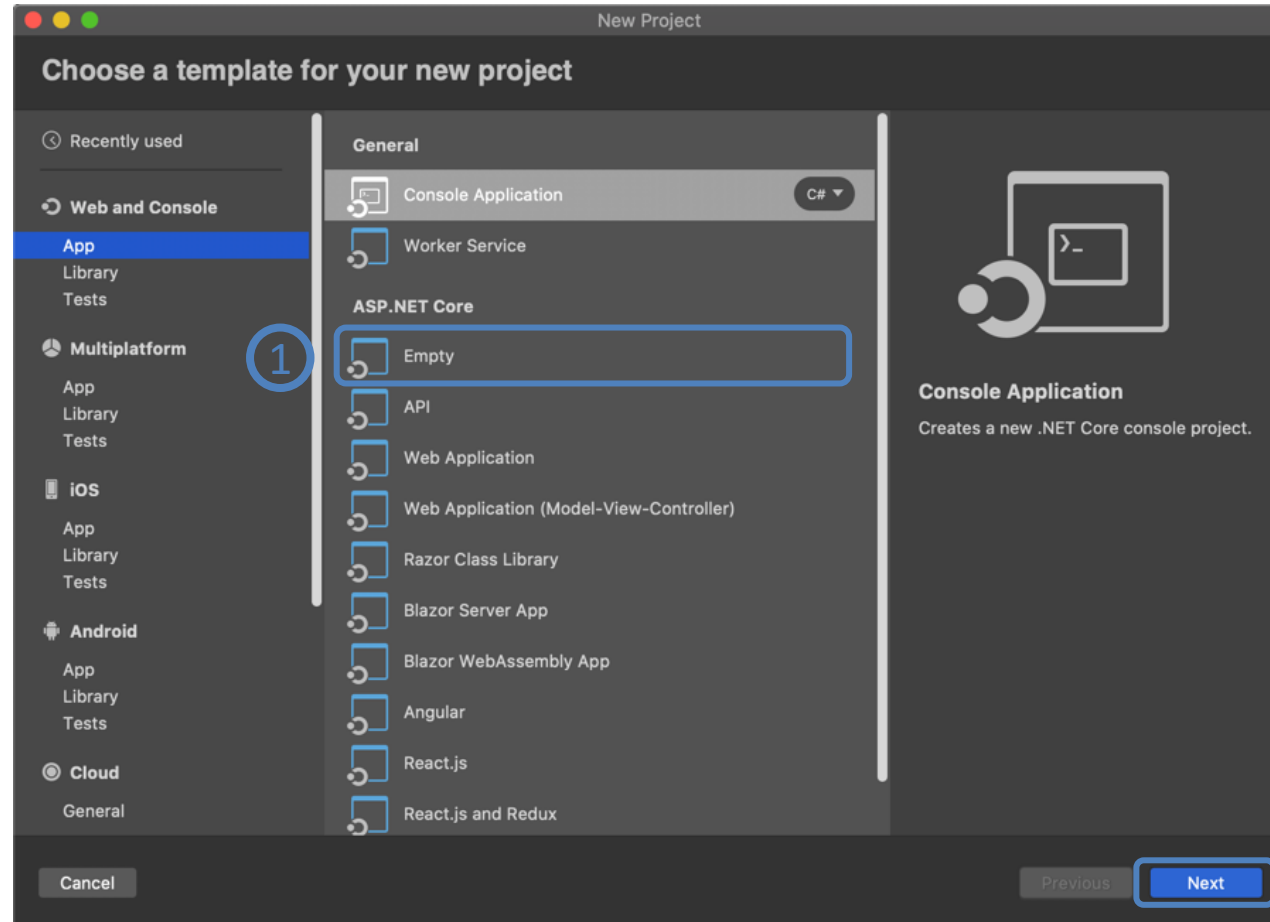
❖ Visual Studio(Mac)



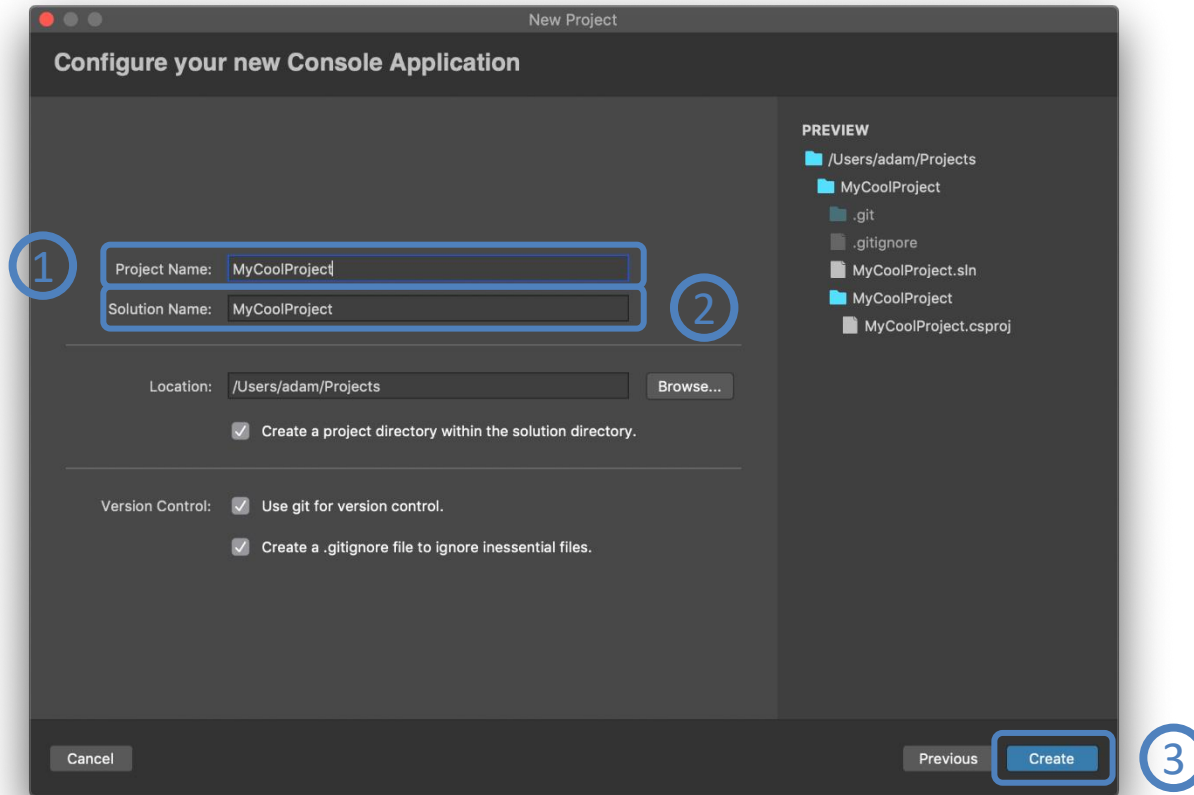
❖ Visual Studio(Mac)



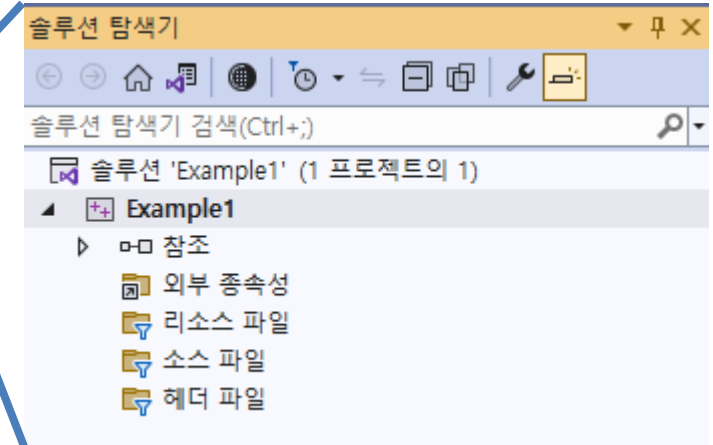
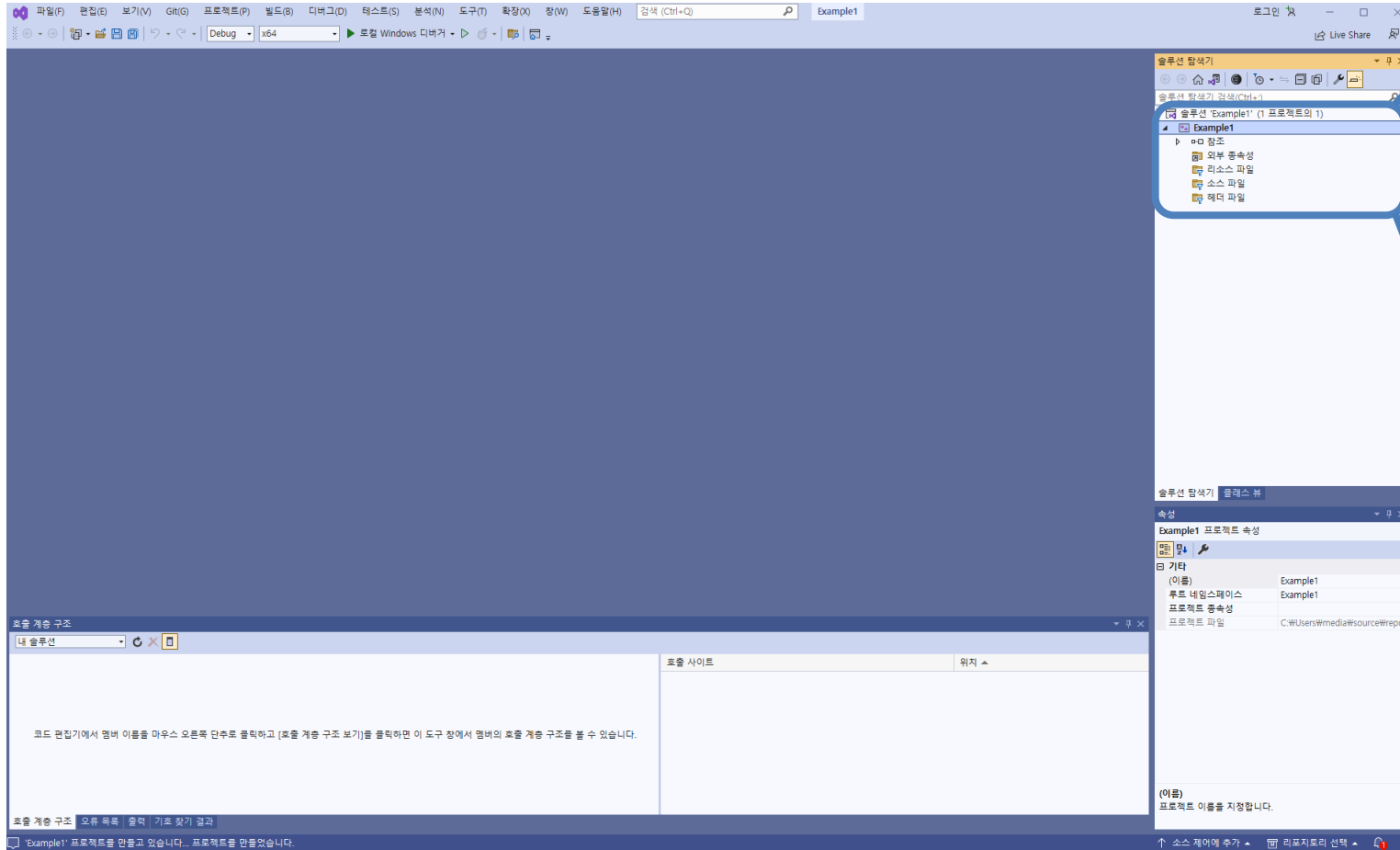
❖ Visual Studio(Mac)



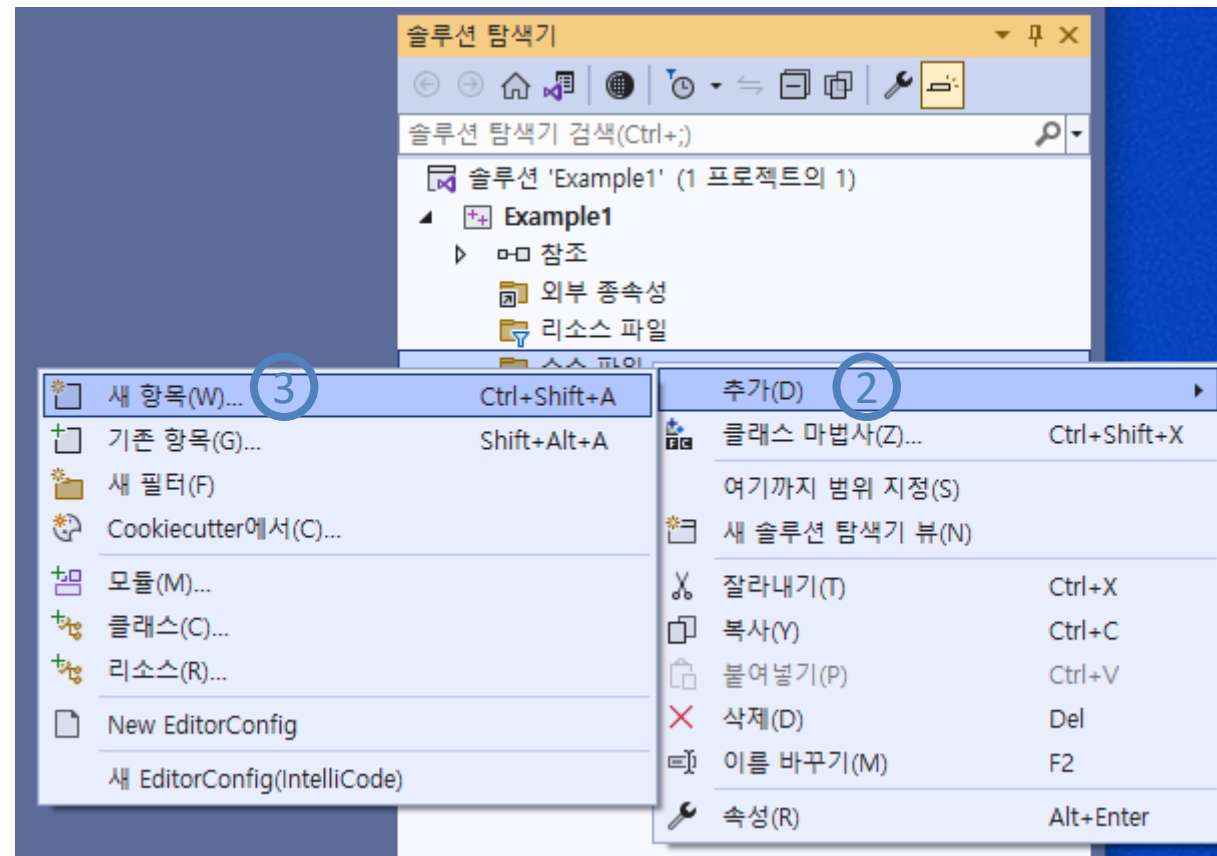
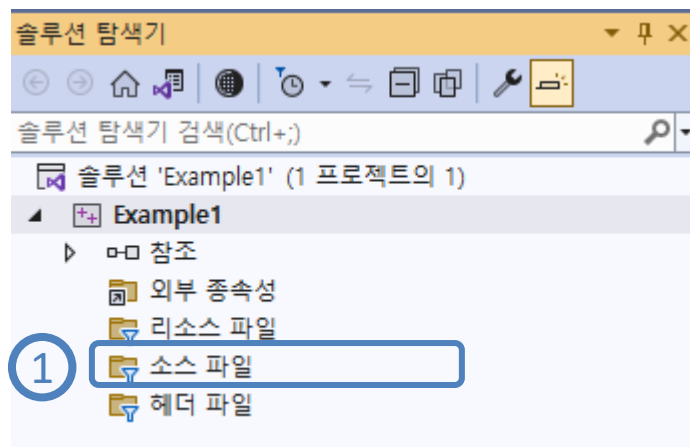
❖ Visual Studio(Mac)



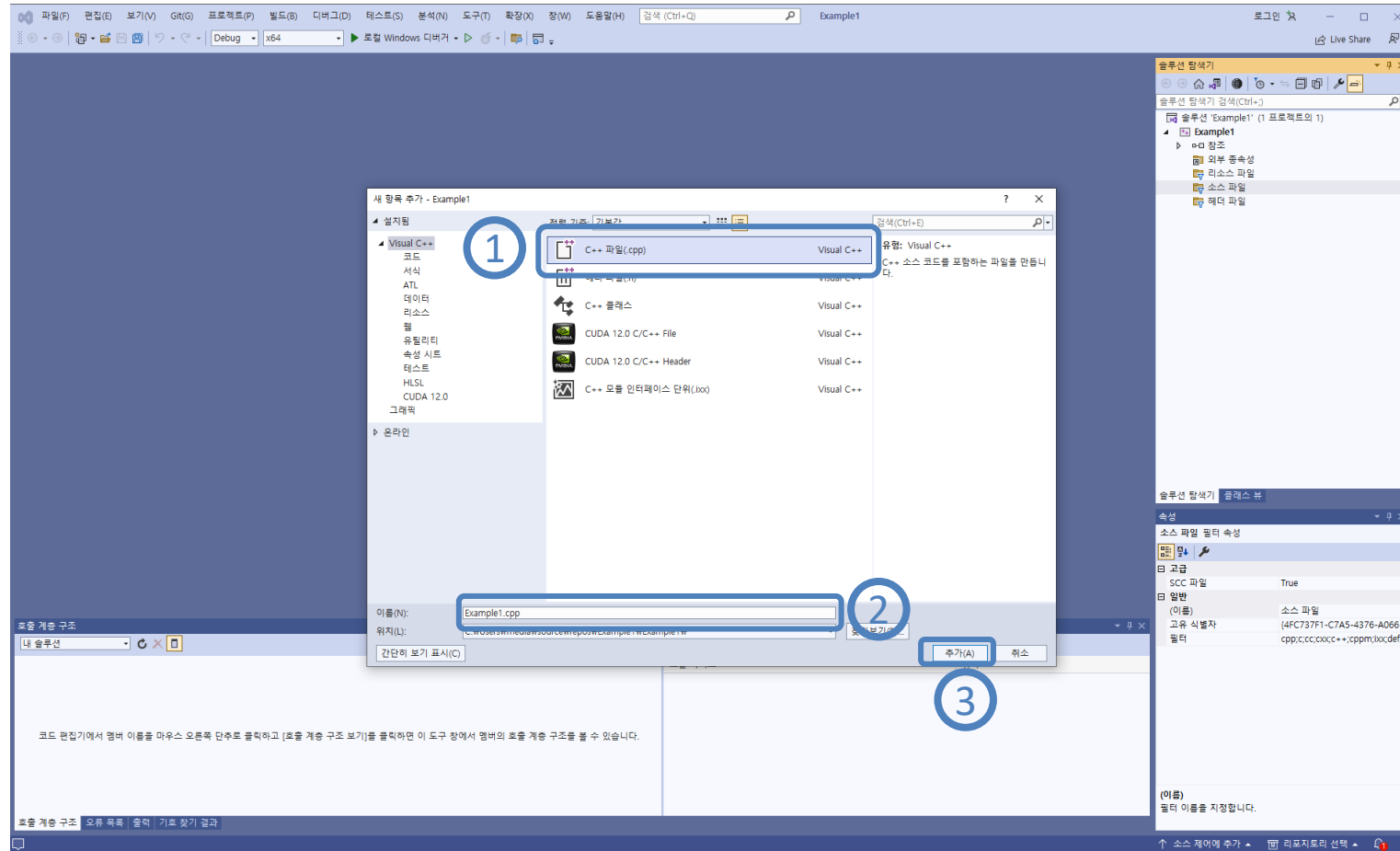
❖ Visual Studio



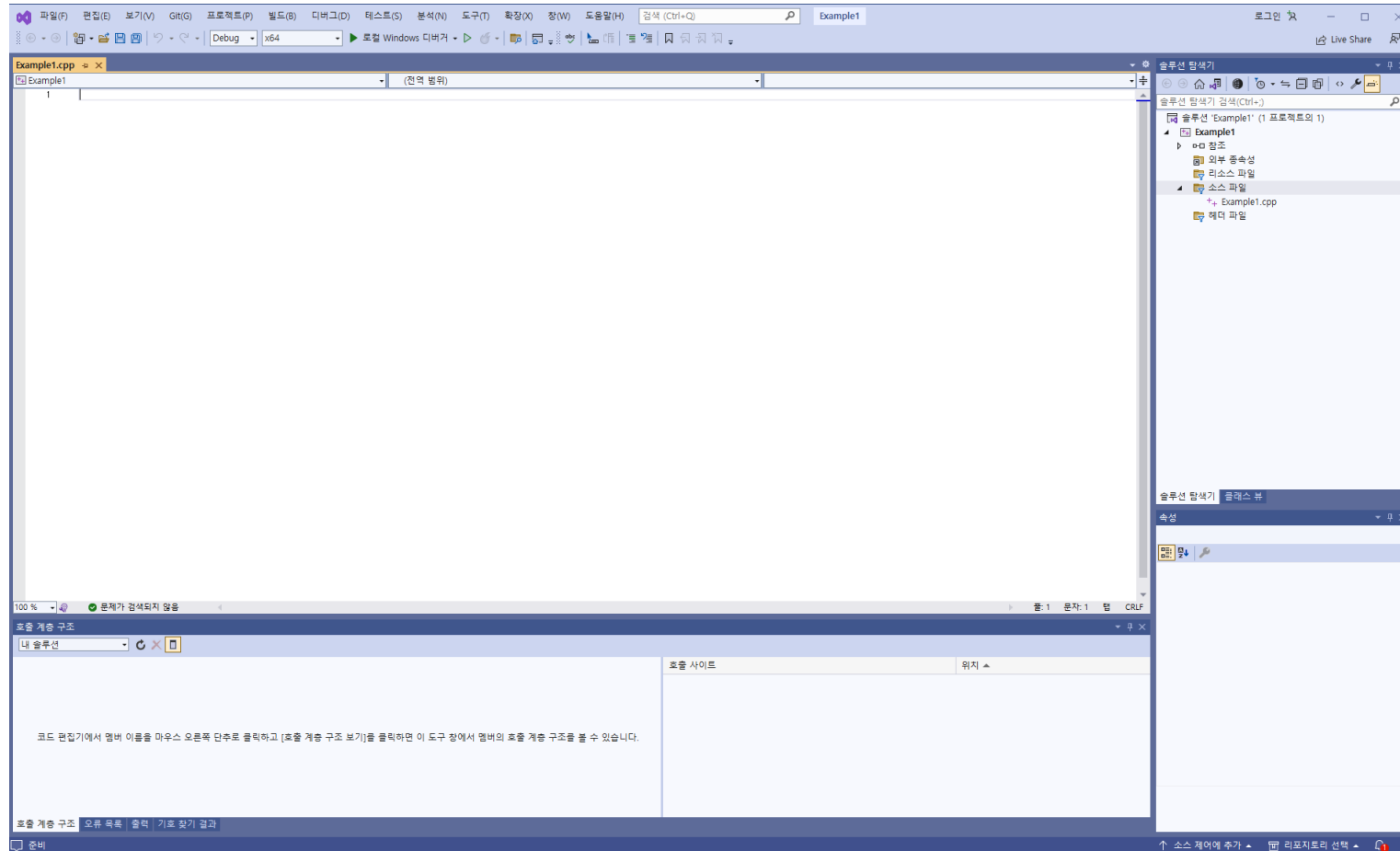
❖ Visual Studio



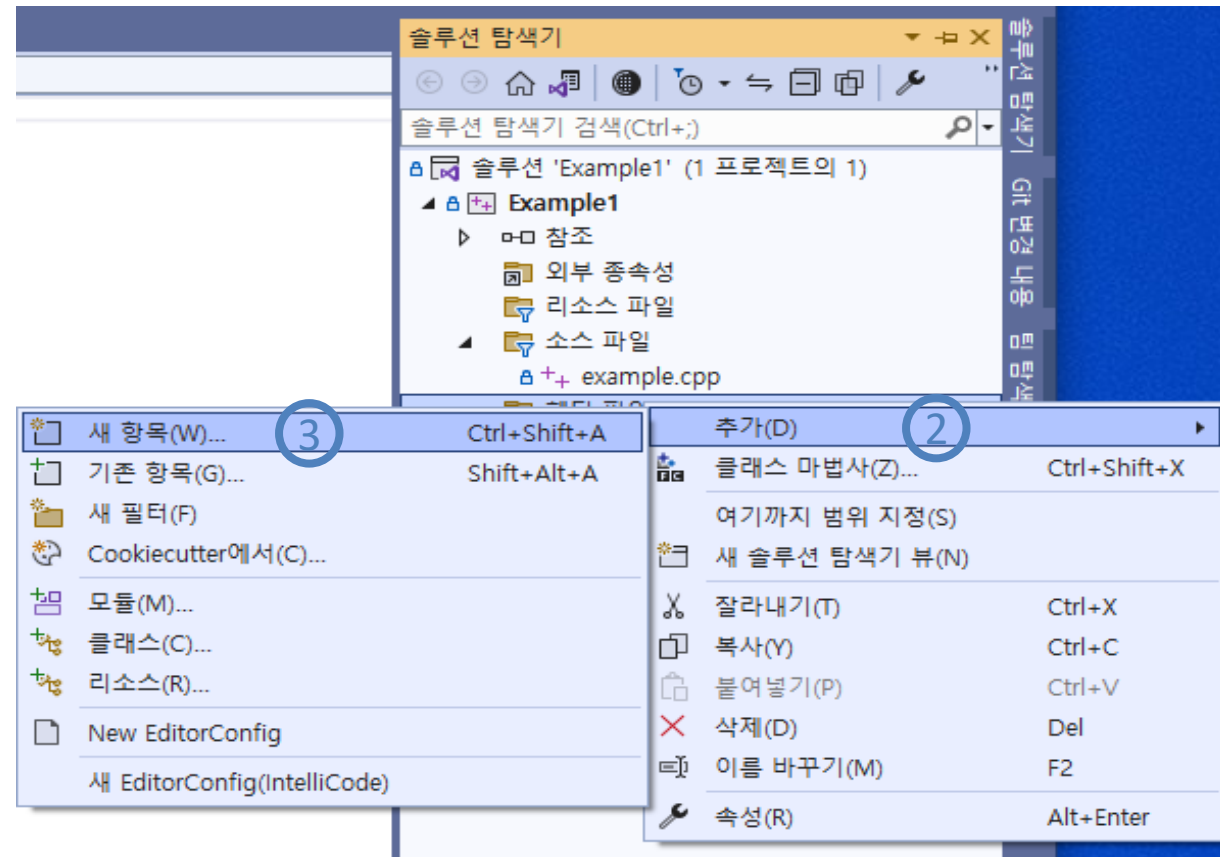
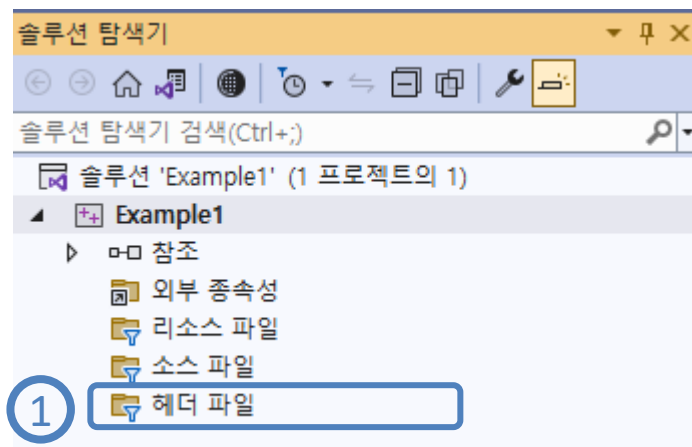
❖ Visual Studio



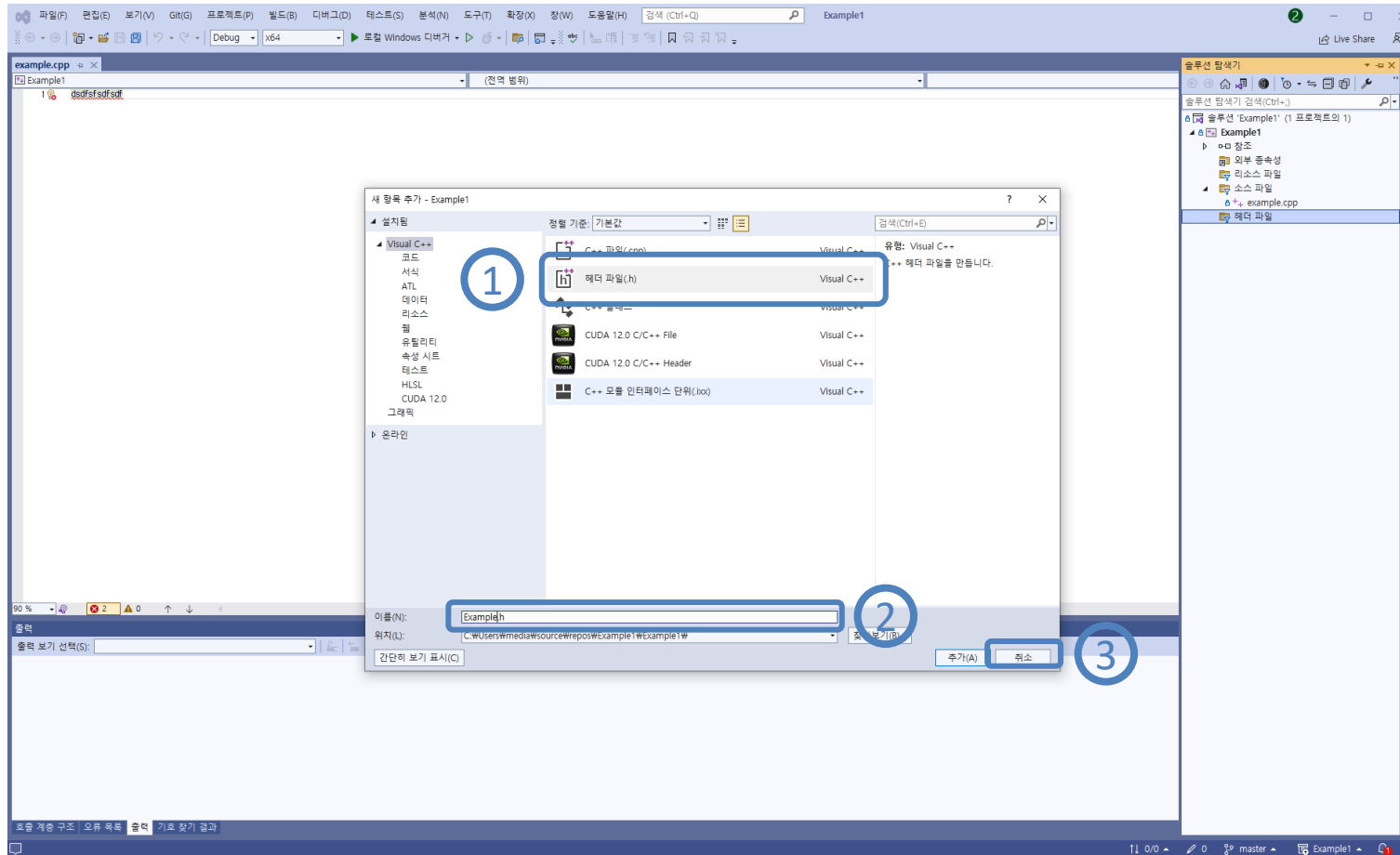
❖ Visual Studio



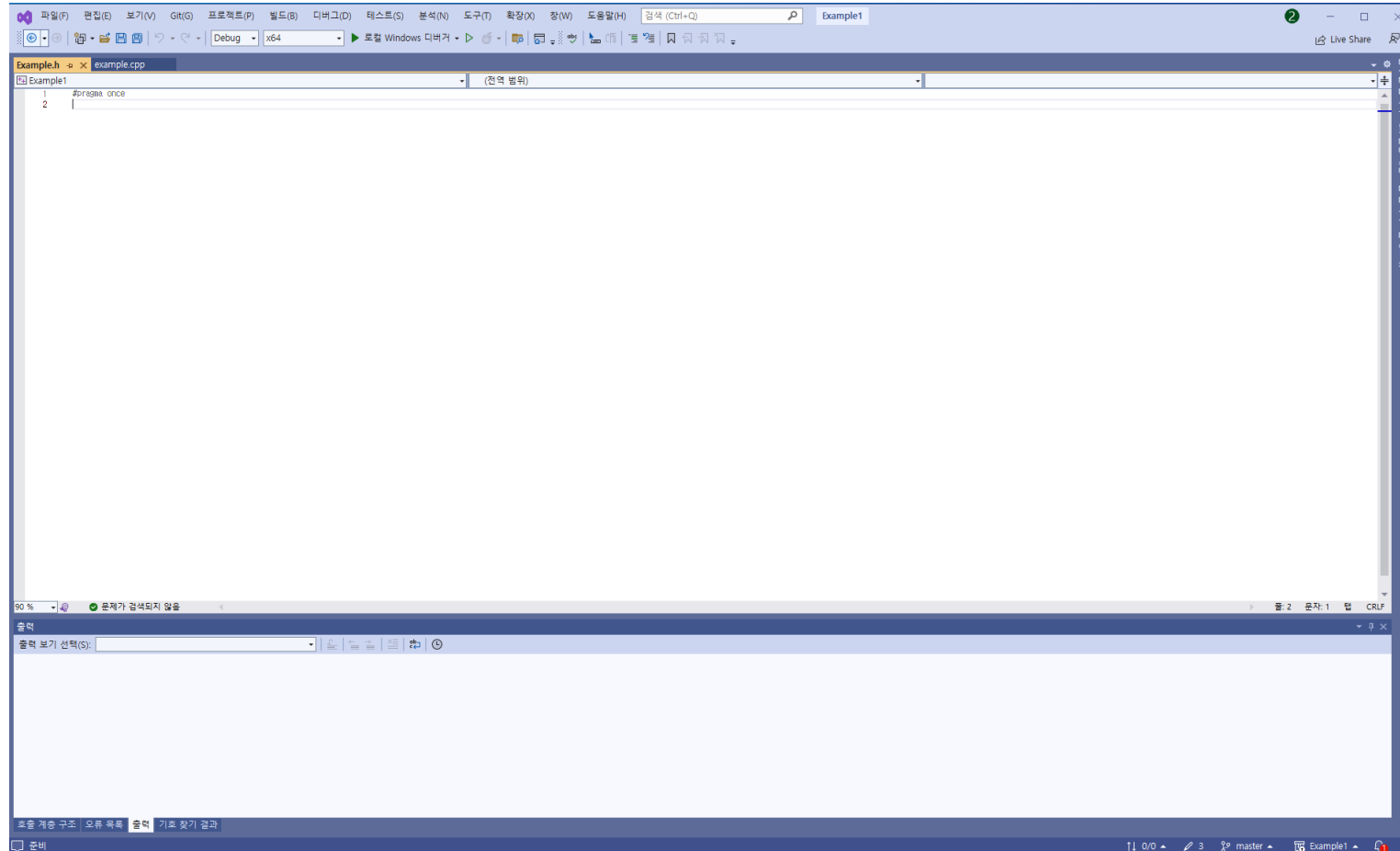
❖ Visual Studio



❖ Visual Studio



❖ Visual Studio





*Thank
you*

