

Imperatív programozás

Statikus programszerkezet

Kozsik Tamás és mások

Eötvös Loránd Tudományegyetem

2022. szeptember 3.



Tartalomjegyzék



ELTE
EÖTVÖS LORÁND
TUDOMÁNYEGYETEM

- Kifejezés
- Utasítás
- Alprogram
- Modul



- Nagyobb egység
- Nagy belső kohézió
- Szűk interfész
 - Gyenge kapcsolat modulok között
 - Jellemzően egyirányú

- Előfordítás
- Fordítás
- Szerkesztés (linkelés)
- Optimalizálás
- Kódgenerálás
- stb.

- Preprocessing
- Szövegszerű helyettesítések
- Kimenete a fordítási egység
- `gcc -E main.c`
- #-tel kezdődő sorok

- Konstand érték definíciója
- Az értéke fordításkor meghatározható
- Nem kell, hogy legyen értéke
- Lehet függvényyszerű



Előfordító utasításai

#define

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

```
#define SIZE 10
```

```
int main()
{
    int arr[SIZE];
    int sum = 0;

    for (int i = 0; i < SIZE; ++i)
        sum += arr[i];

    printf("%d\n", sum);
}
```



Előfordító utasításai

#define

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

```
int main()
{
    int arr[SIZE];
    int sum = 0;

    for (int i = 0; i < SIZE; ++i)
        sum += arr[i];

    printf("%d\n", sum);
}
```

Fordítás

```
gcc -DSIZE=10 main.c
```

Előfordító utasításai

#define

```
#define SQR(x) x * x  
printf("%d\n", SQR(5));
```

```
/* 25 */
```



ELTE
EÖTVÖS LORÁND
TUDOMÁNYEGYETEM

Előfordító utasításai

#define

```
#define SQR(x) x * x
printf("%d\n", SQR(5));           /* 25      */
printf("%d\n", SQR(2 + 3));       /* Nem 25  */
printf("%d\n", 2 + 3 * 2 + 3);    /* 11      */
```



Előfordító utasításai

#define

```
#define SQR(x) x * x
printf("%d\n", SQR(5));           /* 25      */
printf("%d\n", SQR(2 + 3));       /* Nem 25 */
printf("%d\n", 2 + 3 * 2 + 3);    /* 11      */

#define SQR(x) (x) * (x)
printf("%d\n", SQR(2 + 3));
printf("%d\n", (2 + 3) * (2 + 3)); /* 25      */
```



Előfordító utasításai

#define

```
#define SQR(x) x * x
printf("%d\n", SQR(5));           /* 25      */
printf("%d\n", SQR(2 + 3));       /* Nem 25 */
printf("%d\n", 2 + 3 * 2 + 3);    /* 11      */

#define SQR(x) (x) * (x)
printf("%d\n", SQR(2 + 3));
printf("%d\n", (2 + 3) * (2 + 3)); /* 25      */
printf("%d\n", 100 / SQR(5));     /* Nem 4   */
printf("%d\n", 100 / (5) * (5));  /* 100     */
```



Előfordító utasításai

#define

```
#define SQR(x) x * x
printf("%d\n", SQR(5));           /* 25      */
printf("%d\n", SQR(2 + 3));       /* Nem 25   */
printf("%d\n", 2 + 3 * 2 + 3);    /* 11      */

#define SQR(x) (x) * (x)
printf("%d\n", SQR(2 + 3));
printf("%d\n", (2 + 3) * (2 + 3)); /* 25      */
printf("%d\n", 100 / SQR(5));     /* Nem 4    */
printf("%d\n", 100 / (5) * (5));  /* 100     */

#define SQR(x) ((x) * (x))
printf("%d\n", 100 / SQR(5));
printf("%d\n", 100 / ((5) * (5))); /* 4       */
```



Előfordító utasításai

#if, #ifdef, #ifndef, #else, #elif, #endif

```
#include <stdio.h>

#ifdef __linux__
void f() {
    printf("This is Linux\n");
}
#elif _WIN32
void f() {
    printf("This is Windows\n");
}
#endif

int main() {
    f();
}
```



Előfordító utasításai

#include

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

```
int main()
{
    srand(time(0));
    int* ptr = (int*)malloc(sizeof(int));
    *ptr = rand() % 100 + 1;
    printf("%d\n", *ptr);
    free(ptr);
}
```



- Fordítási egység a bemenete
Forráskód (source code) forrásfájlban (source file)
 - `factorial.c`
- Fordítóprogram (compiler)
 - `gcc -c factorial.c`
- Tárgykód a kimenete (target code, object code)
 - `factorial.o`

Fordítási hibák

- Nyelv szabályainak megsértése
- Fordítóprogram detektálja

```
factorial.c
```

```
int factorial(int n)
{
    int result = 1;
    for (i = 2; i <= n; ++i)
        result *= i;
    return result;
}
```

```
gcc -c factorial.c
```

```
main.c: In function 'factorial':
```

```
main.c:4:8: error: 'i' undeclared (first use in this function)
```

```
  4 |   for (i = 2; i <= n; ++i)
    |         ^
```

- Tárgykód(ok) a bemenete
 - `factorial.o`
- Szerkesztőprogram (linker)
 - `gcc factorial.o`
- Végrehajtható kód (executable)
 - `a.out` (alapértelmezett név)



Több fordítási egység

factorial.c

```
int factorial(int n)
{
    int result = 1;
    for (int i = 2; i <= n; ++i)
        result *= i;
    return result;
}
```

main.c

```
#include <stdio.h>

int factorial(int n);

int main() {
    printf("%d\n", factorial(5));
}
```

Fordítás, szerkesztés, futtatás

```
$ gcc -c factorial.c -o factorial.o
$ gcc -c main.c -o main.o
$ gcc factorial.o main.o -o a.out
$ ./a.out
$ gcc main.c factorial.c
```

Szerkesztési hiba

factorial.c

```
int factorial(int n)
{
    int result = 1;
    for (int i = 2; i <= n; ++i)
        result *= i;
    return result;
}
```

main.c

```
#include <stdio.h>

int faktorial(int n);

int main() {
    printf("%d\n", faktorial(5));
}
```

Fordítás, szerkesztés, futtatás

```
$ gcc -c factorial.c main.c
$ gcc factorial.o main.o
main.o: In function `main':
main.c:(.text+0xa): undefined reference to `faktorial'
collect2: error: ld returned 1 exit status
```

Fordítási és futási idejű szerkesztés

Statikus szerkesztés

- Még a program futtatása előtt
- A tárgykódok előállítása után „egyből”
- Előnye: szerkesztési hibák fordítási időben

Dinamikus szerkesztés

- A program futtatásakor
- Dinamikusan szerkeszthető tárgykód
 - Linux *shared object*: `.so`
 - Windows *dynamic-link library*: `.dll`
- Előnyei
 - Kisebb végrehajtható állomány
 - Kevesebb memóriafogyasztás

- Header file: `.h`
- Modulok közti interfész
 - `extern`
 - `static`
- Modulban és kliensében `#include`
 - Típus egyeztetés
 - Szerkesztési hibák megelőzése
- Előfeldolgozó (preprocessor)

Fejállományok

Motiváció

calc.h

```
int factorial(int n);  
int square(int n);
```

main.c

```
#include <stdio.h>  
#include "calc.h"  
  
int main()  
{  
    printf("%d\n", factorial(5));  
    printf("%d\n", square(5));  
}
```

calc.c

```
int factorial(int n)  
{  
    int result = 1;  
  
    for (int i = 2; i <= n; ++i)  
        result *= i;  
  
    return result;  
}  
  
int square(int n)  
{  
    return n * n;  
}
```


Include guard

Többszörös beillesztés elkerülésére

`low_level_module.h`

```
#ifndef LOW_LEVEL_MODULE
#define LOW_LEVEL_MODULE
// Some definition...
#endif
```

`middle_module.h`

```
#ifndef MIDDLE_MODULE
#define MIDDLE_MODULE
#include "low_level_module.h"
...
#endif
```

`main.c`

```
#include "low_level_module.h"
#include "middle_module.h"
```

Include guard

```
#ifndef VECTOR_H
#define VECTOR_H

#define VEC_EOK 0
#define VEC_ENOMEM 1

struct VECTOR_S { ... };
typedef struct VECTOR_S* vector_t;

extern int vectorErrno;

void* vectorAt(vector_t v, size_t idx);
void vectorPushBack(vector_t v, void* src);

#endif
```



static, extern

positive.c

```
int positive = 1;
static int negative = -1;
extern int increment;

static void compensate() {
    negative -= increment;
}

void signal() {
    positive += increment;
    compensate();
}
```

main.c

```
#include <stdio.h>

int increment = 3;
extern int positive;
extern void signal();

int main() {
    signal();
    printf("%d\n", positive);
    return 0;
}
```

