



ISO/ANSI C

Podział kodu programu na pliki

180

ISO/ANSI C - biblioteki

Dzielenie kodu na kilka plików źródłowych

- Duży kod warto dzielić na fragmenty o wspólnej funkcjonalności (np. funkcje numeryczne, funkcje we/wy, funkcje dostępu do plików, itp.)
- Fragmenty należy umieszczać w oddzielnych plikach.
- Pliki dołączamy do naszego pliku za pomocą dyrektywy **#include**, np.:


```
#include "sortowanie.c"
```
- Połączenia dokonuje preprocesor kodu.
- Preprocesor kodu składa postać pośrednią pliku do kompilacji zgodnie z dyrektywami.
- Dyrektywa zaczyna się od znaku # i nie kończy się średnikiem.
- Każda dyrektywa występuje w osobnej linii.

© UKSW, WMP, SNS, Warszawa 181

181

ISO/ANSI C - biblioteki

Łączenie – przykład:

Kod z pliku wskazanego przez **#include** jest łączony z kodem naszego pliku tworząc w momencie kompilacji postać pośrednią, tj. jedną dużą całość ułożoną sekwencyjnie wg kolejności dołączeń:

Plik A.txt:

```
Był skrzypek rodem z Prabusów,  
#include "B.txt"  
od skrzypiec zamiast butów.
```

Plik B.txt:

```
miął nogi za duże do butów.  
Wszystkie go uwierały,  
więc nosił futerały
```

Jaki tekst wygeneruje preprocesor kodu przetwarzając plik A.txt?

© UKSW, WMP, SNS, Warszawa 182

182

ISO/ANSI C - biblioteki

Dzielenie kodu na kilka plików źródłowych

- Przy wielopoziomowych włączeniach kodu pojawia się problem.

Przykład:

A korzysta z B i C. B korzysta z D, a także C korzysta z D. Wtedy D zostanie dołączony dwa razy.

W finalnej postaci kodu całego programu przygotowanej przez preprocesor funkcje z D pojawią się dwa razy – wystąpi błąd kompilacji. ☹

© UKSW, WMP, SNS, Warszawa 183

183

ISO/ANSI C - biblioteki

lato.h:
Lato Lato wszędzie
Zwarowało oszalało moje serce
Lato Lato wszędzie
A ty dziewczę zaraz wpadniesz w moje ręce

Zwrotka1.h:
#include lato.h
Rzecz między nami była cicha
Westchnąłem do ciebie
Tak jak się wzdycha
I było nam ciasno, miło
Duzo się spało i często się pilo
No i czego, czego jeszcze chcesz?

Zwrotka2.h:
#include lato.h
Pisze i wymyśla słowa piosenki
Zebyś pomyślała jak jestem wielki
I nie wiesz że to właśnie ja
Chcę dać ci wielki wino balon
No i czego, czego jeszcze chcesz?

Zwrotka3.h:
#include lato.h
Ptaki zaczęły świtem na niebie
Zaspiewałem kilka dźwięków tylko dla ciebie
I w oczy twoje zamglone spoglądam
Krzyczę do ucha "Ciebie pozostanę"
Tylko ciebie ciebie jeszcze chcę

main.cpp:
#include "Zwrotka1.h"
#include "Zwrotka2.h"
#include "Zwrotka3.h"

© UKSW, WMP, SNS, Warszawa 184

184

ISO/ANSI C - biblioteki

Dzielenie kodu na kilka plików źródłowych

- Przy wielopoziomowych włączeniach kodu pojawia się problem.

Przykład:

A korzysta z B i C. B korzysta z D i C korzysta z D. Wtedy D zostanie dołączony dwa razy.

- Rozwiązanie: należy włączać same deklaracje, a nie definicje funkcji. Natomiast definicje podać tylko raz, na końcu kodu programu.

© UKSW, WMP, SNS, Warszawa 185

185

ISO/ANSI C - biblioteki

Dzielenie kodu na kilka plików źródłowych

Deklaracja:

```
char flip(char , struct klucz );
```

Definicja:

```
char flip(char c, struct klucz k) {  
    int i;  
    for(i=0; i<24; i++)  
        if (c==k.mapa[i]) return k.mapa[(i+k.skok)%10];  
    return c;  
};
```

© UKSW, WMP, SNS, Warszawa

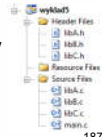
186

186

ISO/ANSI C - biblioteki

Dzielenie kodu na kilka plików źródłowych

- Tworząc biblioteki, dla każdego zestawu funkcji tworzymy dwa pliki: z deklaracjami i z definicjami. Np. biblioteka z funkcjami do sortowania mogłaby mieć pliki: `sortowanie.h` i `sortowanie.c`
- w pliku z metodą `main` dołączamy tylko nagłówki, np.:
`#include "sortowanie.h"`
- Jak i kiedy dołączamy plik z definicjami „.c”?
- Do tego służy „projekt” w środowisku programistycznym. W ramach zakładanego projektu wskazujemy wszystkie pliki „.c” oraz wszystkie pliki „.h”, które zawierają niezbędny kod naszego programu.



© UKSW, WMP, SNS, Warszawa

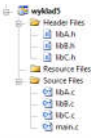
187

187

ISO/ANSI C - biblioteki

Dzielenie kodu na kilka plików źródłowych

```
!>----- Rebuild All started: Project: wyklat5, Configuration: Release|Win32 -----  
!>Deleting intermediate and output files for project 'wyklat5', configuration 'Release|Win32'  
!>Compiling...  
!>libC.c  
!>libB.c  
!>libA.c  
!>main.c  
!>Linking...  
!>Generating code  
!>Finished generating code  
!>Embedding manifest...  
!>Build log was saved at "file:///d:/Pracownia/Lakademicka-socjazyk/uksw-mp-programowanie_obiekt  
!>wyklat5 - 0 error(s), 0 warning(s)  
***** Rebuild All: 0 succeeded, 0 failed, 0 skipped *****
```



Proces kompilacji jest dwuetapowy:

1. Poszczególne pliki *.c kompilowane są kolejno; kody funkcji bibliotecznych nie są jeszcze konieczne (wystarczą same nagłówki).
2. Pliki wynikowe kompilacji są łączone (linkowane) w jeden plik *.exe

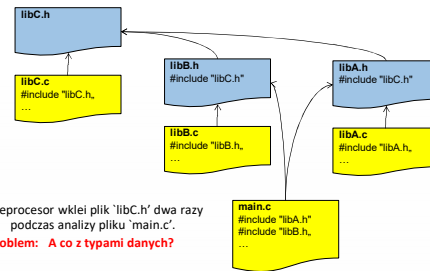
© UKSW, WMP, SNS, Warszawa

188

188

ISO/ANSI C - biblioteki

Przykład:



Preprocesor wklei plik 'libC.h' dwa razy podczas analizy pliku 'main.c'.

Problem: A co z typami danych?

© UKSW, WMP, SNS, Warszawa

189

189

ISO/ANSI C - biblioteki

Budowa pliku nagłówkowego *.h

Co zrobić, żeby plik 'libC.h' jednak nie został wklejony dwa razy?

Wykorzystać dyrektywy preprocesora:

```
#ifndef nazwaPliku_h  
#define nazwaPliku_h  
/*  
    tutaj deklaracje funkcji  
*/  
#endif
```

#define – definiuje (tworzy) nową stałą „nazwaPliku_h”

#ifndef – sprawdza, czy nie jest zdefiniowana stała „nazwaPliku_h”. Jeżeli nie, włącza kod znajdujący się poniżej tej dyrektywy, do kodu wyjściowego.

#endif – znacznik końca tekstu objętego funkcją `#ifndef`

© UKSW, WMP, SNS, Warszawa

190

190

ISO/ANSI C - biblioteki

lato.h:

```
#ifndef LATO_H  
#define LATO_H  
Lato Lato wszędzie  
Zwironowało oszaleło moje serce  
Lato Lato wszędzie  
A ty dziewczę zaraz wpadniesz w moje ręce  
#endif
```

Zwrotka1.h:

```
#include lato.h  
Rzecz między nami była cicha  
Westchnęłam do ciebie  
Tak jak się wzdycha  
I było nam cisno, miło  
Duzo się spało i często się pilo  
No i czego, czego jeszcze chcesz?
```

Zwrotka2.h:

```
#include lato.h  
Piśze i wymyślam słowa piosenki  
Żebyś pomyślała jak jestem wielki  
I nie wiesz że to właśnie ja  
Chce dać ci wielki wina balon  
No i czego, czego jeszcze chcesz?
```

Zwrotka3.h:

```
#include lato.h  
Praki zarywały światem na niebie  
Zaspiewałem kilka dźwięków tylko dla ciebie  
I w oczy twoje zamglone spoglądam  
Krzycząc do słońca "Ciebie posągam"  
Tylko ciebie ciebie jeszcze chcę
```

main.cpp:

```
#include "Zwrotka1.h"  
#include "Zwrotka2.h"  
#include "Zwrotka3.h"
```

© UKSW, WMP, SNS, Warszawa

191

191

ISO/ANSI C - biblioteki

Budowa pliku nagłówkowego *.h

Podsumowując:
użycie dyrektyw **#ifndef**, **#define**, **#endif** gwarantuje, że niezależnie ile razy pojawi się dyrektywa **#include "libc.h"** treść pliku zostanie dołączona w tylko jednym egzemplarzu.

- Nazwy zmiennych, które definiujemy za pomocą preprocesora muszą być unikatowe podczas kompilacji projektu dla każdego używanego pliku.
- Najpopularniejszą metodą zapewnienia sobie unikatowych nazw zmiennych, jest używanie nazwy pliku dla zmiennej preprocesora.

© UKSW, WMP, SNS, Warszawa

192

192

ISO/ANSI C - biblioteki

Budowa pliku źródłowego *.c

```
#include "nazwaPliku.h"

/*
  tutaj definicje funkcji
*/
```

© UKSW, WMP, SNS, Warszawa

193

193

ISO/ANSI C - biblioteki

Alternatywą dla:

```
#ifndef nazwaPliku_h
#define nazwaPliku_h
/*
  tutaj deklaracje funkcji
*/
#endif
```

w środowisku Visual Studio jest

```
#pragma once
```

Zapobiega wielokrotnemu załączeniu treści całego pliku.
Ale nie należy do standardu..

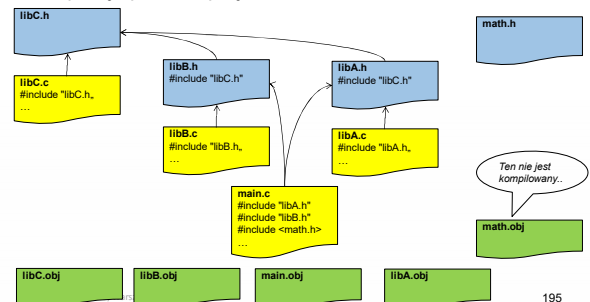
© UKSW, WMP, SNS, Warszawa

194

194

ISO/ANSI C - biblioteki

Kompilacja plików z projektu:



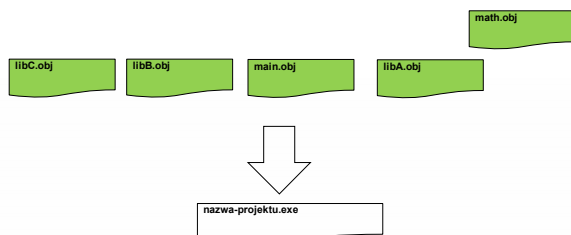
© UKSW, WMP, SNS, Warszawa

195

195

ISO/ANSI C - biblioteki

Linkowanie wyników kompilacji:



© UKSW, WMP, SNS, Warszawa

196

196

ISO/ANSI C - biblioteki

Zmienne zewnętrzne

Zmienne są „widziane” w kodzie znajdującym się poniżej ich deklaracji w obrębie bloku danych a najlepszym razie w obrębie pliku

Zmienne zadeklarowane w pliku poza ciałem funkcji nazywane są zmiennymi globalnymi

Aby zmienna globalna z jednego pliku była „widziana” w drugim, musi zostać zadeklarowana z modyfikatorem **extern**

Zadeklarowanie:

```
extern double x;
```

stanowi informację dla kompilatora, że zmienna ta jest lub będzie zdefiniowana w innym pliku/module

© UKSW, WMP, SNS, Warszawa

197

197

ISO/ANSI C - biblioteki

Zmienne zewnętrzne

Deklaracja zmiennej z modyfikatorem **extern** :

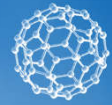
- deklaracja zmiennej nie jest związana z jej definicją,
- żadna zmienna nie jest tworzona, tzn. pamięć nie jest przydzielana,
- ta sama zmienna może być zadeklarowana jako **extern** wiele razy w wielu plikach, ale zdefiniowana może być tylko raz,
- zmiennej deklarowanej nie wolno inicjować:
`extern double x = 0; /* Nie! */`

Po takiej inicjalizacji kompilator zignoruje słowo kluczowe **'extern'** i potraktuje powyższą deklarację jak definicję. Kompilator po znalezieniu właściwej definicji nie zwróci komunikatu o błędzie, ponieważ będzie ona znajdowała się w innym pliku, a więc będzie traktowana jako definicja innej zmiennej, co prawda o tej samej nazwie, ale innej – bo w innym pliku.

© UKSW, WMP, SNS, Warszawa

198

198



ISO/ANSI C zakończenie

199

ISO/ANSI C – zakończenie

Odczyt i zapis bieżącej daty i czasu odbywa się za pomocą funkcji zdefiniowanych w bibliotece `<time.h>`

Zdefiniowane są tam następujące typy i stałe:

`CLOCKS_PER_SEC` – liczba „tyknięć” na sekundę

`clock_t`, `time_t` – typy arytmetyczne do reprezentacji czasu

```
struct tm {
    int tm_sec      /* liczba sekund [0-61] (61 pozwala na 2 sekundy przestępne) */
    int tm_min     /* liczba minut [0-59] */
    int tm_hour    /* liczba godzin po północy [0-23] */
    int tm_mday    /* dzień miesiąca [1-31] */
    int tm_mon     /* miesiąc w roku [0-11] */
    int tm_year    /* bieżący rok-1900 */
    int tm_wday    /* nr dnia licząc od niedzieli [0-6] */
    int tm_yday    /* nr dnia licząc od pierwszego stycznia [0-365] */
    int tm_isdst   /* znacznik uwzględniania czasu zimowego i letniego */
}
```

Struktura używana do reprezentacji czasu kalendarzowego

© UKSW, WMP, SNS, Warszawa

200

200

ISO/ANSI C – zakończenie

Sekunda przestępna, nazywana też **sekundą skokową**

– dodatkowa sekunda dodawana czasem (zwykle w czerwcu lub w grudniu) w celu zsynchronizowania uniwersalnego czasu koordynowanego ze średnim czasem słonecznym.

<https://www.gum.gov.pl/pl/wiadomosci/informacje-i-komunikaty/sekunda-przestepna/>

© UKSW, WMP, SNS, Warszawa

201

201

ISO/ANSI C – zakończenie

clock_t clock(void);

Zwraca liczbę „tyknięć” od chwili uruchomienia danego procesu. Aby dostać liczbę sekund należy podzielić tą wartość przez `CLOCKS_PER_SEC`

double difftime(time_t timer1, time_t timer0);

Różnica w sekundach pomiędzy dwoma wskazaniami czasu

time_t time(time_t *timer);

Aktualny czas systemowy

Uwaga: zmienna typu `time_t` reprezentuje wskazania czasu od północy, 1 stycznia 1970 do 3:14:07, 19 stycznia 2038

© UKSW, WMP, SNS, Warszawa

202

202

ISO/ANSI C – zakończenie

Przykład:

```
time_t start, finish;
long loop;
double result, elapsed_time;

time( &start );
for( loop = 0; loop < 500000000; loop++ )
    result = 3.63 * 5.27;
time( &finish );
elapsed_time = difftime( finish, start );
```

© UKSW, WMP, SNS, Warszawa

203

203

ISO/ANSI C – zakończenie

```
time_t mktime( struct tm *timeptr );  
Konwertuje dane zapisane w strukturze reprezentującej typ kalendarzowy do postaci  
wskazania czasu typu time_t  
  
char *asctime( const struct tm *timeptr );  
Konwertuje dane zapisane w strukturze reprezentującej typ kalendarzowy do postaci  
tekstowej, np.:  
Sun Feb 03 11:38:58 2002  
  
struct tm *localtime( const time_t *timer );  
Konwertuje wskazanie czasu na typ kalendarzowy wg czasu lokalnego  
  
struct tm *gmtime( const time_t *timer );  
Konwertuje wskazanie czasu typu time_t do postaci struktury tm
```

© UKSW, WMP, SNS, Warszawa

204

204

ISO/ANSI C – zakończenie

Przykład:

```
struct tm *newtime;  
time_t aclock;  
  
time( &aclock ); /* odczytaj czas */  
newtime = localtime( &aclock ); /* Konwertuj do postaci struct tm */  
  
/* Wypisz czas lokalny w oknie konsoli */  
printf( „Bieżąca data i czas: %s”, asctime( newtime ) );
```

© UKSW, WMP, SNS, Warszawa

205

205

ISO/ANSI C – zakończenie

```
size_t strftime( char *strDest, size_t maxsize, const char *format, const struct tm *timeptr  
);  
Formatuje zapis tekstowy daty i czasu  
%a Abbreviated weekday name  
%A Full weekday name  
%b Abbreviated month name  
%B Full month name  
%c Date and time representation appropriate for locale  
%d Day of month as decimal number (01 – 31)  
%H Hour in 24-hour format (00 – 23)  
%I Hour in 12-hour format (01 – 12)  
%j Day of year as decimal number (001 – 366)  
%m Month as decimal number (01 – 12)  
%M Minute as decimal number (00 – 59)  
%p Current locale's A.M./P.M. indicator for 12-hour clock  
%S Second as decimal number (00 – 59)  
%U Week of year as decimal number, with Sunday as first day of week (00 – 53)  
%w Weekday as decimal number (0 – 6, Sunday is 0)  
%W Week of year as decimal number, with Monday as first day of week (00 – 53)  
%x Date representation for current locale  
%X Time representation for current locale  
%y Year without century, as decimal number (00 – 99)  
%Y Year with century, as decimal number  
%z, %Z Either the time-zone name or time-zone abbreviation, depending on registry settings; no chars if time zone is unknown  
%% Percent sign  
© UKSW, WMP, SNS, Warszawa
```

206

206

ISO/ANSI C – zakończenie

Przykład:

```
time_t rawtime;  
struct tm * timeinfo;  
char buffer [80];  
  
time ( &rawtime );  
timeinfo = localtime ( &rawtime );  
  
strftime (buffer,80, „Teraz jest %I:%M%p.”,  
timeinfo);  
  
puts (buffer);  
  
Na wyjściu:  
Teraz jest 03:21PM.
```

© UKSW, WMP, SNS, Warszawa

207

207

ISO/ANSI C – zakończenie

Zmienne systemowe (PATH, LIB, itd.) można odczytać używając funkcji `getenv`

```
char *getenv( const char *varname );
```

Przykład:

```
char *pathvar;  
pathvar = getenv( "PATH" );  
if ( pathvar != NULL )  
printf( "PATH variable is: %s\n", pathvar );
```

© UKSW, WMP, SNS, Warszawa

208

208

ISO/ANSI C – zakończenie

Polecenie systemowe można wywołać używając funkcji `system`

```
int system( const char *command );
```

Przykład:

```
system( "type crt.txt" );  
/* wypisuje w oknie konsoli zawartość pliku  
tekstowego crt.txt */
```

© UKSW, WMP, SNS, Warszawa

209

209

ISO/ANSI C – zakończenie

Aby zakończyć działanie programu musi być wykonana instrukcja: `exit`, `abort`, lub `return` w funkcji `main`

`void abort(void);`

Przerywa działanie programu, nie zwraca sterowania do procesu nadrzędnego ale wyświetla komunikat:

*This application has requested the Runtime to terminate it in an unusual way.
Please contact the application's support team for more information.*

© UKSW, WMP, SNS, Warszawa

210

210

ISO/ANSI C – zakończenie

`void exit(int status);`

Funkcja kończy działanie programu w trybie prawidłowego zakończenia

Standardowo przyjmuje się dwie wartości status:

0 (**EXIT_SUCCESS**) – program zakończony prawidłowo;

1 (**EXIT_FAILURE**) - program zakończony niepoprawnie

Przykład:

```
exit( 0 );
```

© UKSW, WMP, SNS, Warszawa

211

211

ISO/ANSI C – zakończenie

Funkcja `exit` przez wyjściem z programu wywołuje wszystkie funkcje wskazane wywołaniem funkcji `atexit`. Może być ich do 32.

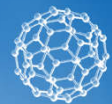
Przykład:

```
void fn1(void) {  
    printf( "do jutra.\n" );  
}  
void fn2(void) {  
    printf( "Do widzenia " );  
}  
int main( void ) {  
    atexit( fn1 );  
    atexit( fn2 );  
    exit( 0 );  
} /* Co pojawi się w oknie konsoli? */
```

© UKSW, WMP, SNS, Warszawa

212

212



Koniec części wykładu dotyczącej ISO/ANSI C

213