ФГОБУ ВПО "СибГУТИ"
Кафедра вычислительных систем

ПРОГРАММИРОВАНИЕ

Технические аспекты использования функций

Преподаватель:

Перышкова Евгения Николаевна

Рассматриваемые вопросы

1. **Основные этапы компиляции программ на языке Си**
    1) **компиляция;**
    2) **компоновка исполняемых файлов;**
    3) **компоновка статических библиотек.**
2. **Роль прототипа в организации вызова функции.**

Основные этапы компиляции программ

1. **Компиляция** одного или нескольких файлов ***.c*** с исходным кодом в объектные файлы ***.o***.

2. **Компоновка** одного или нескольких объектных файлов в исполняемый файл или библиотеку.

Объектный модуль (англ. object file) – файл с промежуточным представлением отдельного файла программы, полученный в результате обработки исходного кода компилятором.

Объектный файл содержит в себе особым образом подготовленный код (т.н. двоичный или бинарный), который может быть объединён с другими объектными файлами при помощи **компоновщика** для получения готового исполнимого модуля, либо библиотеки.

Трансляция

Трансляция (*англ*. translate – переводить): "перевод" программы на языке Си на языка ассемблера, приближенный к машинному языку. *Транслятор* обычно выполняет также диагностику ошибок, формирует словари идентификаторов, выдаёт для печати тексты программы и т. д.

Транслятор, который преобразует программы в машинный язык, принимаемый и исполняемый непосредственно процессором, называется *компилятором*.

В результате компиляции формируются объектные файлы, которые представляют собой:

1. список доступных в данном модуле процедур и данных
2. блоки машинного кода и данных, с *неопределенными адресами ссылок* на данные и процедуры в других объектных модулях, а также.

Пример трансляции программы

main.c

```
int main(){
    in(&x,&y);
    z=calc(x, y);
    out(z);
}
```

inp.c

```
void in(*x, *y){
….
}
void out(z){
….
}
```

calc.c

```
int calc(x,y){
    return x  + y;
}
```

```
$ gcc -c main.c
$ gcc -c inp.c
$ gcc -c calc.c
```

Неопределенные ссылки на функции in, out, calc

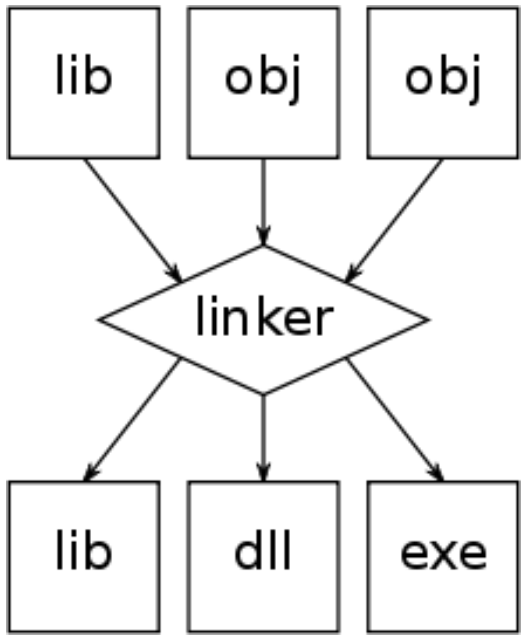
main.o
export: main
need: in, calc, out
main:
011011010111...
[call **in,** call **calc**
call **out**]
10101101101010

inp.o
export: in, out
in:
10011100111001
out:
01010101010010

calc.o
export: calc
calc:
01010101101100
10101110101001
01011101111000

Компоновка

Компоновщик (также редактор связей, линкер – от англ. link editor, linker) — программа, которая производит компоновку: принимает на вход один или несколько объектных модулей и собирает по ним исполнимый или библиотечный модуль.



Компоновка (2)

main.o
export: main
need: in, calc, out
main:
011011010111...
[call **in,** call **calc**
call **out**] …

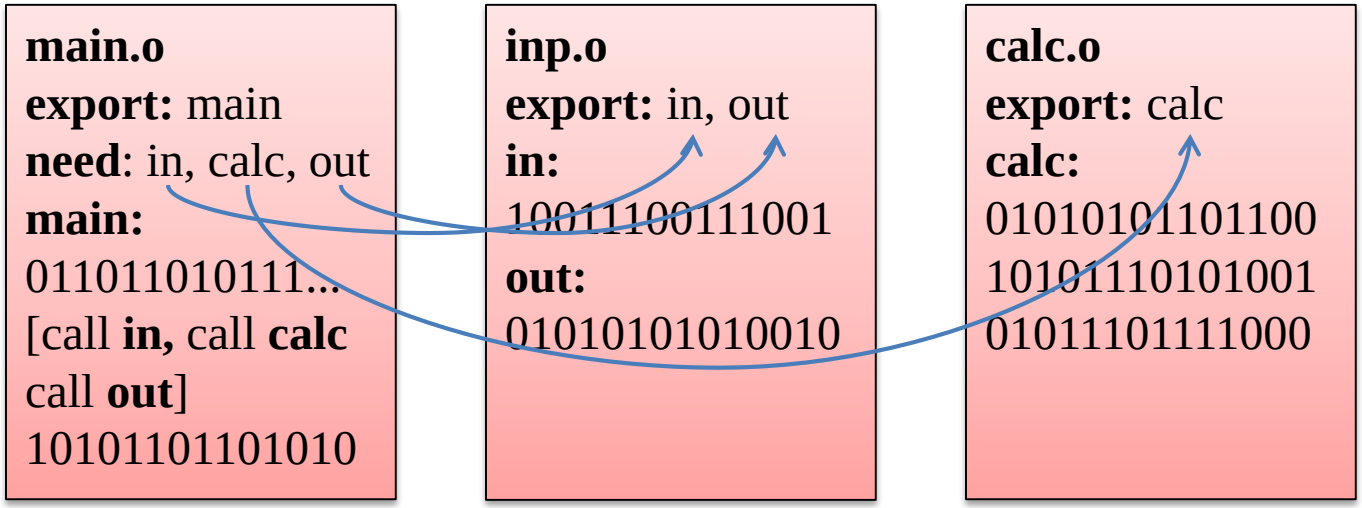
Для связывания модулей компоновщик использует таблицы символов, созданные компилятором в каждом из объектных модулей. Эти таблицы могут содержать символы следующих типов:

1. Экспортируемые имена — функции и переменные, определённые в данном модуле и предоставляемые для использования другим модулям;

2. Неопределённые или импортируемые имена — функции и переменные, на которые ссылается модуль, но не определяет их внутри себя;

3. Локальные — могут использоваться внутри объектного файла для упрощения процесса настройки адресов.

Пример компоновки исполняемой программы




```
$ gcc –o prog main.o inp.o calc.o
```

prog
main: 011011010111… [call **in,** call **calc,** call **out**] 101011011010
in: 10011100111001
out: 01010101010010
calc: 0101010110110010101110101001010111011110000

Пример компоновки статической библиотеки

inp.o
export: in, out
in:
10011100111001
out:
01010101010010

calc.o
export: calc
calc:
01010101101100
10101110101001
01011101111000

Программа ar предназначена для создания, модификации и извлечения файлов из архива. Архив представляет собой набор файлов, размещенных так, что существуют возможность обращения к каждому из них по отдельности.

```
$ ar rc libmy.a inp.o calc.o
```

libmy.a

inp.o
export: in, out
in:
10011100111001
out:
01010101010010

calc.o
export: calc
calc:
01010101101100
10101110101001
01011101111000

Пример компоновки статической библиотеки (2)

libmy.a

inp.o
export: in, out
in:
10011100111001
out:
01010101010010

calc.o
export: calc
calc:
01010101101100
10101110101001
01011101111000

```
$ ranlib libmy.a // построение индекса библиотеки
```

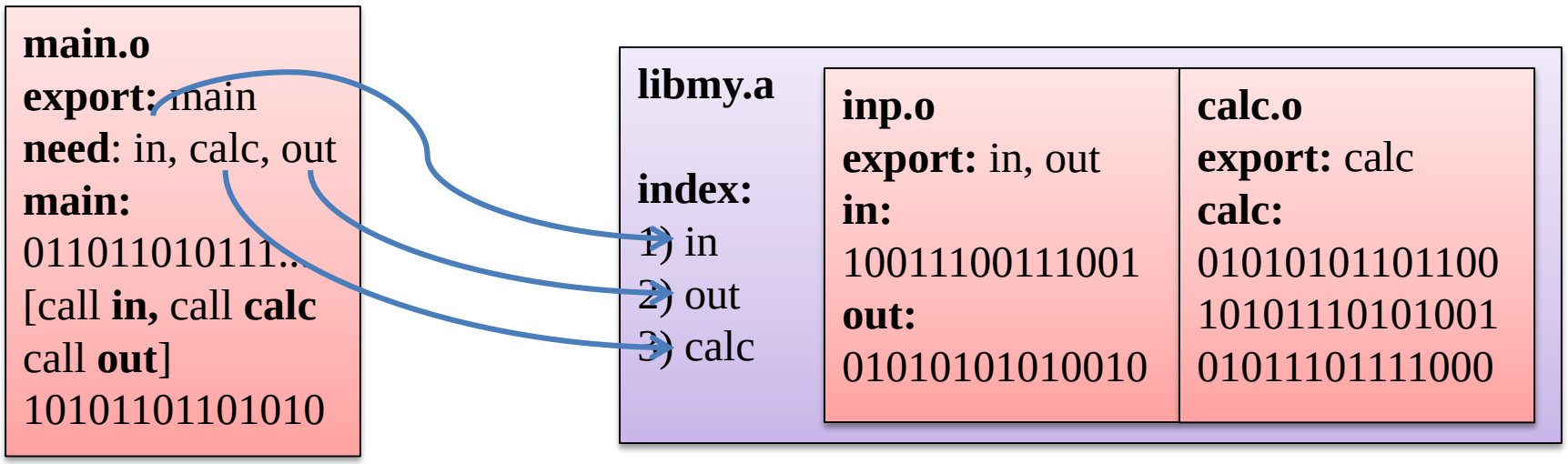
libmy.a

index:
1) in
2) out
3) calc

inp.o
export: in, out
in:
10011100111001
out:
01010101010010

calc.o
export: calc
calc:
01010101101100
10101110101001
01011101111000

Компоновка исполняемого файла с библиотекой




```
$ gcc -c main.c
$ gcc -o prog main.o -L. –lmy
```

либо

```
$ gcc main.c -L. -lfs -o rezult
```

prog
main: 011011010111… [call **in,** call **calc,** call **out**] 101011011010
in: 10011100111001
out: 01010101010010
calc: 0101010110110010101110101001010111011110000

Роль прототипа: возвращаемое значение

main_bad.c:

```
#include <stdio.h>

int main()
{
    float z = func(10);
}
```

func.c:

```
#include <stdio.h>

float func(int x){
    float y = 0.5*x;
    return y;
}
```

```
$ gcc -o test_bad main_bad.c func.c
$ ./test_bad
z = 1084227584.000000
```

Если прототип функции не задан, то по умолчанию в качестве типа возвращаемого значения используется int и производится принятая по умолчанию процедура "повышения типов" (type promotion).

Роль прототипа: возвращаемое значение (2)

main_good.c:

```
#include <stdio.h>
float func(int x);
int main()
{
    float z = func(10);
}
```

func.c:

```
#include <stdio.h>

float func(int x){
    float y = 0.5*x;
    return y;
}
```

```
$ gcc -o test_good main_good.c func.c
$ ./test_good
z = 0.5
```

Сравнительный анализ

main_bad.c:

```
#include <stdio.h>

int main()
{
    float z = func(10);
}
```

```
$ gcc -S main_bad.c
```

```
subq      $16, %rsp
movl      $10, %edi
movl      $0, %eax
call      func
cvtsi2ss  %eax, %xmm0
movss     %xmm0, -4(%rbp)
```

main_good.c:

```
#include <stdio.h>
float func(int x);
int main()
{
    float z = func(10);
}
```

```
$ gcc -S main_bad.c
```

```
subq   $32, %rsp
movl   $10, %edi

call   func

movss  %xmm0, -4(%rbp)
```

cvtsi2ss – функция преобразования (cvt – convert) 32-bit целого (si – signed integer) к вещественному типу с одинарной точностью (ss – scalar single-precision)

Роль прототипа: аргументы

main_bad.c:

```
#include <stdio.h>

int main(){
    short s = 5;
    float f = 2.3f;
    printf("%d\n",x(s,f));
    return 0;
}
```

func.c:

```
#include <stdio.h>
int x(short t, float g)
{
   printf("t=%hd\n",t);
   printf("g=%f\n",g);
   return  (int)(t + g);
}
```

```
$ gcc -o test_bad main_bad.c func.c
$ ./test_bad
t=5
g=36893488147419103232.000000
-2147483648
```

- При вызове функции любые целочисленные параметры "расширяются" до типа int.
- Значения вещественного типа приводятся к типу double.

Роль прототипа: аргументы

main_bad.c:

```
#include <stdio.h>
int x(short t, float g);
int main(){
    short s = 5;
    float f = 2.3f;
    printf("%d\n",x(s,f));
    return 0;
}
```

func.c:

```
#include <stdio.h>
int x(short t, float g)
{
   printf("t=%hd\n",t);
   printf("g=%f\n",g);
   return  (int)(t + g);
}
```

```
$ gcc -o test_bad main_bad.c func.c
$ ./test_bad
t=5
g=2.300000
7
```

Сравнительный анализ (2)

main_bad.c:

```
#include <stdio.h>

int main(){
    short s = 5;
    float f = 2.3f;
    printf("%d\n",x(s,f));
    return 0;
}
```

```
$ gcc -S main_bad.c
```

```
movw        $5, -6(%rbp)
movl        .LC0(%rip), %eax
movl        %eax, -4(%rbp)
movss       -4(%rbp), %xmm0
cvtps2pd    %xmm0, %xmm0
movswl      -6(%rbp), %eax
movl        %eax, %edi
movl        $1, %eax
call        x
```

main_bad.c:

```
#include <stdio.h>
int x(short t, float g);
int main(){
    short s = 5;
    float f = 2.3f;
    printf("%d\n",x(s,f));
    return 0;
}
```

```
$ gcc -S main_good.c
```

```
movw        $5, -6(%rbp)
movl        .LC0(%rip), %eax
movl        %eax, -4(%rbp)
movswl      -6(%rbp), %edx
movl        -4(%rbp), %eax
movl        %eax, -20(%rbp)
movss       -20(%rbp), %xmm0
movl        %edx, %edi
call        x
```

Сравнительный анализ (3)

```
movw        $5, -6(%rbp)
movl        .LC0(%rip), %eax
movl        %eax, -4(%rbp)
movss       -4(%rbp), %xmm0
cvtps2pd    %xmm0, %xmm0
movswl      -6(%rbp), %eax
movl        %eax, %edi
movl        $1, %eax
call        x
```

cvtps2pd – преобразовать (cvt – convert) два упакованных вещественных числа с одинарной точностью (ps - packed single-precision floating-point) в два вещественных числа с двойной точностью (pd – packed double-precision floating-point).

cvtps2pd %xmm0, %xmm1