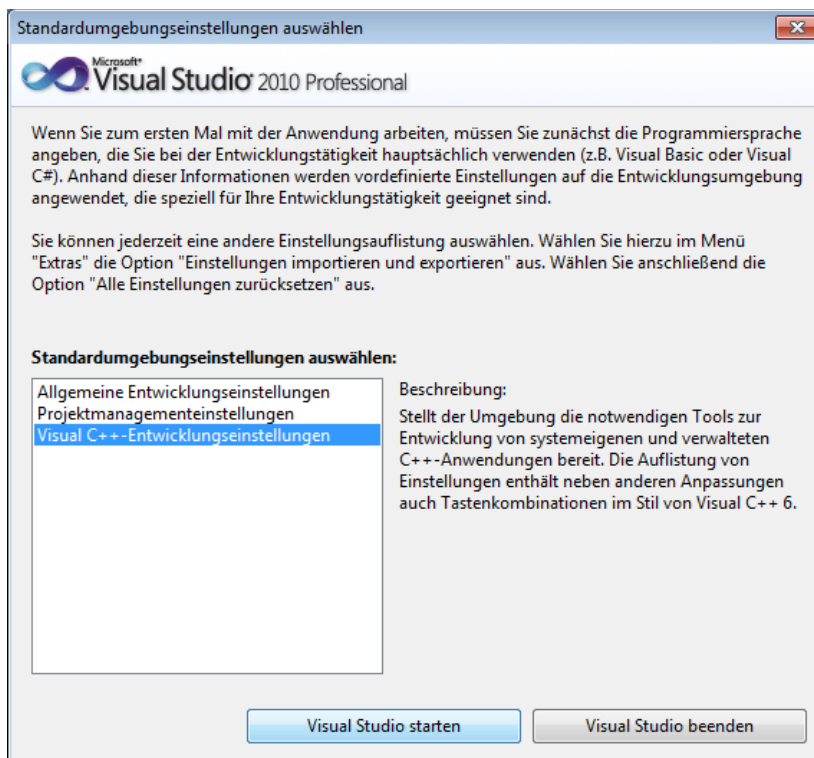


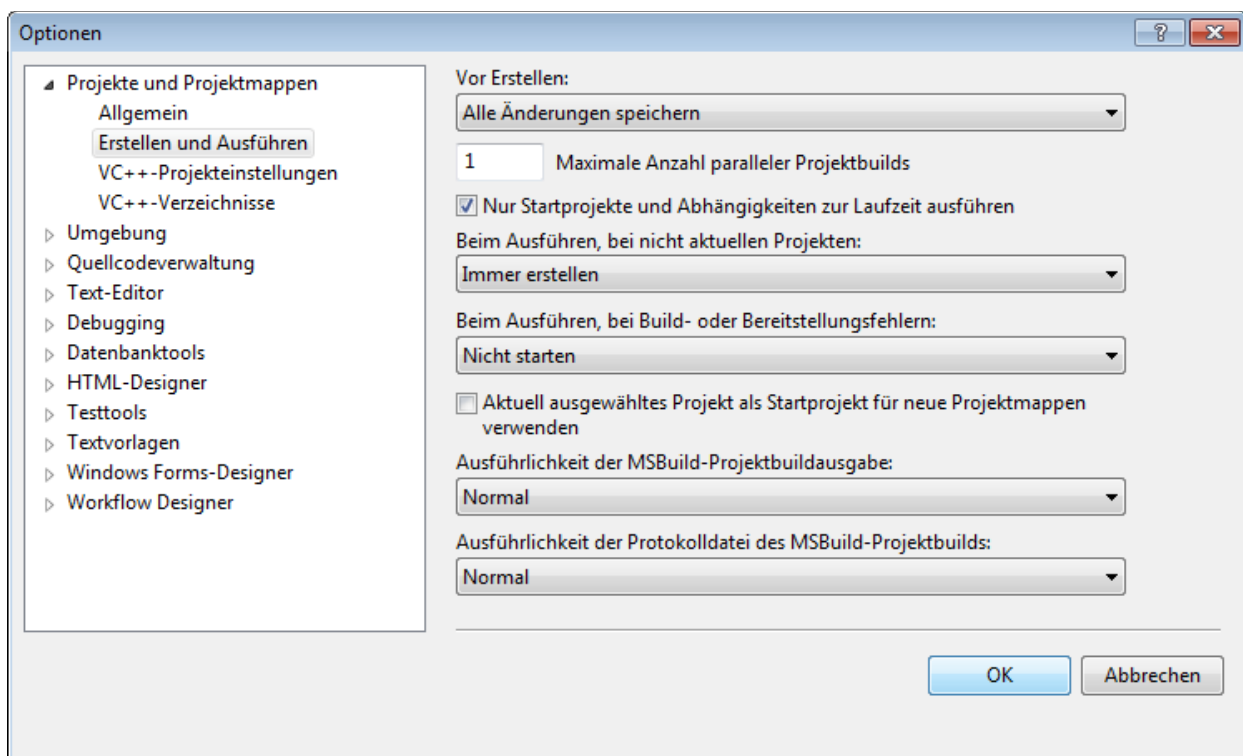
Was muss ich beim ersten Start von Visual Studio 2010 beachten?

Wenn Du Visual Studio 2010 das erste Mal startest, sollte folgender Dialog erscheinen:



(Sollte dieser Dialog nicht erscheinen, befolge die Anweisungen im zweiten Absatz des Screenshots.) Wähle wie im Screenshot gezeigt „Visual C++-Entwicklungseinstellungen“.

Wähle danach Extras / Optionen... / Projekte und Projektmappen / Erstellen und Ausführen. Übernimm die gezeigten Einstellungen in den Dropdown-Menüs rechts:



Wie erstelle ich ein neues Projekt in einer neuen Projektmappe?


Datei / Neu / Projekt...

Installierte Vorlagen: Visual C++ / Allgemein / Leeres Projekt

Name: Hello

Projektmappenname: Woche3

OK

Projektmappenexplorer: Projektmappe "Woche3" / Hello / Quelldateien  / Hinzufügen / Neues Element...

Installierte Vorlagen: Visual C++ / Code / C++-Datei (.cpp)

Name: hello.c

OK

```
#include <stdio.h>
#include <stdlib.h>
```

```
int main()
{
    printf("Goodbye Karel,\nHello C!\n");
    system("pause");
}
```

Projektmappenexplorer: Projektmappe "Woche3" / Hello  / Eigenschaften

Konfigurationseigenschaften / C/C++ / Allgemein / Warnstufe: Level 4 (/W4)

Warnungen als Fehler behandeln: Ja (/WX)

Starte das Projekt probenhalber via F5 oder über das Menü Debuggen / Debugging starten.

Wie füge ich ein weiteres Projekt zu einer bestehenden Projektmappe hinzu?

Projektmappenexplorer: Projektmappe "Woche3"  / Hinzufügen / Neues Projekt...

Installierte Vorlagen: Visual C++ / Allgemein / Leeres Projekt

Name: Summen OK

(Die Einstellungen /W4 und /WX müssen für jedes Projekt erneut vorgenommen werden!)

Projektmappenexplorer: Projektmappe "Woche3" / Summen  / Als Startprojekt festlegen

warning C4996: 'scanf': This function or variable may be unsafe.

In Visual Studio gibt es eine (angeblich) sicherere Alternative namens `scanf_s`. Dabei handelt es sich um eine Microsoft-spezifische, nicht-standardkonforme Funktion. Ich werde daher im Rahmen dieser Veranstaltung nicht näher auf `scanf_s` eingehen.

Um diese Warnung zu deaktivieren, füge folgende Zeile unterhalb der includes ein:

```
#pragma warning(disable: 4996)    // Yes, we want to use scanf!
```

Weitere Warnungen solltest Du nur in Absprache mit Deinem Betreuer deaktivieren!

error C2143: Syntaxfehler: Es fehlt ';' vor 'Typ'

Hierbei handelt es sich um eine der verwirrendsten Fehlermeldungen von Visual Studio. Wahrscheinlich hast Du versucht, in der Mitte eines Blocks eine Variable zu definieren:

```
{
    int a;                // Variablendefinition :)
    printf("a? ");        // keine Variablendefinition
    scanf("%d", &a);

    int b;                // zu spät :(
    printf("b? ");
    scanf("%d", &b);
}
```

Variablen müssen am Anfang eines Blocks definiert werden. Nach der ersten Anweisung, die keine Variablendefinition ist, sind Variablendefinitionen in C89 nicht mehr zulässig.

warning C4013: '*Funktionsname*' undefiniert; Annahme: extern mit Rückgabotyp int

Wahrscheinlich hast Du Dich beim Schreiben eines Funktionsnamens vertippt. Beispiel:

```
double answer(double a, double b)
{
    return ((a+b)*(a+b)-(a*a)-(b*b)) / (2*a*b) + 41.0;
}

int main()
{
    double x = anwser(1, 2);
}
```

error C2371: 'Funktionsname': Neudefinition; unterschiedliche Basistypen

Wahrscheinlich hast Du eine Funktion oberhalb ihrer Definition aufgerufen. Beispiel:

```
int main()
{
    double x = answer(1, 2);           // Aufruf
}

double answer(double a, double b)    // Definition
{
    return ((a+b)*(a+b)-(a*a)-(b*b))/(2*a*b) + 41.0;
}
```

In den meisten modernen Programmiersprachen wie Java oder C# spielt die Reihenfolge der Funktionsdefinitionen keine Rolle. In C kann man eine Funktion dagegen leider nicht so ohne weiteres oberhalb ihrer Definition aufrufen. Falls Du die Definition nicht über ihren ersten Aufruf verschieben willst, deklariere die Funktion vor ihrem ersten Aufruf:

```
double answer(double, double);       // Deklaration

int main()
{
    double x = answer(1, 2);           // Aufruf
}

double answer(double a, double b)    // Definition
{
    return ((a+b)*(a+b)-(a*a)-(b*b))/(2*a*b) + 41.0;
}
```

warning C4715: Nicht alle Steuerelementpfade geben einen Wert zurück.

Wenn eine Funktion einen anderen Ergebnistyp als `void` hat, dann muss sichergestellt sein, dass jeder Pfad durch die Funktion mit einer passenden `return`-Anweisung endet:

```
int vorzeichen(int x)
{
    if (x < 0) return -1;
}
```

In diesem Beispiel würde für `x >= 0` keine `return`-Anweisung ausgeführt werden, womit die Funktion ihr Versprechen, einen `int` zu liefern, nicht einhalten würde. Stattdessen würde ein mehr oder weniger „zufälliger“ Wert als Ergebnis geliefert werden. Diese Warnung ist in den meisten Fällen ein sicheres Indiz für einen Programmierfehler.

Man beachte, dass Visual Studio keine Beziehungen zwischen den Bedingungen versteht:

```
int vorzeichen(int x)
{
    if (x < 0) return -1;
    else if (x >= 0) return +1;
}
```

Ein Mensch sieht sofort, dass genau eine dieser beiden Bedingungen auf jeden Fall zutrifft. Visual Studio sieht dagegen nur folgende Struktur:

```
Ergebnistyp Funktionsname(Parametertyp Parametername)
{
    if (Bedingung1) return Ergebnis1;
    else if (Bedingung2) return Ergebnis2;
}
```

und kommt folgerichtig zu dem Schluss: „Wenn beiden Bedingungen falsch sind, wird keine `return`-Anweisung ausgeführt, also muss ich eine Warnung generieren!“

Trotzdem ist diese Warnung hilfreich, denn die zweite Bedingung ist redundant:

```
int vorzeichen(int x)
{
    if (x < 0) return -1;
    else return +1;
}
```

Wir brauchen nicht zu prüfen, ob `(x >= 0)` gilt, da wir bereits wissen, dass `(x < 0)` nicht gilt!