```
from tkinter import *
import tkinter.font as font
from collections import defaultdict

window = Tk()

HumanT20 = font.Font(family='휴먼모음T', size=20)
HumanT12 = font.Font(family='휴먼모음T', size=12)
ClearGothic10 = font.Font(family='맑은 고딕', size=10)
NanumGothic12 = font.Font(family='NanumbarunGothic', size=11)

window.option_add( "*font", NanumGothic12)
window.title("전공진입요건 계산 프로그램")
window.geometry("910x720")

selected_계열 = StringVar()
selected_계열.set("계열 선택")
selected_계열_string = StringVar()
selected_계열_string.set("[계열을 입력하세요]")

selected_지원학과 = StringVar()
selected_지원학과.set("지원학과 선택")

explain_전공진입요건 = StringVar()
explain_전공진입요건.set("")
explain_전공진입요건_1 = "의사소통 4학점, 기본영어+전문영어 4학점, 창의와사유 2학점,\n
소프트웨어기초 2학점, 기초인문사회과학 6학점"
explain_전공진입요건_2 = "의사소통 4학점, 기본영어+전문영어 4학점, 창의와사유 2학점,\n
소프트웨어기초 2학점, 기초자연과학 12학점"
explain_전공진입요건_3 = "의사소통 4학점, 기본영어+전문영어 4학점,\n창의와사유 2학점,
기초자연과학 15학점, 창의적공학설계"

major_entrance_rule_HS = ['의사소통', '기본영어', '전문영어', '창의와사유', '소프트웨어기초'
, '기초인문사회과학']
major_entrance_rule_N = ['의사소통', '기본영어', '전문영어', '창의와사유', '소프트웨어기초',
 '기초자연과학']
major_entrance_rule_E = ['의사소통', '기본영어', '전문영어', '창의와사유', '기초자연과학', '
창의적공학설계']


explain_전공진입요건_dict = {
    "인문과학계열": explain_전공진입요건_1,
    "사회과학계열": explain_전공진입요건_1,
    "자연과학계열": explain_전공진입요건_2,
    "공학계열": explain_전공진입요건_3,
}

GPA_score = {'A+': 4.5, 'A': 4.0, 'B+': 3.5, 'B': 3.0, 'C+': 2.5, 'C': 2.0, 'D+': 1.5, 'D':
1.0, 'F': 0.0}

authorityHSN = [
    ['기본영어', '2', '영어쓰기'],
    ['소프트웨어기초', '2', '컴퓨팅사고와SW코딩'],
    ['일반선택', '1', 'FYE세미나I', 'P(P/F)'],
    ['기본영어', '2', '영어발표'],
    ['소프트웨어기초', '2', '문제해결과알고리즘'],
    ['일반선택', '1', 'FYE세미나II', 'P(P/F)']
]
```

```
authorityE = [
    ['기본영어', '2', '영어쓰기'],
    ['기초자연과학', '3', '공학컴퓨터프로그래밍'],
    ['일반선택', '1', 'FYE세미나I', 'P(P/F)'],
    ['기본영어', '2', '영어발표'],
    ['기초자연과학', '3', '프로그래밍기초와실습'],
    ['일반선택', '1', 'FYE세미나II', 'P(P/F)'],
    ['전공', '3', '창의적공학설계']
]

def change_계열(a):
    selected_계열_string.set("[{} 전공진입 요건]".format(a))
    explain_전공진입요건.set(explain_전공진입요건_dict[a])

    if a != '공학계열':
        for i in range(6):
            tmp_list = authorityHSN[i]
            if i >= 3:
                i += 9
            territory_string[i].set(tmp_list[0])
            credit_string[i].set(tmp_list[1])
            name_string[i].set(tmp_list[2])
            if tmp_list[0] == '일반선택':
                gpa_string[i].set(tmp_list[3])
    else:
        for i in range(6):
            tmp_list = authorityE[i]
            if i >= 3:
                i += 9
            territory_string[i].set(tmp_list[0])
            credit_string[i].set(tmp_list[1])
            name_string[i].set(tmp_list[2])
            if tmp_list[0] == '일반선택':
                gpa_string[i].set(tmp_list[3])

        territory_string[23].set(authorityE[6][0])
        credit_string[23].set(authorityE[6][1])
        name_string[23].set(authorityE[6][2])


title = Label(window, text="전공진입요건 계산 프로그램", font=HumanT20, height=2)
title.grid(row=0, column=0, columnspan=99)

계열 = Label(window, text=" 계열: ")
계열.grid(row=1, column=1)
계열option = OptionMenu(window, selected_계열, "인문과학계열", "사회과학계열", "자연과학계열
", "공학계열", command=change_계열)
계열option.grid(row=1, column=2, columnspan=2)
지원학과 = Label(window, text=" 지원학과: ")
지원학과.grid(row=1, column=4, columnspan=2)
지원학과option = OptionMenu(window, selected_지원학과, "소속대학 원전공", "글로벌융합학부")
지원학과option.grid(row=1, column=6)

전공진입요건_label = Label(window, textvariable=selected_계열_string, height=2)
전공진입요건_label.grid(row=2, column=0, columnspan=99)
전공진입요건_explainLabel = Label(window, textvariable=explain_전공진입요건, height=3)
전공진입요건_explainLabel.grid(row=3, column=0, columnspan=99)
```

```
영역_label = Label(window, text="영역")
학점_label = Label(window, text="학점")
과목명_label = Label(window, text="과목명")
평점_label = Label(window, text="평점")
영역_label2 = Label(window, text="영역")
학점_label2 = Label(window, text="학점")
과목명_label2 = Label(window, text="과목명")
평점_label2 = Label(window, text="평점")
영역_label.grid(row=4, column=0)
학점_label.grid(row=4, column=1)
과목명_label.grid(row=4, column=2)
평점_label.grid(row=4, column=3)
영역_label2.grid(row=4, column=4)
학점_label2.grid(row=4, column=5)
과목명_label2.grid(row=4, column=6)
평점_label2.grid(row=4, column=7)

territory = []
credit = []
name = []
gpa = []

territory_string = []
credit_string = []
name_string = []
gpa_string = []

for i in range(24):
    territory_string.append(StringVar())
    territory_string[i].set("영역 선택")
    credit_string.append(StringVar())
    credit_string[i].set("0")
    name_string.append(StringVar())
    gpa_string.append(StringVar())
    gpa_string[i].set("A+")

    territory.append(OptionMenu(window, territory_string[i], "인성", "리더십", "기본영어", "
전문영어", "글로벌문화",
                                "의사소통", "창의와사유", "소프트웨어기초", "
기초인문사회과학", "기초자연과학",
                                "인간/문화", "사회/역사", "자연/과학/기술", "일반선택", "
기타교양", "전공"))
    credit.append(Entry(window, textvariable=credit_string[i], width=3, justify='center'))
    name.append(Entry(window, textvariable=name_string[i]))
    gpa.append(OptionMenu(window, gpa_string[i], "A+", "A", "B+", "B", "C+", "C", "D+", "D",
 "F", "P(P/F)", "F(P/F)"))

    territory[i].config(width=12)
    gpa[i].config(width=4)

    if i <= 11:
        territory[i].grid(row=i+5, column=0)
        credit[i].grid(row=i+5, column=1)
        name[i].grid(row=i+5, column=2)
        gpa[i].grid(row=i+5, column=3)
    else:
        territory[i].grid(row=i-7, column=4)
```

```
        credit[i].grid(row=i-7, column=5)
        name[i].grid(row=i-7, column=6)
        gpa[i].grid(row=i-7, column=7)

error_message = StringVar()


def calculate():
    global successful
    successful = True
    major = selected_계열.get()
    want_to_in = selected_지원학과.get()

    if major == '계열 선택':
        error_message.set("계열을 선택해주세요.")
        successful = False
        return False
    elif major in ['인문과학계열', '사회과학계열']:
        rule = major_entrance_rule_HS
        available_territory = {'의사소통': 4, '기본영어+전문영어': 4, '창의와사유': 2, '
소프트웨어기초': 2, '기초인문사회과학': 6}
    elif major == '자연과학계열':
        rule = major_entrance_rule_N
        available_territory = {'의사소통': 4, '기본영어+전문영어': 4, '창의와사유': 2, '
소프트웨어기초': 2, '기초자연과학': 12}
    elif major == '공학계열':
        rule = major_entrance_rule_E
        available_territory = {'의사소통': 4, '기본영어+전문영어': 4, '창의와사유': 2, '
기초자연과학': 15, '창의적공학설계': 3}
    else:
        error_message.set("오류가 발생했습니다.")
        successful = False
        return False

    if want_to_in == '지원학과 선택':
        error_message.set("지원하고자 하는 학과를 선택해주세요.")
        successful = False
        return False
    elif want_to_in in ['소속대학 원전공', '글로벌융합학부']:
        pass
    else:
        error_message.set("지원학과 관련 오류가 발생했습니다.")
        successful = False
        return False

    less_complete = dict()
    personal_credit = calculate_credit()
    if personal_credit == False:
        error_message.set("요건 충족 여부 판단 중 오류가 발생했습니다.\n학점에 숫자를
입력했는지 확인하세요.")
        successful = False
        return False
    for a in available_territory:
        if available_territory[a] > personal_credit[a]:
            less_complete[a] = available_territory[a] - personal_credit[a]

    personal_GPA = calculate_GPA(major, want_to_in, rule)
    if personal_GPA == False:
```

```
            error_message.set("평점 계산 중 오류가 발생했습니다.")
            successful = False
            return False

        if successful:
            if not less_complete:
                result(want_to_in, personal_GPA)
            else:
                result(want_to_in, personal_GPA, complete=False, less=less_complete)

    def calculate_credit():
        global successful
        personal_credit = defaultdict(int)
        for i in range(24):
            tmp_t = territory_string[i].get()
            try:
                tmp_c = int(credit_string[i].get())
            except ValueError:
                successful = False
                return False

            if tmp_t == '영역 선택':
                pass
            elif tmp_t in ['기본영어', '전문영어']:
                personal_credit['기본영어+전문영어'] += tmp_c
            elif tmp_t == '전공':
                tmp_n = name_string[i].get()
                if tmp_n == '창의적공학설계':
                    tmp_t = '창의적공학설계'
                personal_credit[tmp_t] += tmp_c
            else:
                personal_credit[tmp_t] += tmp_c

        return personal_credit


    def calculate_GPA(major, want_to_in, rule):
        global successful
        personal_credit = 0
        total_gpa = 0
        if want_to_in == '소속대학 원전공':
            for i in range(24):
                tmp_t = territory_string[i].get()
                if (major == '공학계열') and (tmp_t == '전공'):
                    creative_or_not = name_string[i].get()
                    if creative_or_not == '창의적공학설계':
                        tmp_t = '창의적공학설계'

                try:
                    tmp_c = int(credit_string[i].get())
                except ValueError:
                    successful = False
                    return False

                tmp_g = gpa_string[i].get()

                if tmp_t in rule:
                    if tmp_g not in ['P(P/F)', 'F(P/F)']:
```

```
                    personal_credit += tmp_c
                    total_gpa += tmp_c * GPA_score[tmp_g]

        if personal_credit == 0:
            successful = False
            return False

        GPA_result = total_gpa / personal_credit
        return GPA_result

    elif want_to_in == '글로벌융합학부':
        for i in range(24):
            try:
                tmp_c = int(credit_string[i].get())
            except ValueError:
                successful = False
                return False

            tmp_g = gpa_string[i].get()

            if tmp_g not in ['P(P/F)', 'F(P/F)']:
                personal_credit += tmp_c
                total_gpa += tmp_c * GPA_score[tmp_g]

        GPA_result = total_gpa / personal_credit
        return GPA_result

    elif want_to_in == '지원학과 선택':
        successful = False
        return False
    else:
        successful = False
        return False


def result(want_to_in, gpa_result, complete=True, less=None):
    result_window = Toplevel(window)
    result_window.title('전공진입요건 계산 결과')

    if complete:
        final = Label(result_window, text="축하드립니다!", font=('NanumbarunGothic', 25), fg
="green3", width=20, height=2)
        final2 = Label(result_window, text="전공진입요건을 충족하셨습니다!", font=('
NanumbarunGothic', 15))
        final.grid(row=0, column=0, rowspan=2, columnspan=2)
        final2.grid(row=2, column=0, columnspan=2, sticky='n')

        final_gpa = Label(result_window, text="{} 진입시 반영되는".format(want_to_in), font=
('NanumbarunGothic', 18), anchor='s', height=2)
        final_gpa2 = Label(result_window, text="평점의 평균은 {}입니다!".format(gpa_result),
 font=('NanumbarunGothic', 18))
        final_gpa.grid(row=3, column=0, columnspan=2)
        final_gpa2.grid(row=4, column=0, columnspan=2)
    else:
        final = Label(result_window, text="충족하지 못했습니다...", font=('NanumbarunGothic'
, 25), fg="red3", width=20, height=2)
        final2 = Label(result_window, text="전공진입 영역별로 아직 채우지 못한 학점입니다.",
 anchor='n')
```

```
        final.grid(row=0, column=0, rowspan=2, columnspan=2)
        final2.grid(row=2, column=0, columnspan=2, sticky='n')

        less_frames = []
        for terri, cre in less.items():
            tmp_frame = Frame(result_window)
            terri_label = Label(tmp_frame, text=terri, width=15, height=2, anchor='s', font=
('NanumbarunGothic', 14))
            cre_label = Label(tmp_frame, text="{}학점".format(cre), width=15, height=2,
anchor='n', font=('NanumbarunGothic', 14))
            terri_label.pack(fill=BOTH)
            cre_label.pack(fill=BOTH)
            less_frames.append(tmp_frame)

        row_count = 2
        for i, frame in enumerate(less_frames):
            if i % 2 == 0:
                row_count += 1
                now_column = 0
            else:
                now_column = 1
            frame.grid(row=row_count, column=now_column, sticky=W+E)

        final_gpa = Label(result_window, text="{} 진입시 반영되는".format(want_to_in), font=
('NanumbarunGothic', 18), anchor='s', height=2)
        final_gpa2 = Label(result_window, text="평점의 평균은 현재까지 {}입니다.".format(
gpa_result), font=('NanumbarunGothic', 18))
        final_gpa.grid(row=98, column=0, columnspan=2)
        final_gpa2.grid(row=99, column=0, columnspan=2)
    result_window.mainloop()




Calculate_button = Button(window, text='전공진입요건 충족여부 계산하기!', command=calculate)
Calculate_button.grid(row=17, column=0, columnspan=99)

Calculate_error = Label(window, textvariable=error_message)
Calculate_error.grid(row=18, column=0, columnspan=99)


window.mainloop()
```