CFG LOOP.C CON COMPILAZIONE -O2

CFG FUNZIONE LOOP

```
; Function Attrs: nofree norecurse nosync nounwind uwtable
define dso_local i32 @loop(i32 noundef %0, i32 noundef %1, i32
noundef %2) local_unnamed_addr #1 {
  %4 = load i32, i32* @g, align 4, !tbaa !3
  %5 = icmp sgt i32 %1, %0
  br i1 %5, label %6, label %10
```

LLVM IR — BB1

```
6:                                ; preds = %3
  %7 = sub i32 %1, %0
  %8 = mul i32 %7, %2
  %9 = add i32 %4, %8
  store i32 %9, i32* @g, align 4, !tbaa !3
  br label %10
```

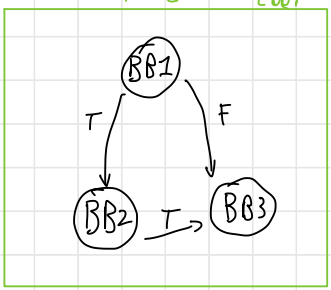
BB2

```
10:                               ; preds = %6, %3
  %11 = phi i32 [ %9, %6 ], [ %4, %3 ]
  ret i32 %11
}
```

BB3

```
int loop(int a, int b, int c) {
  int i, ret = 0;
  for (i = a; i < b; i++) {
    g_incr(c);
}
```

SORGENTE



CFG G_INCREMENT

```
int g;

int g_incr(int c) {
 g += c;
 return g;
}
```

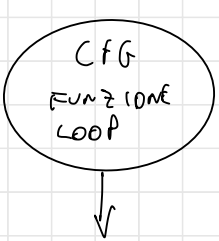
SORGENTE

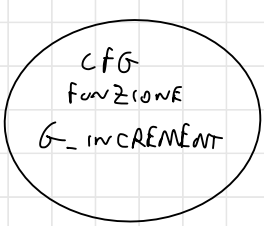
```
@g = dso_local local_unnamed_addr global i32 0, align 4

define dso_local i32 @g_incr(i32 noundef %0) local_unnamed_addr #0
{
 %2 = load i32, i32* @g, align 4, !tbaa !3
 %3 = add nsw i32 %2, %0
 store i32 %3, i32* @g, align 4, !tbaa !3
 ret i32 %3
}
```

LLVM IR — BB

In questo caso, i due CFG relativi alle singole funzioni non sono collegati in quanto, compilando il codice sorgente usando l'ottimizzazione -O2, LLVM ha eseguito una ottimizzazione per cui il corpo della funzione g_increment e' stato direttamente inserito all'interno del basic block relativo al corpo del loop della funzione Loop.

CFG PROGRAMMA



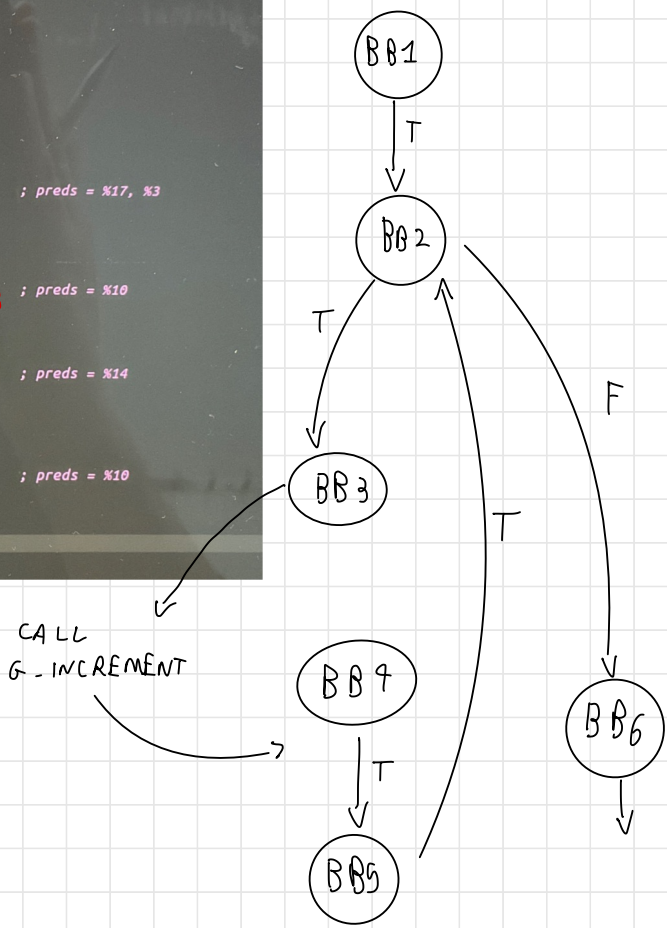


CFG LOOP.c CON COMPILAZIONE -O0

CFG FUNZIONE LOOP

```
; Function Attrs: noinline nounwind optnone uwtable
define dso_local i32 @loop(i32 noundef %0, i32 noundef %1, i32 noundef %2) #0 {
  %4 = alloca i32, align 4 ;a
  %5 = alloca i32, align 4 ;b
  %6 = alloca i32, align 4 ;c
  %7 = alloca i32, align 4 ;i
  %8 = alloca i32, align 4 ;ret
  store i32 %0, i32* %4, align 4
  store i32 %1, i32* %5, align 4
  store i32 %2, i32* %6, align 4
  store i32 0, i32* %8, align 4
  %9 = load i32, i32* %4, align 4
  store i32 %9, i32* %7, align 4
  br label %10

10:                                               ; preds = %17, %3
  %11 = load i32, i32* %7, align 4
  %12 = load i32, i32* %5, align 4
  %13 = icmp slt i32 %11, %12
  br i1 %13, label %14, label %20

14:                                               ; preds = %10
  %15 = load i32, i32* %6, align 4
  %16 = call i32 @g_incr(i32 noundef %15)
  br label %17

17:                                               ; preds = %14
  %18 = load i32, i32* %7, align 4
  %19 = add nsw i32 %18, 1
  store i32 %19, i32* %7, align 4
  br label %10, !llvm.loop !10

20:                                               ; preds = %10
  %21 = load i32, i32* %8, align 4
  %22 = load i32, i32* @g, align 4
  %23 = add nsw i32 %21, %22
  ret i32 %23
}
```

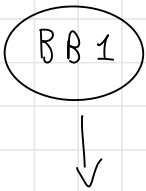
BB1
BB2
BB3
BB4
BB5
BB6



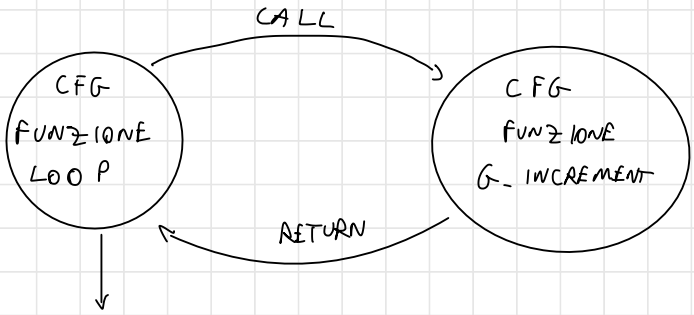
CFG FUNZIONE G_INCREMENT

```
; Function Attrs: noinline nounwind optnone uwtable
define dso_local i32 @g_incr(i32 noundef %0) #0 {
  %2 = alloca i32, align 4
  store i32 %0, i32* %2, align 4
  %3 = load i32, i32* %2, align 4
  %4 = load i32, i32* @g, align 4
  %5 = add nsw i32 %4, %3
  store i32 %5, i32* @g, align 4
  %6 = load i32, i32* @g, align 4
  ret i32 %6
}
```

BB 1



CFG INTERO PROGRAMMA

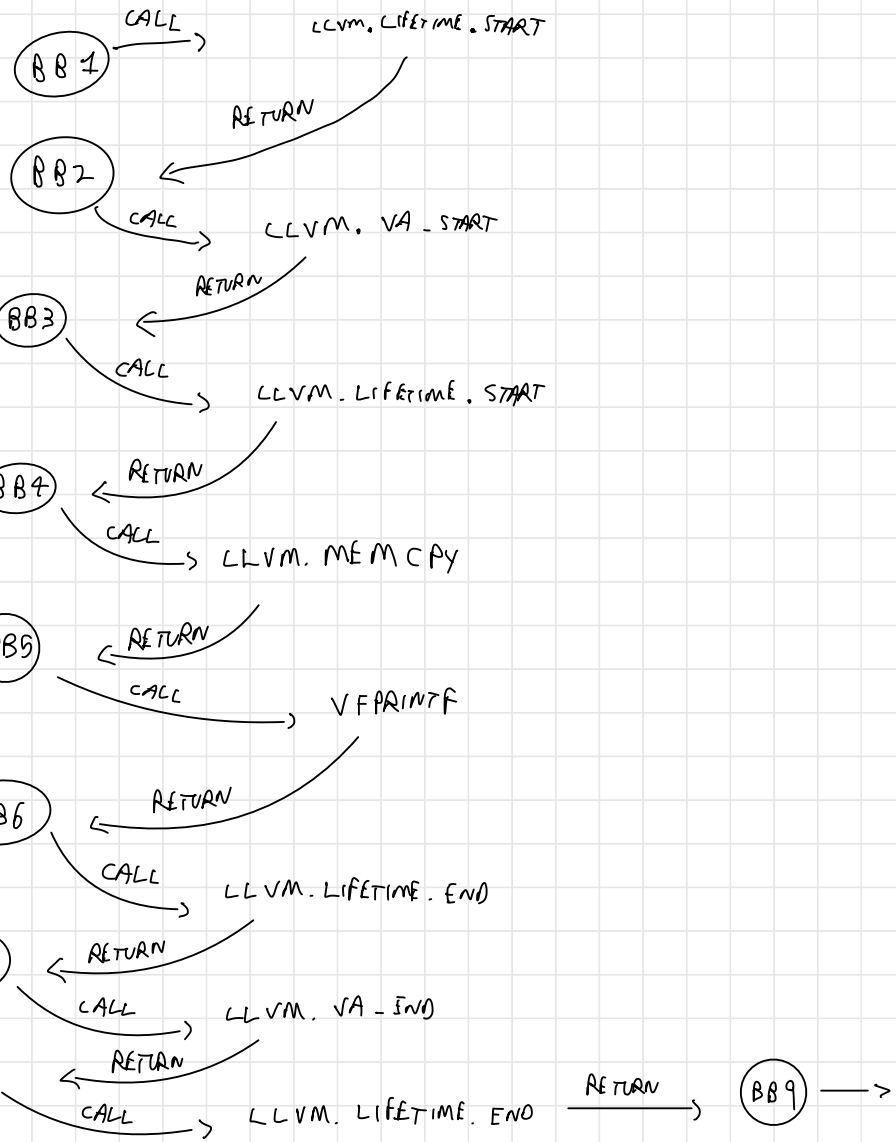


FILE FIBONACCI.C CON COMPILAZIONE -O2

CFG FUNZIONE PRINTF

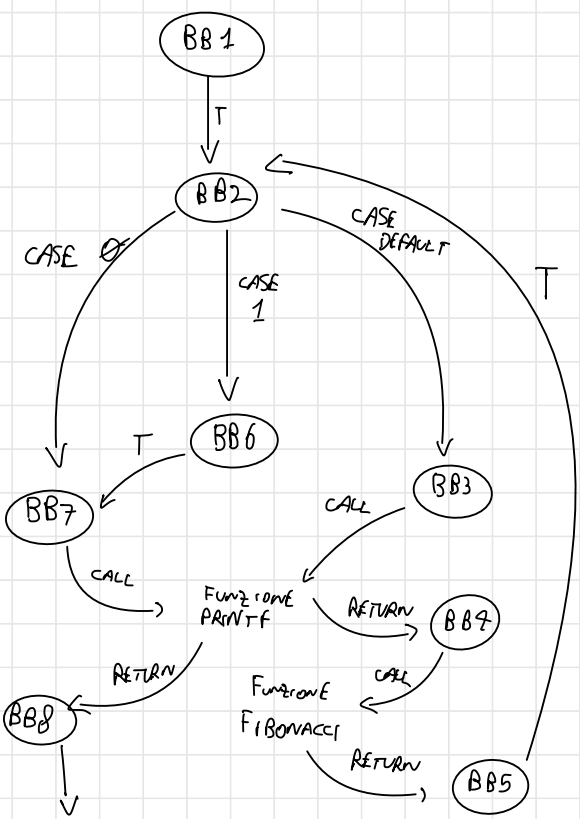
```
; Function Attrs: nofree nounwind uwtable
define dso_local i32 @printf(i8* nocapture noundef readonly %0, ...) local_unnamed_addr #0 {
  %2 = alloca %"struct.std::__va_list", align 8
  %3 = alloca %"struct.std::__va_list", align 8
  %4 = bitcast %"struct.std::__va_list"* %2 to i8*
  call void @llvm.lifetime.start.p0i8(i64 32, i8* nonnull %4) #5
  call void @llvm.va_start(i8* nonnull %4)
  %5 = load %struct._IO_FILE*, %struct._IO_FILE** @stdout, align 8, !tbaa !10
  %6 = bitcast %"struct.std::__va_list"* %3 to i8*
  call void @llvm.lifetime.start.p0i8(i64 32, i8* nonnull %6) #5
  call void @llvm.memcpy.p0i8.p0i8.i64(i8* noundef nonnull align 8 dereferenceable(32) %6, i8* noundef nonnull align 8 dereferenceable(32) %4, i64 32, i1 false), !tbaa.struct !14
  %7 = call i32 @vfprintf(%struct._IO_FILE* noundef %5, i8* noundef %0, %"struct.std::__va_list"* noundef nonnull %3)
  call void @llvm.lifetime.end.p0i8(i64 32, i8* nonnull %6) #5
  call void @llvm.va_end(i8* nonnull %4)
  call void @llvm.lifetime.end.p0i8(i64 32, i8* nonnull %4) #5
  ret i32 %7
}
```

BB1 BB2 BB3 BB4 BB5 BB7 BB9



CFG FUNZIONE FIBONACCI

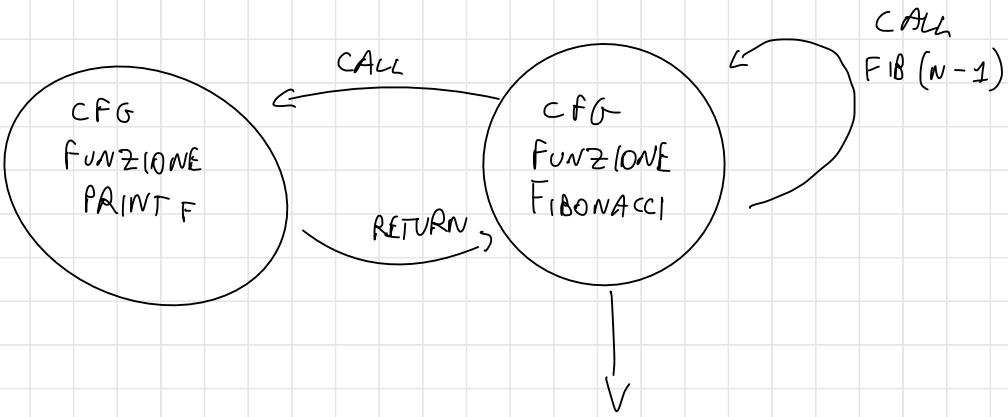
```
define dso_local i32 @Fibonacci(i32 noundef %0) local_unnamed_addr #0 {
  br label %2                                                          BB1

2:                                                ; preds = %5, %1
  %3 = phi i32 [ 0, %1 ], [ %10, %5 ]
  %4 = phi i32 [ %0, %1 ], [ %7, %5 ]                                  BB2
  switch i32 %4, label %5 [
    i32 0, label %12
    i32 1, label %11
  ]

5:                                                ; preds = %2          BB3
  %6 = add nsw i32 %4, -1
  %7 = add nsw i32 %4, -2
  %8 = tail call i32 (i8*, ...) @printf(i8* noundef nonnull dereferenceable(1) getelementptr inbounds ([22 x i8], [22 x i8]* @.str.2, i64 0, i64
0), i32 noundef %4, i32 noundef %6, i32 noundef %7)
  %9 = tail call i32 @Fibonacci(i32 noundef %6)                         BB4
  %10 = add nsw i32 %9, %3                                              BB5
  br label %2

11:                                               ; preds = %2
  br label %12                                                         BB6

12:                                               ; preds = %2, %11    BB7
  %13 = phi i8* [ getelementptr inbounds ([9 x i8], [9 x i8]* @.str.1, i64 0, i64 0), %11 ], [ getelementptr inbounds ([9 x i8], [9 x i8]* @.str,
i64 0, i64 0), %2 ]
  %14 = tail call i32 (i8*, ...) @printf(i8* noundef nonnull dereferenceable(1) %13)
  %15 = add nsw i32 %4, %3                                              BB8
  ret i32 %15
}
```



Il codice mostrato nell'immagine presenta gia' delle grosse ottimizzazione per permettere di ridurre il numero di chiamate ricorsive da effettuare; infatti, all'interno del registro %10 viene via a via salvato il valore della chiamata della funzione di Fibonacci del valore immediatamente precedente a quello che si sta considerando in questo momento.
In questo modo, per ogni numero maggiore di 1, invece che effettuare due chiamate ricorsive (Fib(n-1) e Fib(n-2)), viene chiamato solo Fib(n-1) visto che di sicuro, prima di essere giunto alla chiamata Fib(n-1) devo anche aver effettuato la chiamata per Fib(n-2), e quindi tale valore lo ho gia' calcolato (e' presente ogni volta all'interno del registro %10).

Un'altra ottimizzazione presente all'interno di questo codice fa' riferimento alla gestione dei due casi base: in pratica, invece che definire due blocchi identici, vengono definiti due basic blocks separati: uno e' effettivamente quello che presenta la stampa del valore, mentre l'altro presenta solamente un salto incondizionato.
In questa maniera, gestendo l'arrivo del flusso di controllo al basic block avente le istruzioni mediante l'istruzione PHI, siamo in grado di capire quale valore dobbiamo andare a stampare, se 0 oppure 1.

CFG PROGRAMMA FIBONACCI



FILE FIBONACCI.C CON COMPILAZIONE -O0

CFG FUNZIONE PRINTF

```
; Function Attrs: noinline nounwind optnone uwtable
define dso_local i32 @printf(i8* noundef %0, ...) #0 {
  %2 = alloca i8*, align 8
  %3 = alloca i32, align 4
  %4 = alloca %"struct.std::__va_list", align 8
  %5 = alloca %"struct.std::__va_list", align 8
  store i8* %0, i8** %2, align 8
  %6 = bitcast %"struct.std::__va_list"* %4 to i8*
  call void @llvm.va_start(i8* %6)
  %7 = load %struct._IO_FILE*, %struct._IO_FILE** @stdout, align 8
  %8 = load i8*, i8** %2, align 8
  %9 = bitcast %"struct.std::__va_list"* %5 to i8*
  %10 = bitcast %"struct.std::__va_list"* %4 to i8*
  call void @llvm.memcpy.p0i8.p0i8.i64(i8* align 8 %9, i8* align 8 %10, i64 32, i1 false)
  %11 = call i32 @vfprintf(%struct._IO_FILE* noundef %7, i8* noundef %8, %"struct.std::__va_list"* noundef %5)
  store i32 %11, i32* %3, align 4
  %12 = bitcast %"struct.std::__va_list"* %4 to i8*
  call void @llvm.va_end(i8* %12)
  %13 = load i32, i32* %3, align 4
  ret i32 %13
}
```

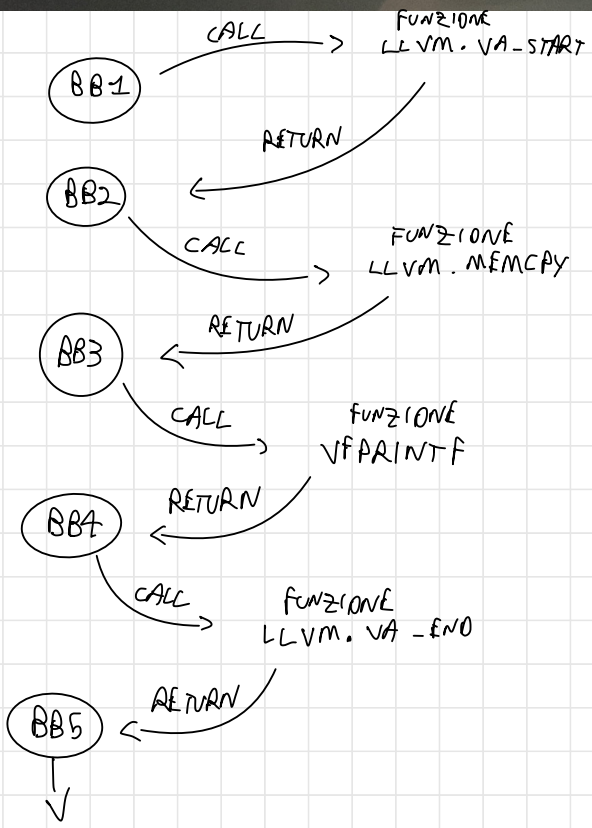
BB1

BB2

BB3

BB4

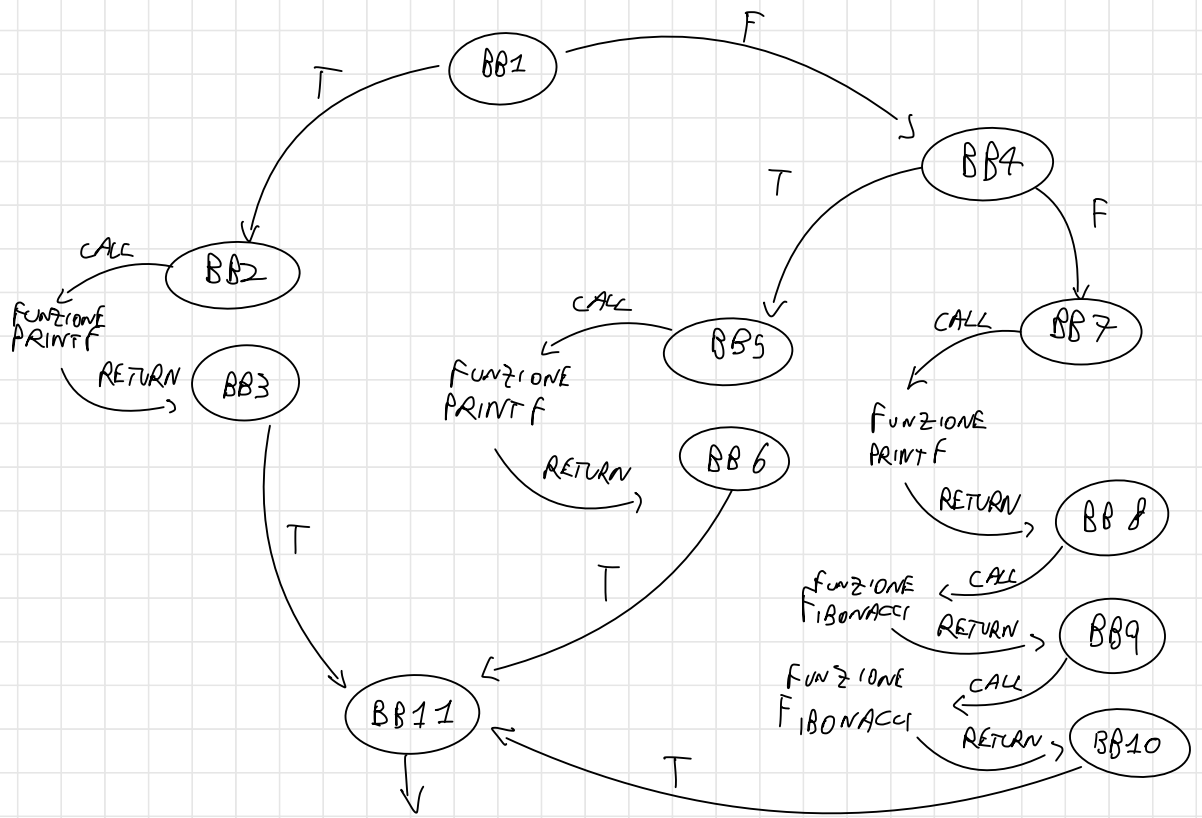
BB5



CFG FUNZIONE FIBONACCI

```
define dso_local i32 @Fibonacci(i32 noundef %0) #0 {
  %2 = alloca i32, align 4
  %3 = alloca i32, align 4
  store i32 %0, i32* %3, align 4
  %4 = load i32, i32* %3, align 4
  %5 = icmp eq i32 %4, 0
  br i1 %5, label %6, label %8

6:                                                ; preds = %1
  %7 = call i32 (i8*, ...) @printf(i8* noundef getelementptr inbounds ([9 x i8], [9 x i8]* @.str, i64 0, i64 0))
  store i32 0, i32* %2, align 4
  br label %27

8:                                                ; preds = %1
  %9 = load i32, i32* %3, align 4
  %10 = icmp eq i32 %9, 1
  br i1 %10, label %11, label %13

11:                                               ; preds = %8
  %12 = call i32 (i8*, ...) @printf(i8* noundef getelementptr inbounds ([9 x i8], [9 x i8]* @.str.1, i64 0, i64 0))
  store i32 1, i32* %2, align 4
  br label %27

13:                                               ; preds = %8
  %14 = load i32, i32* %3, align 4
  %15 = load i32, i32* %3, align 4
  %16 = sub nsw i32 %15, 1
  %17 = load i32, i32* %3, align 4
  %18 = sub nsw i32 %17, 2
  %19 = call i32 (i8*, ...) @printf(i8* noundef getelementptr inbounds ([22 x i8], [22 x i8]* @.str.2, i64 0, i64 0), i32 noundef %14, i32 noundef %16, i32 noundef %18)
  %20 = load i32, i32* %3, align 4
  %21 = sub nsw i32 %20, 1
  %22 = call i32 @Fibonacci(i32 noundef %21)
  %23 = load i32, i32* %3, align 4
  %24 = sub nsw i32 %23, 2
  %25 = call i32 @Fibonacci(i32 noundef %24)
  %26 = add nsw i32 %22, %25
  store i32 %26, i32* %2, align 4
  br label %27

27:                                               ; preds = %13, %11, %6
  %28 = load i32, i32* %2, align 4
  ret i32 %28
}
```

BB 1
BB 2
BB 3
BB 4
BB 5
BB 6
BB 7
BB 8
BB 9
BB 10
BB 11



CFG PROGRAMMA FIBONACCI