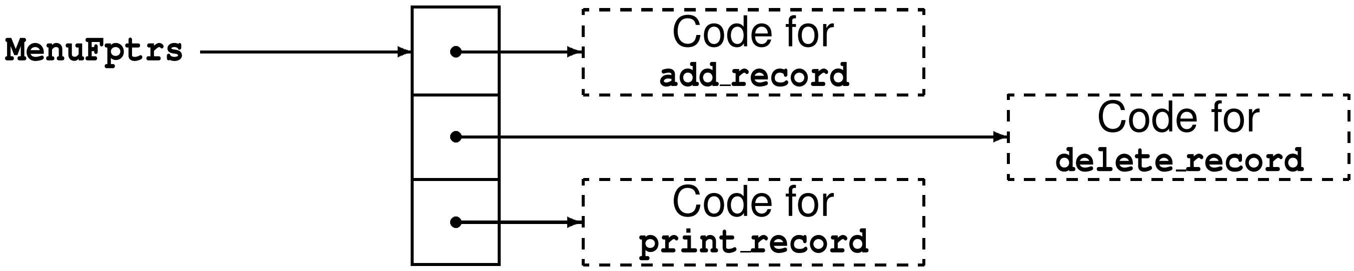
# CHAPTER 17

# COMPLEX DECLARATIONS

# AN ARRAY OF POINTERS TO FUNCTIONS



# record.h

(Figure 17.2 – Page 425)

```
/*
 * Prototypes for dummy functions used to test the menu.
 */
extern void add_record(void);
extern void del_record(void);
extern void print_record(void);
```

# record.c

(Figure 17.3 – Page 425)

```c
/*
 * Dummy functions to test our menu program.
 */
#include <stdio.h>
#include "record.h"

#define ANNOUNCE(x) printf("In %s\n", x)

void add_record(void)   { ANNOUNCE("add_record"); }

void del_record(void)   { ANNOUNCE("del_record"); }

void print_record(void) { ANNOUNCE("print_record"); }
```

# usemenu.c

(Figure 17.4 – Page 426)

```
/*
 * Print menu and then obtain and execute the user's choice.
 */
#include <stdio.h>
#include "record.h"

const char *const MenuEntries[] =
{
  "Add record", "Delete record", "Print record"
};

void (*MenuFptrs[])(void) =
{
  add_record, del_record, print_record
};

int main()
{
  void menu_process(int, const char *const[], void (*[])(void));
  const int choices = sizeof(MenuFptrs)/sizeof(MenuFptrs[0]);

  menu_process(choices, MenuEntries, MenuFptrs);
  return 0;
}
```

# prmenu.c

(Figure 17.5 – Page 427)

```
/*
 * Display menu and obtain and execute user choices.
 *   menu_process - obtain and execute user's choice.
 *   menu_choice - request and get the user's choice.
 */
#include <stdio.h>

#define MAXLEN 80

static int menu_choice(int n, const char *const items[])
{
  int  getline(char *buf, int buflen);
  int  i, num;                          /* index, selection */
  char line[MAXLEN + 1];

  printf("Your choices are:\n");
  for (i = 1; i <= n; i++)
    printf("\t%i) %s\n", i, items[i - 1]);
  printf("Or type a 0 to quit: ");
  while (getline(line, MAXLEN) != -1)
    if (sscanf(line, "%i", &num) == 1 && 0 <= num && num <= n)
      return num;
    else
      printf("Enter a value between 0 and %i: ", n);
  return 0;
}

void menu_process(int n,
                  const char *const items[],
                  void (*fptrs[])(void))
{
  int choice;                            /* holds user's choice */

  while ((choice = menu_choice(n, items)) != 0)
    fptrs[choice - 1]();
}
```

# menu.h

(Figure 17.6 – Page 428)

```
/*
 * Definition of a menu structure.
 */
typedef struct menu_item
{
  char *entry;                          /* string to write */
  void (*fptr)(void);                   /* function to execute */
} MENU_ITEM;

void menu_process(int n, const MENU_ITEM tab[]);
```

# prmenu2.c

(Figure 17.7 – Page 429)

```
/*
 * Display menu and obtain/execute user choices.
 *   menu_process - get and execute user's choice.
 *   menu_choice - read and verify choice from user.
 *   menu_search - search menu to find user's choice.
 */
#include <stdio.h>
#include <stddef.h>
#include <string.h>
#include "menu.h"

#define MAXLEN 80

static const MENU_ITEM *menu_search(const MENU_ITEM *ptr,
                                    const MENU_ITEM *endptr,
                                    const char *target)
{
  for (; ptr < endptr; ptr++)
    if (strncmp(target, ptr->entry, strlen(target)) == 0)
      return ptr;
  return NULL;
}
```

# prmenu2.c

(Figure 17.7 – Page 429)

```
static void (*menu_choice(int n, const MENU_ITEM menu[]))(void)
{
  int   getline(char *buf, int buflen);
  char  line[MAXLEN + 1];
  const MENU_ITEM *ptr;
  const MENU_ITEM *const endptr = menu + n;

  printf("Your choices are:\n");
  for (ptr = menu; ptr < endptr; ptr++)
    printf("\t%s\n", ptr->entry);
  printf("Or type QUIT to quit: ");
  while (getline(line, MAXLEN) != -1 && strcmp(line, "QUIT") != 0)
  {
    if ((ptr = menu_search(menu, endptr, line)) != NULL)
      return ptr->fptr;
    printf("Select an appropriate menu entry: ", n);
  }
  return NULL;
}

void menu_process(int n, const MENU_ITEM menu[])
{
  void (*choiceptr)(void);               /* user-selected function */

  while ((choiceptr = menu_choice(n, menu)) != NULL)
    choiceptr();                         /* execute user's choice */
}
```

# usemenu2.c

(Figure 17.8 – Page 430)

```
/*
 * Menu processor, rewritten to represent menus with structures.
 */
#include "menu.h"
#include "record.h"

MENU_ITEM Menu[] =
{
  {"Add record",    add_record},
  {"Delete record", del_record},
  {"Print record",  print_record}
};

int main()
{
  const int choices = sizeof(Menu)/sizeof(Menu[0]);

  menu_process(choices, Menu);
  return 0;
}
```

# DECLARATORS

- A type declaration consists of a basic type followed by a *declarator*.
- It specifies the type of the identifier contained within the declarator.
- For example: `double (*funcptr)(void);`
  - `double` is the basic type.
  - `(*funcptr)(void)` is the declarator.
  - `funcptr` is the identifier within the declarator.
  - So `funcptr` is a pointer to a function that takes no arguments and returns a `double`.

# SIMPLE DECLARATORS

- Simplest declarators are a type plus a variable:

```
int    i;              /* an int */
double d;              /* a double */
```

- More complex declarators combine identifiers with **`*`**, **`()`**, and **`[]`**.

```
int  *iptr;            /* pointer to an int */
char *cptr;            /* pointer to a char */

double dtab[10];        /* array of 10 doubles */
char   ctab[10];        /* array of 10 characters */

int    ifunc(void);    /* function returning an int */
double dfunc(void);    /* function returning a double */
```

# MORE COMPLEX DECLARATORS

- We can combine these pieces but we have to worry about operator precedence.

- When combining declarators, `*` has lower precedence than either `()` or `[]`.

```
int *iptrtab[10];       /* array of 10 pointers to int */
char *dptrtab[10];      /* array of 10 pointers to char */

char *cptrfunc(void);   /* function returning ptr to char */
int  *iptrfunc(void);   /* function returning ptr to int */
```

- To declare pointers we need to use parentheses:

```
char (*crowptr)[10];    /* ptr to array of 10 characters */
int  (*irowptr)[10];    /* ptr to array of 10 integers */

int    (*iptr)(void);   /* ptr to func returning int */
double (*dptr)(void);   /* ptr to func returning double */
```

# CONSTRUCTING COMPLEX DECLARATIONS

- Break the English description of an identifier's type into pieces.
  - an array of
  - a pointer to
  - a function returning
  - the base type
- Decide on a declarator for each individual piece.
- Substitute the topmost for the identifier in the declarator beneath it.
- Parenthesize pointer declarators whenever we substitute them into an array or function declarator.
- When there's only one declaration left, add the base type.

# EXAMPLE OF CONSTRUCTING A DECLARATION

- Break it into pieces:

  *an array of*
  *pointers to*
  *functions taking no arguments and returning*
  **void**

- Decide on a declarator for each piece:

```
table[]
*ptr
func(void)
```

- Substitute topmost declarator in one underneath it:

```
*table[]
func(void)
```

- Repeat until only a single declarator remains:

```
(*table[])(void)
```

- Add the base type:

```
void (*table[])(void)
```

# UNDERSTANDING COMPLEX DECLARATIONS

- Take the type declaration and find its innermost declarator.
- Write down its type.
- Substitute an identifier for this innermost declarator.
- Ignore any parentheses surrounding it.
- Repeat the process until only a single identifier remains.
- Tricky part is locating the innermost declarator.
  - It's the identifier and an immediately following `[]` or `()`.
  - If neither of these is present, it's an immediately preceding `*`.

# EXAMPLE OF DECLARATION UNDERSTANDING

- What's the type of:

  `void (*table[])(void)`

- Locate the innermost declarator: **`table[]`**

  *An array of*

- Subsitute a variable for **`table[]`** and repeat:

  `void (*x)(void);`

- Innermost declarator: **`*x`**

  *pointers to*

- Subsitute a variable for **`(*x)`** and repeat:

  `void y(void);`

- Innermost declarator: **`y(void)`**

  *a function taking no arguments and returning* **`void`**.

# typedefs.h

(Figure 17.9 – Page 435)

```
/*
 * Useful typedefs for menu program.
 */
typedef char *STRING;
typedef void (*PFR_VOID)(void);
typedef PFR_VOID MENU_FUNC_PTR;
```

# usemenu3.c

(Figure 17.10 – Page 436)

```
/*
 * Print menu and then obtain and execute the user's choice.
 */
#include <stdio.h>
#include "record.h"
#include "typedefs.h"

const STRING MenuEntries[] =
{
  "Add record", "Delete record", "Print record"
};

const MENU_FUNC_PTR MenuFptrs[] =
{
  add_record, del_record, print_record
};

int main()
{
  void menu_process(int, const STRING [], const MENU_FUNC_PTR []);
  const int choices = sizeof(MenuFptrs)/sizeof(MenuFptrs[0]);

  menu_process(choices, MenuEntries, MenuFptrs);
  return 0;
}
```

# prmenu3.c

(Figure 17.11 – Page 437)

```
/*
 * Display menu and obtain and execute user choices.
 *   menu_process - obtain and execute user's choice.
 *   menu_choice - request and get the user's choice.
 */
#include <stdio.h>
#include "typedefs.h"

#define MAXLEN 80

static int menu_choice(int n, const STRING items[])
{
  int  getline(STRING buf, int buflen);
  int  i, num;                              /* index, selection */
  char line[MAXLEN + 1];

  printf("Your choices are:\n");
  for (i = 1; i <= n; i++)
    printf("\t%i) %s\n", i, items[i - 1]);
  printf("Or type a 0 to quit: ");
  while (getline(line, MAXLEN) != -1)
    if (sscanf(line, "%i", &num) == 1 && 0 <= num && num <= n)
      return num;
    else
      printf("Enter a value between 0 and %i: ", n);
  return 0;
}

void menu_process(int n,
                  const STRING items[],
                  const MENU_FUNC_PTR fptrs[])
{
  int choice;                                /* holds user's choice */

  while ((choice = menu_choice(n, items)) != 0)
    fptrs[choice - 1]();
}
```

# menu2.h

(Figure 17.12 – Page 438)

```
/*
 * Definition of a menu structure
 */
#include "typedefs.h"

typedef struct menu_item
{
  STRING entry;           /* String to write */
  MENU_FUNC_PTR fptr;     /* Function to execute */
} MENU_ITEM;

void menu_process(int, const MENU_ITEM []);
```

# prmenu4.c

(Figure 17.13 – Page 439)

```
/*
 * Display menu and obtain/execute user choices.
 *   menu_process - get and execute user's choice.
 *   menu_choice - read and verify choice from user.
 *   menu_search - search menu to find user's choice.
 */
#include <stdio.h>
#include <stddef.h>
#include <string.h>
#include "menu2.h"

#define MAXLEN 80

static const MENU_ITEM *menu_search(const MENU_ITEM *ptr,
                                    const MENU_ITEM *endptr,
                                    const STRING target)
{
  for (; ptr < endptr; ptr++)
    if (strncmp(target, ptr->entry, strlen(target)) == 0)
      return ptr;
  return NULL;
}
```

# prmenu4.c

(Figure 17.13 – Page 439)

```
static MENU_FUNC_PTR menu_choice(int n, const MENU_ITEM menu[])
{
  int   getline(STRING buf, int buflen);
  char  line[MAXLEN + 1];
  const MENU_ITEM *ptr;
  const MENU_ITEM *const endptr = menu + n;

  printf("Your choices are:\n");
  for (ptr = menu; ptr < endptr; ptr++)
    printf("\t%s\n", ptr->entry);
  printf("Or type QUIT to quit: ");
  while (getline(line, MAXLEN) != -1 && strcmp(line, "QUIT") != 0)
  {
    if ((ptr = menu_search(menu, endptr, line)) != NULL)
      return ptr->fptr;
    printf("Select an appropriate menu entry: ", n);
  }
  return NULL;
}

void menu_process(int n, const MENU_ITEM menu[])
{
  MENU_FUNC_PTR choiceptr;               /* user-selected function */

  while ((choiceptr = menu_choice(n, menu)) != NULL)
    choiceptr();                         /* execute user's choice */
}
```

# etoc.c

(Figure 17.14 – Page 440)

```
/*
 * Pseudo-English-to-C converter.
 */
#include <stdio.h>
#include <string.h>
#include <stddef.h>

#define MAXLEN    80

char declarator[MAXLEN * 5];          /* more than enough room */

int main()
{
  int  getline(char *buf, int len);
  void wrap(char *towrap, const char *frontptr, const char *backptr);
  char line[MAXLEN + 1];              /* input line */
  char desc[MAXLEN + 1];              /* description of type */
  char type[MAXLEN + 1];              /* base type */
  char *descptr;                      /* ptr to next piece */
```

```
  while (getline(line, MAXLEN) != -1)
    if (sscanf(line, "%s %s\n", desc, type) != 2)
      printf("Need description and type, ie \"pfp int\"\n");
    else
    {
      strcpy(declarator, "thing");
      for (descptr = desc; *descptr != '\0'; descptr++)
        switch (*descptr)
        {
          case 'p':  wrap(declarator, "*", NULL);
                     break;
          case 'f':  if (*declarator != '*')
                       wrap(declarator, NULL, "()");
                     else
                       wrap(declarator, "(", ")()");
                     break;
          case 'a':  if (*declarator != '*')
                       wrap(declarator, NULL, "[]");
                     else
                       wrap(declarator, "(", ")[]");
                     break;
        }
      printf("%s %s\n", type, declarator);
    }
  return 0;
}
```

# wrap.c

(Figure 17.15 – Page 441)

```
/*
 * Function to wrap one string inside two other strings.
 */
#include <stddef.h>
#include <string.h>

void wrap(char *string, const char *fstr, const char *bstr)
{
  char *endptr = string + strlen(string);  /* initial end */
  char *ptr = endptr;                      /* traversing ptr */
  int  flen;

  if (fstr != NULL)
  {
    flen = strlen(fstr);
    while (ptr-- > string)       /* move over to make room */
      *(ptr + flen) = *(ptr);
    while (*fstr != '\0')        /* do insert */
      *string++ = *fstr++;
    *(endptr += flen) = '\0';
  }
  if (bstr != NULL)              /* append trailing string */
    while ((*endptr++ = *bstr++) != '\0')
      ;
}
```

# CHAPTER SUMMARY

- C allows us to declare arbitrarily complex data types, including types such as arrays of pointers to functions, and functions that return pointers to functions.

- These type declarations appear in numerous places, including variable declarations, function headers, and type casts.

- There are algorithms we can use both to turn an English description of a type into its declaration and to turn a declaration into its English description.

- Alternatively, we can use **`typedef`** to make these types considerably easier to read and declare.