Debugging

Laboratorio di Programmazione I

Corso di Laurea in Informatica
A.A. 2018/2019

Argomenti del Corso

Ogni lezione consta di una spiegazione assistita da slide, e seguita da esercizi in classe

- Introduzione all'ambiente Linux
- Introduzione al C
- Tipi primitivi e costrutti condizionali
- Costrutti iterativi ed array
- Funzioni, stack e visibilità variabili
- Puntatori e memoria
- Debugging
- Tipi di dati utente
- Liste concatenate e librerie
- Ricorsione

Sommario

Origine del Termine

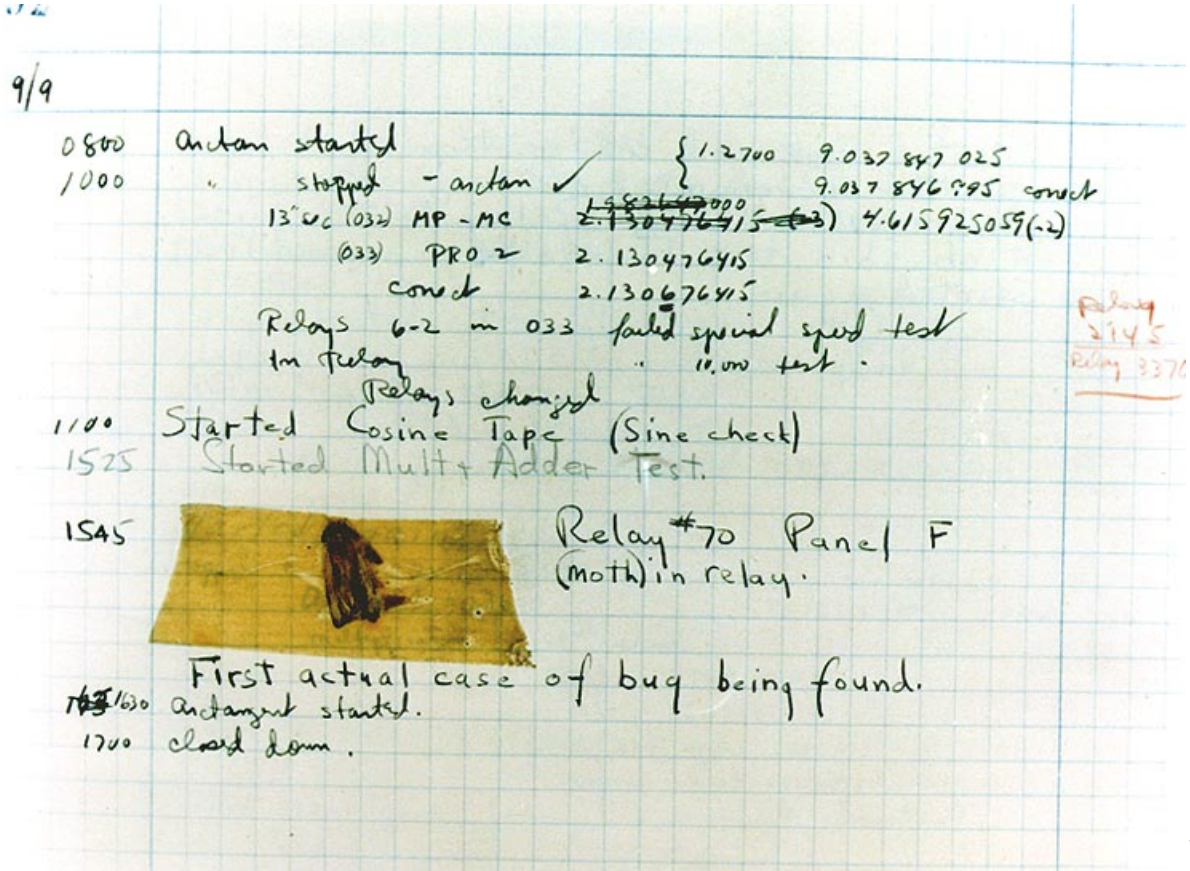
- Il termine bug è in uso nel gergo ingegneristico per indicare un difetto di fabbricazione già dal tardo 1800

Origine del Termine

- Il termine bug è in uso nel gergo ingegneristico per indicare un difetto di fabbricazione già dal tardo 1800
- La sua diffusione in ambito informatico si deve a Grace Hopper.. e ad una sfortunata falena

Origine del Termine

- Il termine bug è in uso nel gergo ingegneristico per indicare un difetto di fabbricazione già dal tardo 1800
- La sua diffusione in ambito informatico si deve a Grace Hopper.. e ad una sfortunata falena



Definizione

- Un Bug è un comportamento erroneo o inatteso di un pezzo di software

Definizione

- Un Bug è un comportamento erroneo o inatteso di un pezzo di software
- Causato da
  - Errori di tipo o di sintassi
  - Typos o errori minori
  - Errori di implementazione
  - Errori logici

Definizione

- Un Bug è un comportamento erroneo o inatteso di un pezzo di software
- Causato da
  - Errori di tipo o di sintassi
  - Typos o errori minori
  - Errori di implementazione
  - Errori logici
- Perchè il debugging è difficile?
  - Il sintomo non è necessariamente indicativo della causa
  - La riproducibilità può essere problematica
  - Presenza di errori multipli e correlati
  - Eliminare un bug può introdurne di nuovi

Un Approccio Pratico al Debugging

Debugging a tempo di compilazione

- Leggere ed interpretare i messaggi di errore

Un Approccio Pratico al Debugging

Debugging a tempo di compilazione

- Leggere ed interpretare i messaggi di errore

Debugging a tempo di esecuzione

- Progettare e comprendere il codice
- Bug ricorrenti ed errori tipici
- Debugging mediante stampe

Un Approccio Pratico al Debugging

Debugging a tempo di compilazione

- Leggere ed interpretare i messaggi di errore

Debugging a tempo di esecuzione

- Progettare e comprendere il codice
- Bug ricorrenti ed errori tipici
- Debugging mediante stampe

Argomenti avanzati (non per questo corso)

- Uso debugger (e.g. GDB)
- Software testing e controllo versionamento

Un Approccio Pratico al Debugging

Debugging a tempo di compilazione

- Leggere ed interpretare i messaggi di errore

Debugging a tempo di esecuzione

- Progettare e comprendere il codice
- Bug ricorrenti ed errori tipici
- Debugging mediante stampe

Argomenti avanzati (non per questo corso)

- Uso debugger (e.g. GDB)
- Software testing e controllo versionamento

I Bug e la Piattaforma di Autovalutazione

Progettazione

Pensare e progettare la soluzione invece di scriverla d'impulso aiuta a ridurre il tempo passato a debuggare il vostro codice

Progettazione

Pensare e progettare la soluzione invece di scriverla d'impulso aiuta a ridurre il tempo passato a debuggare il vostro codice

- Maggiore comprensione della logica che ispira l'implementazione

Progettazione

Pensare e progettare la soluzione invece di scriverla d'impulso aiuta a ridurre il tempo passato a debuggare il vostro codice

- Maggiore comprensione della logica che ispira l'implementazione
- Maggiore consapevolezza di cosa succede ad ogni passo di esecuzione

Progettazione

Pensare e progettare la soluzione invece di scriverla d'impulso aiuta a ridurre il tempo passato a debuggare il vostro codice

- Maggiore comprensione della logica che ispira l'implementazione
- Maggiore consapevolezza di cosa succede ad ogni passo di esecuzione
- Imparate a progettare su carta prima di scrivere codice: e.g. operazioni sulle liste concatenate

Capire il Codice

Saper eseguire un programma a mente o con carta e penna è essenziale per imparare a scovare i Bug

Capire il Codice

Saper eseguire un programma a mente o con carta e penna è essenziale per imparare a scovare i Bug

- Sapere quanto vale una variabile dopo l'esecuzione di un comando

Capire il Codice

Saper eseguire un programma a mente o con carta e penna è essenziale per imparare a scovare i Bug

- Sapere quanto vale una variabile dopo l'esecuzione di un comando
- Conoscere il valore di una espressione (logica o algebrica)

Capire il Codice

Saper eseguire un programma a mente o con carta e penna è essenziale per imparare a scovare i Bug

- Sapere quanto vale una variabile dopo l'esecuzione di un comando
- Conoscere il valore di una espressione (logica o algebrica)
- Capire sotto quali condizioni viene eseguito un comando condizionale o un iteratore

Capire il Codice

Saper eseguire un programma a mente o con carta e penna è essenziale per imparare a scovare i Bug

- Sapere quanto vale una variabile dopo l'esecuzione di un comando
- Conoscere il valore di una espressione (logica o algebrica)
- Capire sotto quali condizioni viene eseguito un comando condizionale o un iteratore
- Tenere sotto controllo i riferimenti tra porzioni della memoria prodotti dai puntatori

Programmare Pulito

Una corretta indentazione e il rispetto delle regole stilistiche che vi sono state spiegate aiuta a comprendere più facilmente il codice

Programmare Pulito

Una corretta indentazione e il rispetto delle regole stilistiche che vi sono state spiegate aiuta a comprendere più facilmente il codice

```
float* range_righe(float** matrice, int righe, int
    colonne, float[] range){
float max, min;
int i,j;
for(i=0;i<righe;i++){max=matrice[i][0];
min=matrice[i][0];
for(j=1;j<colonne;j++){ if(matrice[i][j]<min){ min=
    matrice[i][j];
}else{if(matrice[i][j]>max){max=matrice[i][j];}
}
range[i]=max-min;}
return range;
}
```

Programmare Pulito

Una corretta indentazione e il rispetto delle regole stilistiche che vi sono state spiegate aiuta a comprendere più facilmente il codice

```
float* range_righe(float** matrice, int righe, int
    colonne, float[] range){
float max, min;
int i,j;
for(i=0;i<righe;i++){max=matrice[i][0];
min=matrice[i][0];
for(j=1;j<colonne;j++){ if(matrice[i][j]<min){ min=
    matrice[i][j];
}else{if(matrice[i][j]>max){max=matrice[i][j];}
}
range[i]=max-min;}
return range;
}
```

```
tmp.c:3:69: error: expected ';', ',' or ')' before 'range'
 float* range_righe(float** matrice, int righe, int colonne,...
  ...float[] range){
```

Test: Condizioni logiche

Che cosa stampa questo pezzo di codice?

```
int i=0;
if (i=0)
   printf("Zero");
else
   printf("Uno");
```

Test: puntatori

```
int main() {       int a= 10, b;   int *p, *q;
    printf("Il contenuto di a e' %d\n",a);
    p = &a;
    printf("L'indirizzo di a, cioe' p e' %p\n",p);
    q=p;
    printf("Dopo q = p; q = %p\n", q);
    *p = 15;
    printf("*p = %d\n", *p);
    *q = 27;
    printf("*q = %d inoltre adesso *p = %d\n", *q, *p);
    a = 10;                b = 15;
    printf("Contenuto di a e' %d e di b e' %d\n",a,b);
    p = &a;
    printf("Indirizzo di a, cioe' p e' %p\n",p);
    q = &b;
    printf("Indirizzo di b, cioe' q e' %p\n",q);
    *q = *p;
    printf("Dopo *q = *p; *q = %d\n", *q);
    printf("Contenuto di a e' %d e di b e' %d\n",a,b);
}
```

Test: puntatori e array

```
int main() {
    float V[8]={1.1,2.2,3.3,4.4,5.5,6.6,7.7,8.8};
    int k;
    float *p, *q, *r;
    r = &V[1]
    p=V+7;
    q=p-2;
    k=p-q;
    printf("%f,\t%f,\t%d\n",*p, *q, k);
}
```

Debugging di errori a tempo di compilazione

- Tendenzialmente più facile del debugging degli errori a run-time
- A patto che sappiate interpretare i messaggi di errore

Debugging di errori a tempo di compilazione

- Tendenzialmente più facile del debugging degli errori a run-time
- A patto che sappiate interpretare i messaggi di errore

Chi genera questo errore?

```
int main(void)
/* Print "hello, world" to stdout and return 0. /*
{
printf("hello, world\n");
return 0;
}

tmp.c:3:1: error: unterminated comment
 /* Print "hello, world" to stdout and return 0. /*
 ^
tmp.c: In function ‘main’:
tmp.c:2:1: error: expected ‘{’ at end of input
 int main(void)
 ^
```

Debugging di errori a tempo di compilazione

- Tendenzialmente più facile del debugging degli errori a run-time
- A patto che sappiate interpretare i messaggi di errore

Chi genera questo errore?

Debugging di errori a tempo di compilazione

- Tendenzialmente più facile del debugging degli errori a run-time
- A patto che sappiate interpretare i messaggi di errore

Chi genera questo errore?

```
int main(void)
/* Print "hello, world" to stdout and return 0. */
{
printf("hello, world\n");
returrn 0;
}
```

```
tmp.c: In function ‘main’:
tmp.c:6:1: error: ‘returrn’ undeclared (first use in this function)
 returrn 0;
tmp.c:6:1: note: each undeclared identifier is reported only once for
tmp.c:6:9: error: expected ‘;’ before numeric constant
```

Debugging di errori a tempo di compilazione

- Tendenzialmente più facile del debugging degli errori a run-time
- A patto che sappiate interpretare i messaggi di errore

Chi genera questo errore?

Debugging di errori a tempo di compilazione

- Tendenzialmente più facile del debugging degli errori a run-time
- A patto che sappiate interpretare i messaggi di errore

Chi genera questo errore?

```
int main(void)
/* Print "hello, world" to stdout and return 0. */
{
prinf("hello, world\n");
return 0;
}
```

```
/tmp/cc2t8xpz.o: In function ‘main’:
tmp.c:(.text+0xf): undefined reference to ‘prinf’
collect2: error: ld returned 1 exit status
```

Attenzione ai messaggi di GCC

- Capire quale componente sta generando il messaggio restringe il campo dei possibili errori

Attenzione ai messaggi di GCC

- Capire quale componente sta generando il messaggio restringe il campo dei possibili errori
- Il messaggio riporta la posizione nel file in corrispondenza della quale è stato generato l'errore

Attenzione ai messaggi di GCC

- Capire quale componente sta generando il messaggio restringe il campo dei possibili errori
- Il messaggio riporta la posizione nel file in corrispondenza della quale è stato generato l'errore
- Non sempre il bug si trova in quella posizione
  - Quello è solo il punto in cui GCC si è confuso

Attenzione ai messaggi di GCC

- Capire quale componente sta generando il messaggio restringe il campo dei possibili errori
- Il messaggio riporta la posizione nel file in corrispondenza della quale è stato generato l'errore
- Non sempre il bug si trova in quella posizione
  - Quello è solo il punto in cui GCC si è confuso
  - L'errore comunque si troverà...?

Attenzione ai messaggi di GCC

- Capire quale componente sta generando il messaggio restringe il campo dei possibili errori
- Il messaggio riporta la posizione nel file in corrispondenza della quale è stato generato l'errore
- Non sempre il bug si trova in quella posizione
  - Quello è solo il punto in cui GCC si è confuso
  - L'errore comunque si troverà...?
- Un errore maschera quello successivo

Errori ricorrenti

Esperienza ed esercizio sviluppano l'occhio per gli errori ricorrenti

```
int giorno, pari;
...
switch (giorno) {
      case 1:
       printf("Lunedi\n");
       pari=0;
       break;
       case 2:
       printf("Martedi\n");
       pari=1;
       case 3:
       printf("Mercoledi\n");
       pari=0;
       break;
    ...
}
```

Errori ricorrenti

Esperienza ed esercizio sviluppano l'occhio per gli errori ricorrenti

```
int i, j, a[15];
...
if (i = 15) {
  for (j=0; j<=i; j++) scanf("%d", a[i]);
};
```

Errori ricorrenti

Esperienza ed esercizio sviluppano l'occhio per gli errori ricorrenti

```
int i, j, a[15];
...
if (i = 15) {
  for (j=0; j<=i; j++) scanf("%d", a[i]);
};
```

```
if (0 < i < 15) {
  for (j=0; j<=i; j++) scanf("%d", a[i]);
};
```

Errori ricorrenti

Esperienza ed esercizio sviluppano l'occhio per gli errori ricorrenti

```
int i, j, a[15];
...
if (i = 15) {
  for (j=0; j<=i; j++) scanf("%d", a[i]);
};
```

```
if (0 < i < 15) {
  for (j=0; j<=i; j++) scanf("%d", a[i]);
};
```

```
if (i & j) {
  ...
};
```

Errori ricorrenti

Esperienza ed esercizio sviluppano l'occhio per gli errori ricorrenti

```
float mengoli(int N){
␣int i;
␣float serie;
␣for (i=0;i<N;i++) serie+=1/(i*(i+1));
␣return serie;
}
```

Usare stampe (a schermo) per il debugging

Impariamo ad usare i messaggi sullo schermo per determinare se il codice sta facendo quello che ci aspettiamo

Usare stampe (a schermo) per il debugging

Impariamo ad usare i messaggi sullo schermo per determinare se il codice sta facendo quello che ci aspettiamo

- Come stampare?
- Quando stampare?
- Che cosa stampare?

Come Stampare?

- La prima risposta che vi dovrebbe venire in mente:

Come Stampare?

- La prima risposta che vi dovrebbe venire in mente:
  - `printf("Bug qui?");`

Come Stampare?

- La prima risposta che vi dovrebbe venire in mente:
  - `printf("Bug qui?");`
  - Male: La printf è bufferizzata quindi l'output potrebbe non essere mostrato anche se il programma scoppia dopo la printf.

Come Stampare?

- La prima risposta che vi dovrebbe venire in mente:
  - `printf("Bug qui?");`
  - Male: La printf è bufferizzata quindi l'output potrebbe non essere mostrato anche se il programma scoppia dopo la printf.
- `printf("Bug qui?\n");`

Come Stampare?

- La prima risposta che vi dovrebbe venire in mente:
  - `printf("Bug qui?");`
  - Male: La printf è bufferizzata quindi l'output potrebbe non essere mostrato anche se il programma scoppia dopo la printf.
- `printf("Bug qui?\n");`
  - Poco meglio: Il newline provoca lo svuotamento del buffer, ma non immediato.

Come Stampare?

- La prima risposta che vi dovrebbe venire in mente:
  - `printf("Bug qui?");`
  - Male: La printf è bufferizzata quindi l'output potrebbe non essere mostrato anche se il programma scoppia dopo la printf.
- `printf("Bug qui?\n");`
  - Poco meglio: Il newline provoca lo svuotamento del buffer, ma non immediato.
- `printf("Bug qui?\n"); fflush(stdout);`

Come Stampare?

- La prima risposta che vi dovrebbe venire in mente:
  - `printf("Bug qui?");`
  - Male: La printf è bufferizzata quindi l'output potrebbe non essere mostrato anche se il programma scoppia dopo la printf.
- `printf("Bug qui?\n");`
  - Poco meglio: Il newline provoca lo svuotamento del buffer, ma non immediato.
- `printf("Bug qui?\n"); fflush(stdout);`
  - Meglio: fflush forza lo svuotamento del buffer.

Come Stampare?

- La prima risposta che vi dovrebbe venire in mente:
  - `printf("Bug qui?");`
  - Male: La printf è bufferizzata quindi l'output potrebbe non essere mostrato anche se il programma scoppia dopo la printf.
- `printf("Bug qui?\n");`
  - Poco meglio: Il newline provoca lo svuotamento del buffer, ma non immediato.
- `printf("Bug qui?\n"); fflush(stdout);`
  - Meglio: fflush forza lo svuotamento del buffer.
- `fprintf(stderr, "Bug qui?\n");`

Come Stampare?

- La prima risposta che vi dovrebbe venire in mente:
  - `printf("Bug qui?");`
  - Male: La printf è bufferizzata quindi l'output potrebbe non essere mostrato anche se il programma scoppia dopo la printf.
- `printf("Bug qui?\n");`
  - Poco meglio: Il newline provoca lo svuotamento del buffer, ma non immediato.
- `printf("Bug qui?\n"); fflush(stdout);`
  - Meglio: fflush forza lo svuotamento del buffer.
- `fprintf(stderr, "Bug qui?\n");`
  - Ancora meglio: lo stderr non è buffered.

Quando Stampare?

- Utilizzate le stampe per capire se il flusso di controllo del programma è quello che vi aspettare

Quando Stampare?

- Utilizzate le stampe per capire se il flusso di controllo del programma è quello che vi aspettare
- Esecuzione dei rami `then` ed `else` del comando condizionale

```
if (condizione_difficilissima) {
    fprintf(stderr,"Ramo Then");
    ...
} else {
    fprintf(stderr,"Ramo Else");
    ...
}
```

Quando Stampare?

- Utilizzate le stampe per capire se il flusso di controllo del programma è quello che vi aspettare
- Esecuzione dei rami `then` ed `else` del comando condizionale

```
if (condizione_difficilissima) {
    fprintf(stderr,"Ramo Then");
    ...
} else {
    fprintf(stderr,"Ramo Else");
    ...
}
```

- Per controllare ingresso ed uscita dai cicli

```
fprintf(stderr,"Prima del while");
while (condizione_sempre_vera) {
    fprintf(stderr,"Dentro while");
    ...
}
fprintf(stderr,"Dopo il while");
```

Quando Stampare?

- Cicli e comandi condizionali possono essere annidati

```
fprintf(stderr,"Prima del while 1");
while (condizione1) {
    fprintf(stderr,"-> Dentro while 1");
        while (condizione2) {
         fprintf(stderr,"---> Dentro while 2");
            ...
        }
        fprintf(stderr,"-> Dopo while 1");
        if (condizione3) {
          fprintf(stderr,"---> Dentro Then");
        }
        ...
}
fprintf(stderr,"Dopo while 1");
```

Quando Stampare?

- Cicli e comandi condizionali possono essere annidati

```
fprintf(stderr,"Prima del while 1");
while (condizione1) {
    fprintf(stderr,"-> Dentro while 1");
        while (condizione2) {
         fprintf(stderr,"---> Dentro while 2");
            ...
        }
        fprintf(stderr,"-> Dopo while 1");
        if (condizione3) {
          fprintf(stderr,"---> Dentro Then");
        }
        ...
}
fprintf(stderr,"Dopo while 1");
```

- Evitate di stampare troppo o non ci capirete nulla: procedete al debugging in maniera incrementale

Cosa Stampare?

- Il contenuto delle variabili per assicurarsi che sia quello che vi aspettate

```
int i, N=10, a[N];
...
for (i=N;i>0; i--) {
    fprintf(stderr,"Valore a[%d] = %d",i,a[i]);
        ...
}
```

Cosa Stampare?

- Il contenuto delle variabili per assicurarsi che sia quello che vi aspettate

```
int i, N=10, a[N];
...
for (i=N;i>0; i--) {
    fprintf(stderr,"Valore a[%d] = %d",i,a[i]);
        ...
}
```

- Il valore delle variabili coinvolte in guardie condizionali o di comandi iterativi

```
printf("Guardia: %d, i: %d",0 < i < 15,i);
if (0 < i < 15) {
   ...
 };
```

Cosa Stampare?

- Le variabili puntatore a lista per diagnosticare problemi di memoria

```
ListaDiElementi curr = head;
...
while (condizione_per_scorrere_la_lista) {
 if (curr == NULL) fprintf(stderr,"Curr->NULL!!!");
  curr->info = abs(curr->info);
}
```

Cosa Stampare?

- Le variabili puntatore a lista per diagnosticare problemi di memoria

```
ListaDiElementi curr = head;
...
while (condizione_per_scorrere_la_lista) {
 if (curr == NULL) fprintf(stderr,"Curr->NULL!!!");
  curr->info = abs(curr->info);
}
```

- Altre cose interessanti da stampare:

Cosa Stampare?

- Le variabili puntatore a lista per diagnosticare problemi di memoria

```
ListaDiElementi curr = head;
...
while (condizione_per_scorrere_la_lista) {
 if (curr == NULL) fprintf(stderr,"Curr->NULL!!!");
  curr->info = abs(curr->info);
}
```

- Altre cose interessanti da stampare:
  - Parametri attuali delle funzioni

Cosa Stampare?

- Le variabili puntatore a lista per diagnosticare problemi di memoria

```
ListaDiElementi curr = head;
...
while (condizione_per_scorrere_la_lista) {
 if (curr == NULL) fprintf(stderr,"Curr->NULL!!!");
  curr->info = abs(curr->info);
}
```

- Altre cose interessanti da stampare:
  - Parametri attuali delle funzioni
  - Valori delle variabili modificate in un ciclo e usate dopo l'esecuzione del ciclo

Cosa Stampare?

- Le variabili puntatore a lista per diagnosticare problemi di memoria

```
ListaDiElementi curr = head;
...
while (condizione_per_scorrere_la_lista) {
 if (curr == NULL) fprintf(stderr,"Curr->NULL!!!");
  curr->info = abs(curr->info);
}
```

- Altre cose interessanti da stampare:
  - Parametri attuali delle funzioni
  - Valori delle variabili modificate in un ciclo e usate dopo l'esecuzione del ciclo
  - Contenuto dei campi info degli elementi di lista su cui operate (prima e dopo l'operazione)

Cosa Stampare?

- Le variabili puntatore a lista per diagnosticare problemi di memoria

```
ListaDiElementi curr = head;
...
while (condizione_per_scorrere_la_lista) {
 if (curr == NULL) fprintf(stderr,"Curr->NULL!!!");
  curr->info = abs(curr->info);
}
```

- Altre cose interessanti da stampare:
  - Parametri attuali delle funzioni
  - Valori delle variabili modificate in un ciclo e usate dopo l'esecuzione del ciclo
  - Contenuto dei campi info degli elementi di lista su cui operate (prima e dopo l'operazione)
  - Contenuto delle aree di memoria soggette a dereferenziazione (per vedere se state effettivamente accedendo al contenuto o ad un puntatore)

La Piattaforma di Autovalutazione (PdA)

- Se la PdA vi da un errore:

La Piattaforma di Autovalutazione (PdA)

- Se la PdA vi da un errore:
  - Identificate i test-case errati: possono essere tutti o solo una parte

La Piattaforma di Autovalutazione (PdA)

- Se la PdA vi da un errore:
  - Identificate i test-case errati: possono essere tutti o solo una parte
  - Scaricate i test case e provate in locale quelli con errori usando le tecniche di debug viste oggi

La Piattaforma di Autovalutazione (PdA)

- Se la PdA vi da un errore:
  - Identificate i test-case errati: possono essere tutti o solo una parte
  - Scaricate i test case e provate in locale quelli con errori usando le tecniche di debug viste oggi
  - I test case spesso testano le condizioni limite: e.g. lista vuota, inserimento di molti elementi, inserimenti in testa o in lista vuota, etc.

La Piattaforma di Autovalutazione (PdA)

- Se la PdA vi da un errore:
  - Identificate i test-case errati: possono essere tutti o solo una parte
  - Scaricate i test case e provate in locale quelli con errori usando le tecniche di debug viste oggi
  - I test case spesso testano le condizioni limite: e.g. lista vuota, inserimento di molti elementi, inserimenti in testa o in lista vuota, etc.
- In locale mi funziona tutto ma la PdA mi da errore (a.k.a. la maestra ce l'ha con me!)

La Piattaforma di Autovalutazione (PdA)

- Se la PdA vi da un errore:
  - Identificate i test-case errati: possono essere tutti o solo una parte
  - Scaricate i test case e provate in locale quelli con errori usando le tecniche di debug viste oggi
  - I test case spesso testano le condizioni limite: e.g. lista vuota, inserimento di molti elementi, inserimenti in testa o in lista vuota, etc.
- In locale mi funziona tutto ma la PdA mi da errore (a.k.a. la maestra ce l'ha con me!)
  - Rassegnatevi: c'è un errore

La Piattaforma di Autovalutazione (PdA)

- Se la PdA vi da un errore:
  - Identificate i test-case errati: possono essere tutti o solo una parte
  - Scaricate i test case e provate in locale quelli con errori usando le tecniche di debug viste oggi
  - I test case spesso testano le condizioni limite: e.g. lista vuota, inserimento di molti elementi, inserimenti in testa o in lista vuota, etc.
- In locale mi funziona tutto ma la PdA mi da errore (a.k.a. la maestra ce l'ha con me!)
  - Rassegnatevi: c'è un errore
  - Tipicamente è dovuto ad un utilizzo errato della memoria che in locale non si mostra perchè avendo molta memoria a disposizione non generate il segmentation fault (ma sulla PdA si)

La Piattaforma di Autovalutazione (PdA)

- Se la PdA vi da un errore:
  - Identificate i test-case errati: possono essere tutti o solo una parte
  - Scaricate i test case e provate in locale quelli con errori usando le tecniche di debug viste oggi
  - I test case spesso testano le condizioni limite: e.g. lista vuota, inserimento di molti elementi, inserimenti in testa o in lista vuota, etc.
- In locale mi funziona tutto ma la PdA mi da errore (a.k.a. la maestra ce l'ha con me!)
  - Rassegnatevi: c'è un errore
  - Tipicamente è dovuto ad un utilizzo errato della memoria che in locale non si mostra perchè avendo molta memoria a disposizione non generate il segmentation fault (ma sulla PdA si)
  - A volte generato da utilizzo di un gergo non ANSI-C: e.g. niente dichiarazioni C++ style nelle guardie dei for